OPERAČNÍ SYSTÉMY 1

CYRIL KLIMEŠ

ČÍSLO OPERAČNÍHO PROGRAMU: CZ.1.07
NÁZEV OPERAČNÍHO PROGRAMU:
OP VZDĚLÁVÁNÍ PRO KONKURENCESCHOPNOST

TVORBA DISTANČNÍCH VZDĚLÁVACÍCH MODULŮ PRO CELOŽIVOTNÍ VZDĚLÁVÁNÍ DLE § 60 ZÁKONA Č. 111/1998 SB. O VŠ NA PŘÍRODOVĚDECKÉ FAKULTĚ OSTRAVSKÉ UNIVERZITY

REGISTRAČNÍ ČÍSLO PROJEKTU: CZ.1.07/3.2.07/02.0033

OSTRAVA 2013

Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky

Recenzenti:

Prof. Ing. Radim Farana, CSc.
RNDr. Bogdan Walek, Ph.D.

Název: Operační systémy 1
Autor: doc. Ing. Cyril Klimeš, CSc.
Vydání: první, 2013
Počet stran: 218

Studijní materiály jsou určeny pro distanční kurz: Operační systémy 1

Jazyková korektura nebyla provedena, za jazykovou stránku odpovídá autor.

OBSAH

Předmluva

Současné počítače a jejich operační systémy se vyvíjejí obrovským tempem. Jejich princip funkce je však stále stejný a vychází ze základních koncepcí a principů vzniklých ve druhé polovině minulého století. Předložená publikace se zabývá problematikou těch principů výstavby počítačů a operačních systémů, které jsou platné u současných počítačových systémů zpravidla označovaných jako osobní počítače.

Publikace vznikla na základě zjištění, že v současnosti na knižním trhu takový koncentrovaný souhrn poznatků není k dispozici. Nezabývá se podrobnostmi s řešením konkrétních počítačů a operačních systémů, ale zobecněním jejich principů.

Tato publikace je určena všem zájemcům o problematiku principů výstavby operačních systémů s doplněním potřebných základů počítačové architektury. Zájemce v této publikaci najde algoritmy a prostředky používané u operačních systémů. Text uvádí čtenáře do terminologie a běžných principů výstavby počítačů a operačních systémů a ulehčí studium specializované literatury.

Autor

1 Principy činnosti práce číslicového počítače

V této kapitole se dozvíte:

- Z jakých podsystémů se skládá číslicový počítač?
- Jaké jsou hlavní funkce jednotlivých podsystémů počítače?
- Jak se zobrazují jednotlivé informace v počítači?
- Jakými metodami se určují operandy v instrukcích počítače?

Po jejím prostudování byste měli být schopni:

- Charakterizovat činnost počítače, rozumět samočinnému způsobu zpracování programů v číslicovém počítači.
- Znát způsob zobrazování jednotlivých typů informací ukládaných v paměti počítače. Porozumět dvojkové soustavě a základním operacím v této soustavě, pochopit zobrazování v pevné a pohyblivé řádové čárce.
- Popsat různé metody určování operandů instrukcí což je velmi důležité pro pochopení některých algoritmů operačních systémů.

Klíčová slova této kapitoly:

Von Neumannovo schéma počítače, operační paměť, operační jednotka, řadič, vstupní a výstupní zařízení, instrukční cyklus řadiče, dvojková soustava, instrukce, operand.

Doba potřebná ke studiu: 6 hodin

Průvodce studiem

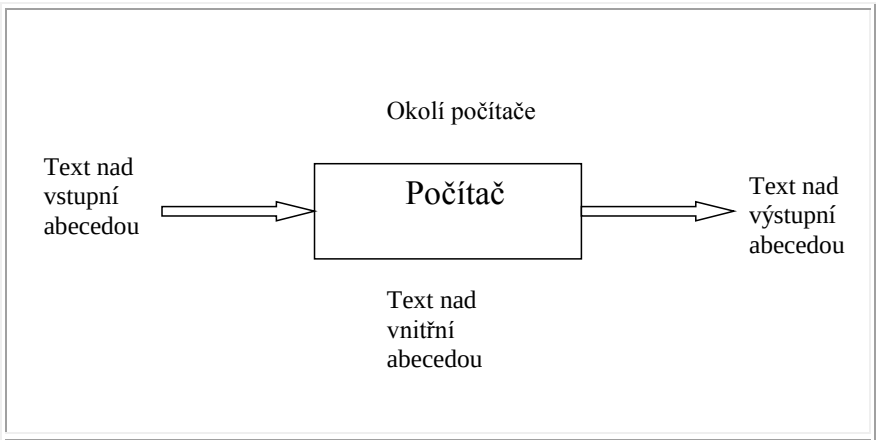
Studium této kapitoly je poměrně náročné zejména pro ty z Vás, kteří dosud nemají žádné znalosti z oblasti architektury číslicových počítačů. V takovém případě Vám zřejmě některé principy činnosti počítače budou připadat obtížně pochopitelné, ovšem nenechte se tím odradit, neboť pochopením této části se Vám usnadní studium následujících kapitol.
Na studium této části si vyhraďte alespoň 6 hodin. Doporučujeme studovat s přestávkami vždy po pochopení jednotlivých podkapitol. Po celkovém prostudování a vyřešení všech příkladů doporučujeme dát si pauzu, třeba 1 den, a pak se pusťte do vypracování korespondenčních úkolů.

1.1 Úvod

Číslicové počítače zpravidla chápeme jako stroje na zpracování informace. Obecně počítače jsou především matematickými stroji, jejichž úkolem je transformace číselných (diskrétních) vstupních hodnot, vstupních údajů. Jsou dále stroji elektronickými, což určuje fyzikální podmínky realizace úkolů transformace číselných údajů. Konečně jsou stroji samočinnými, což znamená, že uvedený úkol realizují bez přímé účasti člověka. Základem pro samočinnost funkce je existence paměti počítače, v níž je uchován návod pro řešení (program) a řídící jednotky, která dokáže tento návod interpretovat v postup.

Činnost počítače lze charakterizovat principiálně takto:

- Počítač přijímá na svém vstupu texty nad vstupní abecedou, transformuje je na texty nad vnitřní abecedou (kódování vstupní informace do vnitřní reprezentace) a ukládá je do paměti.
- V paměti provede zpracování.
- Zpracované texty transformuje do výstupní abecedy (interpretace informace zachycené ve vnitřní reprezentaci) a vydává je na svém výstupu.



Vazba počítače s okolím

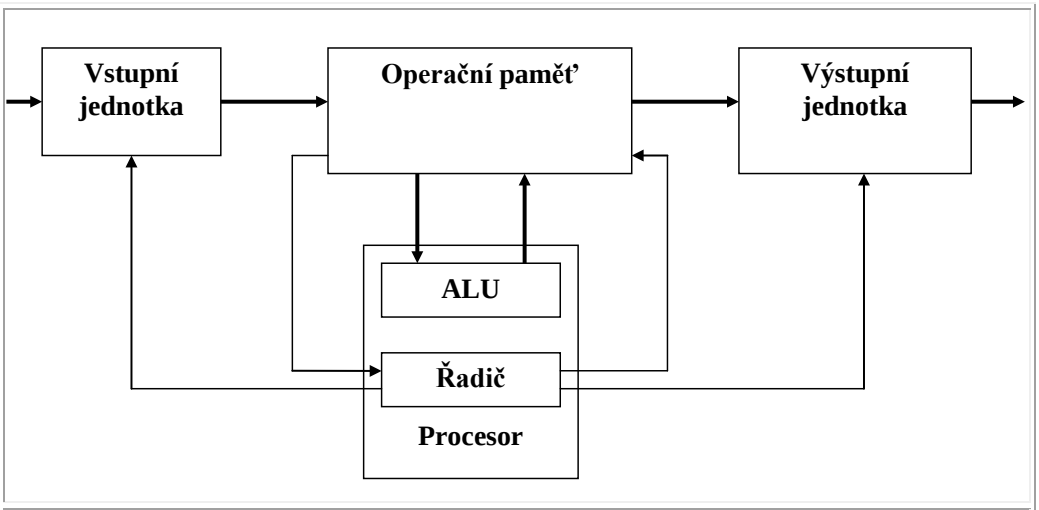
K tomu, aby počítač mohl komunikovat s okolím je nutný společný jazyk počítače a jeho okolí. Jedná se o jazyk, kterému obě strany jednoznačně a shodně rozumějí, tzv. formální jazyk. Vymezíme tři jeho určující složky:

- **abecedu**, tj. výběr přípustných základních symbolů,
- **syntaxi**, skladbu, tj. soubor pravidel, podle nichž lze ze symbolů abecedy tvořit přípustné vyšší tvary (výrazy, slova, věty),
- **sémantiku,** tj. význam, který shodně obě komunikující strany syntaktickým tvarům přiřazují.

Poznamenejme zde, že i počítače s grafickým uživatelským rozhraním pracují shodně. Jejich řídícím jazykem může být kliknutí myši nad ovládacím prvkem s následnou akcí, např. rozbalení nabídky či zavření okna.

1.2 Von Neumannovo schéma počítače

Von Neumannovo schéma bylo navrženo roku 1945 americkým matematikem (narozeným v Maďarsku) Johnem von Neumannem jako model samočinného počítače. Tento model s jistými obměnami zůstal zachován dodnes.



Von Neumannovo schéma

Podle tohoto schématu se počítač skládá z pěti hlavních modulů:

- **Operační paměť**: slouží k uchování zpracovávaného programu, zpracovávaných dat a výsledků výpočtu.
- **ALU – Arithmetical and logical Unit (aritmetickologická jednotka)**: jednotka provádějící veškeré aritmetické výpočty a logické operace. Obsahuje sčítačky, násobičky (pro aritmetické výpočty) a komparátory (pro porovnávání).
- **Řadič**: řídící jednotka, která řídí činnost všech částí počítače. Toto řízení je prováděno pomocí **řídících signálů**, které jsou zasílány jednotlivým modulům. Reakce na řídící signály, stavy jednotlivých modulů, jsou naopak zasílány zpět řadiči pomocí **stavových hlášení.**
- **Vstupní jednotka**: zařízení určená pro vstup programu a dat.
- **Výstupní jednotka**: zařízení určené pro výstup výsledků, které počítač zpracoval.

Ve von Neumannově schématu je možné ještě vyznačit další modul vzniklý spojením předcházejících modulů. Jedná se o **procesor (CPU - Central Processing Unit - centrální procesorová jednotka),** který vznikne spojením řadiče a aritmetickologické jednotky.

Princip činnosti počítače podle von Neumannova schématu:

1. Do operační paměti se pomocí vstupních zařízení přes ALU umístí program, který bude provádět výpočet.
2. Stejným způsobem se do operační paměti umístí data, která bude program zpracovávat.
3. Proběhne vlastní výpočet, jehož jednotlivé kroky provádí ALU. Tato jednotka je v průběhu výpočtu spolu s ostatními moduly řízena řadičem počítače. Mezivýsledky výpočtu jsou ukládány do operační paměti a nebo registrů procesoru.
4. Po skončení výpočtu jsou výsledky poslány přes ALU na výstupní zařízení.

Z hlediska systémového se počítač skládá z:

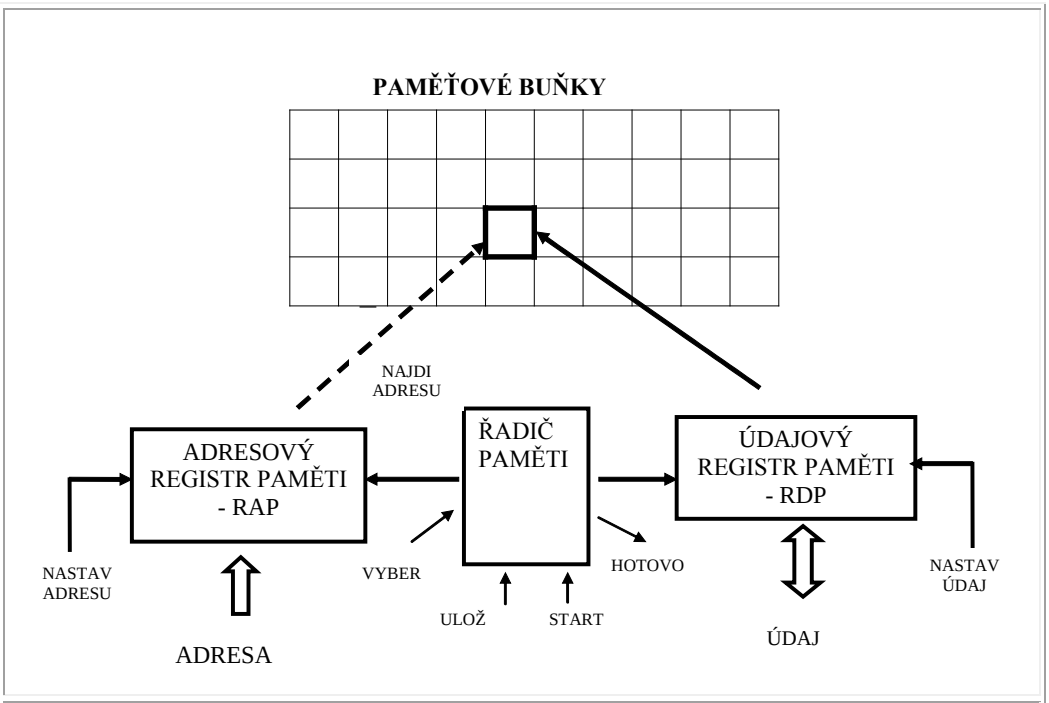
- paměťového podsystému,
- operačního podsystému,

- řídícího podsystému,
- vstupního a výstupního podsystému.

Centrálním podsystémem je paměť. Přejímá informace od vstupního podsystému a předává je výstupnímu podsystému. Řídící podsystém (řadič) získává z paměti instrukce programu a přiděluje ostatním podsystémům postupně úkoly. Operační podsystém získává podle pokynů řídícího podsystému z paměťového podsystému data, zpracovává je a vrací zpět do paměti.

Paměťový podsystém je schopen si zapamatovat, tj. uchovat beze změny určité množství informací. Podsystém tvoří:

- blok adresovatelných paměťových buněk,
- adresový registr paměti (registr adresy paměti – RAP),
- údajový registr paměti (registr dat paměti – RDP),
- řadič paměti.



Paměťový podsystém počítače

Funkce každé paměťové buňky je dána následujícími předpoklady:

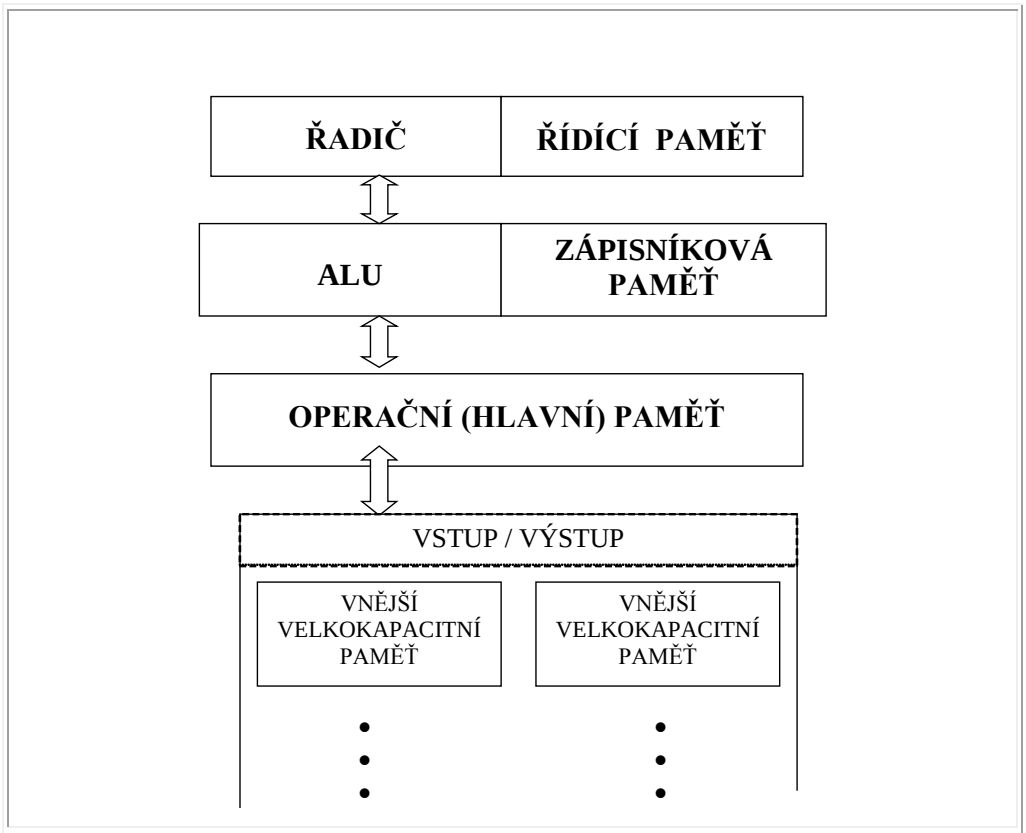
- Obsahuje jedno slovo – slabiku – byte.
- Při získání údaje čtením se obsah neporuší.
- Lze do ní zapsat novou informaci s podmínkou, že se zruší existující informace.
- Má jednoznačně přidělenou adresu.

Adresový registr paměti (RAP) uchovává adresu buňky, se kterou paměť v daném okamžiku pracuje. Tyto adresy získává z některých jiných podsystémů. Na základě signálu „NASTAV ADRESU“ od zdrojového podsystému se paralelně (najednou celá adresa) zapíše do RAP. Údajový registr paměti (RDP) uchovává data. Při výběru informace z paměti zde řadič paměti dočasně uchovává údaj z buňky, na kterou odkazuje adresový registr paměti. Údaj zde uchovaný je dostupný pro žádající podsystém v okamžiku, kdy řadič vyšle signál "HOTOVO". Při ukládání informace do

paměti se údaj ze zdrojového podsystému posílá do registru dat paměti na základě signálu "NASTAV ÚDAJ". Jakmile se nalezne místo, na které odkazuje adresa v adresovaném registru paměti, odešle se řadičem paměti obsah registru dat do této buňky.

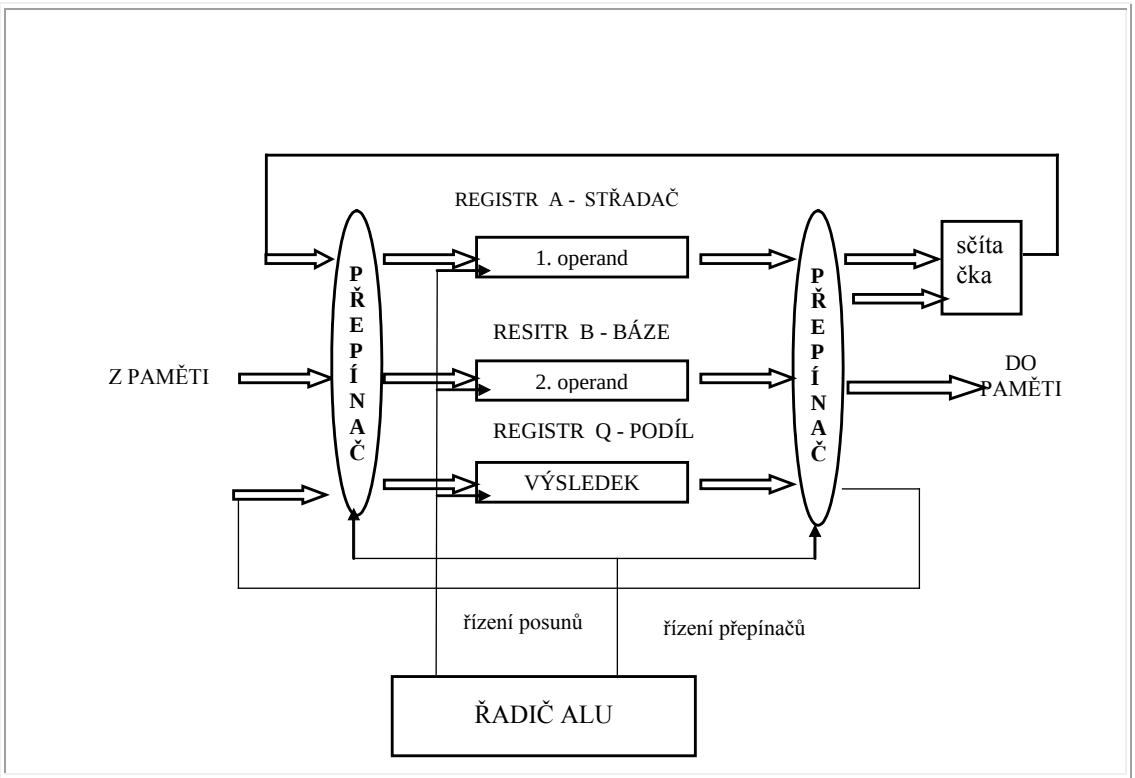
Řadič paměti řídí cyklus paměti. Jednotka požadující spolupráci s pamětí informuje, zda jde o výběr nebo vkládání informace, při čemž je do RAP uložena adresa korespondující buňky. Při ukládání do paměti se naplní RDP požadovanou informací. Řadič paměti uzavře přístup k oběma registrům RAP i RDP, najde buňku a sleduje tok informace mezi registrem dat paměti a zvolenou buňkou. Jakmile je úkol paměti hotov, řadič paměti generuje signál o ukončení "HOTOVO" a může začít další cyklus paměti.

Požadavky, které se v počítačovém systému kladou na paměť se nedají ekonomicky splnit jedinou pamětí. Čím větší je kapacita paměti, tím větší je totiž i její cena. Podobně roste cena i se zkracováním vybavovací doby. Proto rychlé paměti (s krátkou vybavovací dobou) mají obvykle malou kapacitou, pomalé paměti (s delší vybavovací dobou) mají velkou kapacitu. Začlenění jednotlivých úrovní paměti do hierarchie je naznačeno na obrázku.



Hierarchie paměťového podsystému

Operační podsystém provádí všechny aritmetické a logické operace. Přitom spolupracuje s pamětí, z níž vybírá operandy a ukládá do ní výsledky. Operaci, která se má provádět, určuje řídící jednotka. Typické složení operační jednotky je na následujícím obrázku.



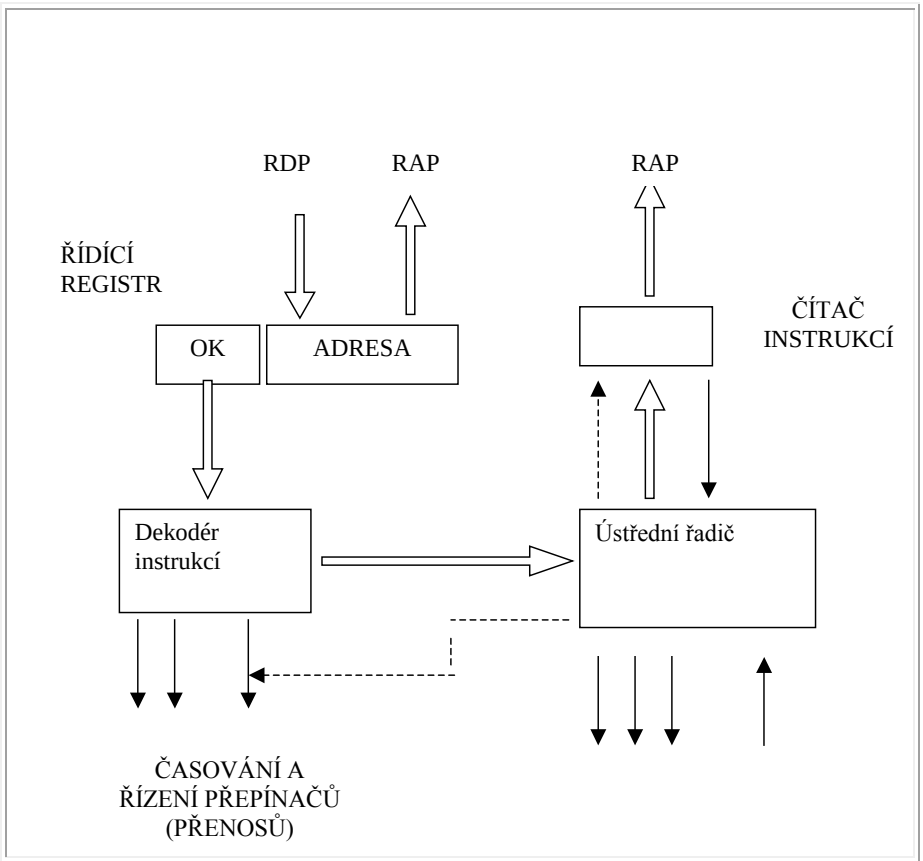
Operační podsystém (ALU) počítače

Tři registry, každý je na jedno slovo, tvoří zápisníkovou paměť. Registry uchovávají operandy, mezivýsledky i konečné výsledky. Označení registrů vychází z jejich funkce:

- RA - střádač (acumulator),
- RB - báze (base),
- RQ - podíl (quotient).

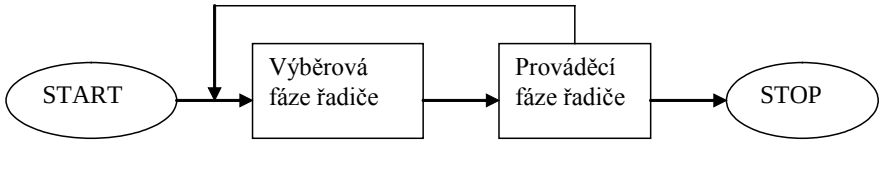
ALU obsahuje dále sčítačku, která má i prostředky pro vytváření doplňkových kódů, čímž umožňuje odčítání. Přepínač řídí tok informací a umožňuje přímý zápis informace z jednoho registru do druhého, průchod informace ze dvou registrů sčítačkou, přesun údaje z paměti do některého registru apod. Řadič ALU je autonomní. Dohlíží na činnost ALU. Jakmile se mu předá instrukce, která se má provést, řídí a časově sleduje provádění příslušného algoritmu. (Pozn. ALU současných procesorů obsahují podstatně větší počet registrů.)

Řídící podsystém (řadič) řídí, resp. dohlíží na veškerou činnost a spolupráci všech podsystémů počítače tak, aby celý systém počítače pracoval podle zadaného programu. Dostává instrukce programu, zpracovává je a transformuje je na posloupnost příkazů pro ostatní části počítače. Těmito příkazy přiděluje úkoly jiným podsystémům. Přitom je stále informován o okamžitém stavu všech podsystémů. Základní funkční jednotky řídícího podsystému jsou uvedeny na obrázku.



Řídící podsystém počítače

Řídící podsystém pracuje ve dvou fázích: výběrové a prováděcí.



Výběrová a prováděcí fáze řadiče

Výběrová fáze (fetch):
Obsah čítače instrukcí se přepíše do adresovaného registru paměti (RAP). V paměti se najde specifikovaná adresa, na níž je uložena instrukce. Instrukce se z tohoto místa přepíše do registru dat (RDP) a odtud se uloží do řídícího registru.

Prováděcí fáze (execute):
Vybraná instrukce, uložená v řídícím registru, se analyzuje, čímž se určí, co se má dělat. Pokud se při zpracování požaduje operand, přepíše se adresová část instrukce do adresového registru paměti. Vybraný údaj se pak odesílá registru dat paměti na místo určení, např. do operační jednotky. Jakmile se operand dostane na místo svého určení, řídící podsystém odešle příslušnému podsystému požadavek na zpracování úkolu a kontroluje jeho provádění. Nakonec připraví adresu další instrukce, např. zvětšením obsahu čítače instrukcí o jedničku a zahájí se nová výběrová fáze. V případě provádění

instrukce skoku se adresová část instrukce přepíše do čítače instrukcí a tím se určí nová adresa vybírané instrukce

Vstupní a výstupní podsystém počítače je spojovacím článkem mezi přídavnými zařízeními a ostatními podsystémy počítače a zahrnuje v sobě logiku řízení vstupů a výstupů. Požadavky na připojení vstupních a výstupních zařízení nejsou vždy známé nebo se nedají dostatečně přesně předvídat při návrhu počítače, a proto se základní vstupní a výstupní funkce z hlediska organizace a připojení přídavných zařízení zajišťují univerzálně.

1.3 Zobrazení informace v počítači

Informace, které číslicové počítače zpracovávají, musí být kódovány a s ohledem na technickou realizaci je nutné k jejich vyjádření nalézt levné a spolehlivé prvky. Z tohoto hlediska jsou zatím stále nejvhodnější prvky se dvěma jednoznačně rozlišitelnými stavy. Tak se dostáváme k dvojkové číselné soustavě, která je dominující soustavou strojového zpracování informací. Dvojková číslice (0 nebo 1) je bit (elementární informace). Znaky (alfabetické a numerické) se vyjadřují většinou osmibitovou skupinou. Pro tyto několikabitové skupiny se používá termín slabika nebo byte. Nejbližší větší skupina s pevně stanoveným počtem bitů je slovo. Délka slov bývá 4 slabiky, tj. 32 bitů nebo 8 slabik, tj. 64 bitů. Je-li v počítači pro všechny operace slovo uložené vždy v jedné buňce paměti, pak jde o počítač slovně orientovaný.

1.3.1 Číselné soustavy a kódy

Zobrazení čísla v poziční číselné soustavě o základu **b** je dána pravidlem

$$(...a_3a_2a_1a_0a_{-1}a_{-2}...)_b = ... a_3b^3 + a_2b^2 + a_1b^1 + a_0b^0 + a_{-1}b^{-1} + a_{-2}b^{-2} ...$$

Pro **b**=10 dostáváme tradiční desítkovou (dekadickou) soustavu. Méně obvyklé jsou soustavy se základem **b**>1, **b** je přirozené a různé od 10.
Ve výše uvedeném zápisu znamená každé a_k "číslici", představující celé číslo

$$0 \leq a_k < \mathbf{b} .$$

Čárka mezi a_0 a a_{-1} se nazývá „řádová poziční čárka“ (v angličtině se užívá tečky).

Dále se budeme zabývat zejména soustavami:
- dvojkovou (binární) kde **b**=2,
- osmičkovou (oktalovou) kde **b**=8,
- šestnáctkovou (hexadecimální) kde **b**=16.

Ve dvojkové soustavě vystačíme se dvěma číslicemi: 0,1.
V osmičkové soustavě potřebujeme 8 číslic: 0,1,2,3,4,5,6,7.

V šestnáctkové soustavě potřebujeme 16 číslic:
0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

Z číselných soustav je nejběžnější dvojková, která jak z hlediska technického, tak i programového vybavení stroje má značné výhody. Z hlediska technického jsou to

např. jednoduché algoritmy aritmetických a logických operací a jejich realizace, ve srovnání s jinými soustavami se jedná o menší požadovaný rozsah paměti apod. Z programového hlediska je třeba ocenit možnost dělení a násobení mocninami dvou prostým bitovým posunem čísla. Hlavní nevýhodou dvojkové soustavy je nutnost převodu při vstupu a výstupu z desítkové soustavy do dvojkové a naopak. Desítková soustava má přednost právě v tom, že zmíněné převody odpadají. Proto se u řady počítačů jednotlivé desítkové číslice kódují do dvojkové soustavy, tzv. dvojkovědesítkovým kódem (BCD).

1.3.1.1 Převody mezi číselnými soustavami

Pracujeme-li s různými číselnými soustavami (o různých základech), musíme vědět, jak se číslo, vyjádřené v jedné soustavě, např. se základem **k** převede do soustavy se základem **m**. Jde-li o celé číslo >1, potom číslo v soustavě o základu k dělíme základem **m**, vyjádřeným v soustavě k, zbytek **h** <0, **m**-1> je nejnižší řád daného čísla v soustavě **m**, vyjádřený ovšem opět v soustavě o základu **k**. Získaný podíl se dělí znovu základem **m**. Tak dostaneme další řád čísla v soustavě **m** atd. Poslední podíl, který je **h** <0, **m**-1>, je nejvyšší řád převáděného čísla v soustavě o základu **m**.

Příklad: Převeďme desítkové číslo 287 do soustavy dvojkové, osmičkové a šestnáctkové.
Ve dvojkové soustavě dostáváme:

287 : 2 = 143	143 : 2 = 71	71 : 2 = 35	35 : 2 = 17
zbytek 1	zbytek 1	zbytek 1	zbytek 1
17 : 2 = 8	8 : 2 = 4	4 : 2 = 2	2 : 2 = 1
zbytek 1	zbytek 0	zbytek 0	zbytek 0 zbytek 1

Zbytky sepíšeme od konce. Číslo 287_{10} je tedy vyjádřeno ve dvojkové soustavě takto:

$(100011111)_2 = 1\cdot2^8+0\cdot2^7+0\cdot2^6+0\cdot2^5+1\cdot2^4+1\cdot2^3+1\cdot2^2+1\cdot2^1+1\cdot2^0 = 256+16+8+4+2+1 = (287)_{10}$

V soustavě osmičkové dostaneme:

287 : 8 = 35	35 : 8 = 4	
7	3	4

Tedy: $(437)_8 = 4\cdot8^2 + 3\cdot8^1 + 7\cdot8^0 = 256 + 16 + 15 = (287)_{10}$

Při převodu do šestnáctkové soustavy obdržíme:

287 : 16 = 17	17 : 16 = 1	
zbytek 15 (F)	zbytek 1	zbytek 1

Obdobně zbytky sepíšeme od konce a obdržíme:

$(11F)_{16} = 1\cdot16^2 + 1\cdot16^1 + 15\cdot16^0 = 256 + 16 + 15 = (287)_{10}$

Máme-li převést číslo <1 ze soustavy o základu **k** do soustavy o základu **m** postupujeme takto: číslo v soustavě **k** se násobí základem **m**, vyjádřeným v soustavě o základu **k**. Celočíselná část výsledku (před řádovou čárkou) je prvá číslice za řadovou

čárkou čísla, převedeného do soustavy o základu **m** vyjádřená ovšem ještě v soustavě o základě **k**. Zbývající část výsledku za řádovou čárkou násobíme znovu základem **m** atd.

Příklad: Převeďme číslo $(0,960375)_{10}$ ze soustavy desítkové do soustavy dvojkové, osmičkové a šestnáctkové.

Ve dvojkové soustavě dostáváme:

0,960375 · 2
1,92075 celá část je 1

0,92075 · 2
1,8415 celá část je 1

0,8415 · 2
1,683 celá část je 1

0,683 · 2
1,366 celá část je 1

0,366 · 2
0,732 celá část je 0

0,732 · 2
1,464 celá část je 1

Celé části (přetečení přes řádovou čárku) sepíšeme a dostáváme:

$(0,960375)_{10} = (0,111101)_2 = 1\cdot2^{-1}+1\cdot2^{-2}+1\cdot2^{-3}+1\cdot2^{-4}+0\cdot2^{-5}+1\cdot2^{-6}... = 0,5+0,25+0,125 +0,0625+0,03125 + …..$

Při převodu do osmičkové soustavy dostáváme:

0.960375 · 8	0.683 · 8	0,464 · 8
7,683	5,464	3,712
celá část je 7	celá část je 5	celá část je 3

Celé části opět sepíšeme a dostáváme:

$(0,960375)_{10} = (0,753...)_8 = 7\cdot8^{-1} + 5\cdot8^{-2} + 3\cdot8^{-3} + ... = 0.875 + 0,078125 + 0,005859375 + ..,$

V šestnáctkové soustavě obdržíme:

0,980375 · 16	0,366 · 16	0,856 · 16
15,366	5,856	13,895
celá část je 15 (F)	celá část je 5	celá část je 13 (D)

Tedy:

$(0,960375)_{10} = (0,F5D...)_{16} = 15 \cdot 16^{-1} + 5 \cdot 16^{-2} + 13 \cdot 16^{-3} ... = 0,9375 + 0,01953125 + 0,003173828 + ...$

Všimněte si, že existuje prostý vztah mezi zápisem čísel v soustavách o základu **b** a $\mathbf{b^k}$:

$(...a_3a_2a_1a_0.a_{-1}a_{-2}...)_b = (...A_3A_2A_1A_0.A_{-1}\ A_{-2}....)b^k$

kde $A_j=(a_{kj+k-1} .....a_{kj+1}\ a_{kj})_b$
Slovy lze tento vztah zhruba vyjádřit tak, že v zápise čísla v soustavě o základu **b** vytvoříme **k**-tice číslic nalevo i napravo od řádové tečky, a každou tuto **k**-tici číslic vyjádříme jednou číslicí v soustavě o základu $\mathbf{b^k}$.

Příklad: Převeďte dvojkové číslo 10100110011001,10011101 do osmičkové a šestnáctkové soustavy.

Při převodu do osmičkové soustavy platí b=2, k=3, $2^k=2^3=8$

Tedy vytvoříme trojice dvojkových číslic, které nahradíme osmičkovými číslicemi dle následující tabulky.

osmičkové	dvojkové	šestnáctkové	dvojkové	šestnáctkové	dvojkové
0	000	0	0000	8	1000
1	001	1	0001	9	1001
2	0l0	2	0010	A	1010
3	011	3	0011	B	1011
4	100	4	0100	C	1100
5	101	5	0101	D	1101
6	110	6	0110	E	1110
7	111	7	0111	F	1111

Převodní tabulka mezi osmičkovou a šestnáctkovou do dvojkové soustavy

Vytvořené trojice dvojkových čísel jsou:

010	100	110	011	001 ,	100	111	010
2	4	6	3	1	4	7	2

Osmičkový zápis výše uvedeného čísla je $(24631,472)_8$.

Převádíme-li číslo do šestnáctkové soustavy, platí: b=2, k=4, $2^k=2^4=16$
Zde vytváříme čtveřice dvojkových číslic, které nahradíme šestnáctkovými číslicemi dle výše uvedené tabulky.

0010	1001	1001	1001 ,	1001	1101
2	9	9	9	9	D

Šestnáctkový zápis daného čísla je $(2999,9D)_{16}$.

Pozn.: při obráceném převodu z osmičkové (resp. šestnáctkové) soustavy do dvojkové, každou osmičkovou (resp. šestnáctkovou) číslici nahradíme trojicí (resp. čtveřicí) dvojkových číslic.

1.3.1.2 Kódy

V řadě počítačů se používá dvojkově kódovaná desítková soustava. Každá desítková číslice se přitom musí vyjádřit dvojkově. Je zřejmé, že musíme použít pro každou číslici v desítkové soustavě nejméně 4 číslice dvojkové soustavy.

Volba konkrétního kódu závisí na jeho použití. Jiné požadavky se kladou na kódy, s nimiž se mají provádět aritmetické operace a jiné požadavky na kódy, používané při přenosu informaci apod.

Pro aritmetické operace jsou vhodné kódy, které vyhovují tzv. **Aikenovým podmínkám**:

1. Každé místo v kódu má mít určitou váhu.
2. Součet vah míst, v nichž je dvojková číslice kódu rovna 1, má dát hodnotu přiřazené desítkové číslice nebo alespoň, větší desítkové číslici má odpovídat větší dvojkové číslo příslušného váhového kódu.
3. Vztah mezi lichými a sudými kódy a přiřazenými desítkovými číslicemi může být sice libovolný, ale u zvoleného kódu má být neměnný.
4. Desítkovým doplňkům desítkových číslic mají odpovídat doplňkové kódy, vzniklé inverzí jednotlivých bitů původních kódů.

U kódů určených pro přenos nás zajímá především odolnost kódů vůči poruchám, možnost zajištění vzniklých chyb a příp. jejich odstranění. V následující tabulce jsou uvedeny některé známé kódy, z nichž však všechny nesplňují výše uvedené podmínky.

	8421	2421	(8421)+3	5421	3n+2	5043210
				Bikvinární		
0	0000	0000	0011	0000	00010	0100001
1	0001	0001	0100	0001	00101	0100010
2	0010	0010	0101	0010	01000	0100100
3	0011	0011	0110	0011	01011	0101000
4	0100	0100	0111	0100	01110	0110000
5	0101	1011	1000	1000	10001	1000001
6	0110	1100	1001	1001	10100	1000010
7	0111	1101	1010	1010	10111	1000100
8	1000	1110	1011	1011	11010	1001000
9	1001	1111	1100	1100	11101	1010000

Tabulka dvojkových kódů pro desítkové číslice

Z více možných způsobů zabezpečení informace (kódů) se stručně zmíníme jen o jedné, tzv. kontrole paritou. Podstata zabezpečení spočívá v tom, že v nejjednodušším případě se k informaci, která je v **n**-řádech, přidružuje informace **n(n+1)**-tém řádu, kterou se doplňuje počet jednotkových bitů na sudý nebo lichý počet (sudá nebo lichá parita). V následující tabulce jsou naznačeny **n**=5 bitové informace se sudou paritou v 6. řádu.

n	n+1
10110	1
01001	0
01111	0
01011	1
01001	0

Příklad zabezpečení informace sudou paritou

Při přenosu informace se po každém přepisu kontroluje, zda počet jedniček v informaci je sudý. V případě lichého počtu se signalizuje chyba. Je třeba připomenout, že paritní bit se k informaci přidává při tvorbě informace, např. na paměťové médium.

1.3.2 Kódování ve dvojkové soustavě

Způsob zobrazeni čísel v počítači nazýváme číselný kód. Ukážeme si tři způsoby kódování čísel ve dvojkové soustavě. Je zřejmé, že kladná a záporná čísla jsou zobrazena v počítači stejně, až na znaménko. Aby počítač pracující ve dvojkové soustavě poznal znaménko čísla, musíme tato znaménka vyjádřit znakem, který počítač dovede rozlišit, tj. v našem případě nulou nebo jedničkou (pozn., pracujeme-li s jinou číselnou soustavou, kóduje se znaménko zvláštním znakem). Kvůli zjednodušení budeme uvažovat o zobrazení čísla s pevnou řádovou čárkou (s čárkou před nejvyšším řádem, což značí $|x| < 1$).

1.3.2.1 Přímý kód

Kladné číslo bude mít v přímém kódu před pevnou řádovou čárkou, tj. ve znaménkovém řádu, nulu, číslo záporné jedničku. Čísla kladná jsou tedy vyjádřena v rozmezí od nuly do jedné, čísla záporná od jedné do dvou, což lze psát:
$(x)_{\text{přímý kód}} = x$ pro $x \geq 0$
$(x)_{\text{přímý kód}} = 1\text{-}x$ pro $x < 0$.

Zapíšeme tedy číslo + 0,21875 jako 0,00111,
ale číslo -0,21875 jako 1,00111 = (1-(-0.00111)).

1.3.2.2 Doplňkový kód

Doplňkový kód nezáporného čísla (kladné a nula) **x** je stejně jako u přímého kódu roven číslu **x**. Doplňkový kód záporného čísla **x** dostaneme tak, ze před pevnou řádovou čárku napíšeme znaménkový znak 1, na všech ostatních místech nahradíme nuly jedničkami.

Obecně můžeme psát:
$(x)_{\text{doplňkový kód}} = x$ pro $x \geq 0$
$(x)_{\text{doplňkový kód}} = 2 + x$ pro $x < 0$.

Zapíšeme tedy + 0,75 jako 0,110
ale číslo - 0,75 jako 1,010
protože 2_{10} v binární soustavě je 10_2

pak 10,000 nebo 1,001
- 0,110 + 1
1,010 1,010

Nula má jediné vyjádření 0,000 ... 0.
Maximální záporné číslo 1,000 ... 0 nemá tedy kladný ekvivalent (jeho dvojkový doplněk je totiž zase 1,000 ... 0).

1.3.2.3 Inverzní kód

Inverzní kód kladného čísla x je roven číslu x. Inverze záporného čísla vznikne tak, že se do znaku napíše jednička a v číslicových řádech se zamění jedničky za nuly a naopak. Je tedy záporné číslo v inverzním kódu v posledním číslicovém řádu o jedničku menší než číslo v doplňkovém kódu.
Tedy:
$(x)_{\text{inverzní kód}} = x$ pro $x >= 0$
$(x)_{\text{inverzní kód}} = 10 + x - 10^{-n}$ pro $x < 0$.

Př. + 0,8128 je ve dvojkové soustavě 0,1101 a - 0,8125 je v inverzním kódu 1,0010

1.3.2.4 Modifikovaný doplňkový kód

Modifikovaný doplňkový kód je obecnější verzí doplňkového kódu.
Je definován

$(x)_{\text{modifikovaný doplňkový kód}} = x$ pro $x >= 0$
$(x)_{\text{modifikovaný doplňkový kód}} = z^2 + x$ pro $x < 0$.

Ve dvojkové soustavě je $z^2 = (100)_2$

Př.
$(+ 11/16)_{10} = (00,1011)_2$

$(- 11/16)_{10} = (11,0101)_2$ protože 100,0000 nebo 11,0100
- 00,1011 + 1
11,0101 11,0101

Modifikovaný doplňkový kód nezáporného čísla **x** je roven tomuto číslu **x**, přičemž ve znaménkových řádech jsou dvě nuly; u záporného čísla se před pevnou řádovou čárku napíší dvě jedničky, na všech ostatních místech nahradíme nuly jedničkami a jedničky nulami a pak k nejnižšímu řádu přičteme jedničku.

1.3.2.5 Modifikovaný inverzní kód

Je definován
$(x)_{\text{modifikovaný inverzní kód}} = x$ pro $x >= 0$
$(x)_{\text{modifikovaný inverzní kód}} = 100 + x - 10^{-n}$ pro $x < 0$.

Modifikovaný inverzní kód nezáporného čísla **x** je roven číslu **x**; u záporného čísla se před pevnou řádovou čárku napíší dvě jedničky a v číslicových řádech se zamění jedničky za nuly a naopak.

Např.
$(+ 11/16)_{10} = (00,1011)_2$

$(- 11/16)_{10} = (11,0100)_2$

1.3.3 Zobrazení instrukcí

Požadavek na provedení operace operační jednotkou, případně jinými podsystémy počítače je zakódován do instrukce. Instrukce určuje, o kterou z operací jde (tzn. operační kód) a kde jsou uloženy operandy v paměti, registrech apod. (tzn. adresy) .

Obecná struktura instrukce je na následujícím obrázku.

Operační kód	Adresová část

Obecná struktura instrukce

Adresová část obsahuje tolik samostatných částí, kolik má instrukce adres (většinou 1,2 nebo 3) - hovoříme potom o jednoadresní, dvouadresní nebo tříadresní instrukci.

Instrukce s **k** = 1,2 ..... adresami mají formát dle následujícího obrázku.

Operační kód	A1	A2		A**k**

Obecná struktura instrukce s k adresami

Porovnání efektivnosti instrukcí s různým počtem adres budeme ilustrovat na příkladu výpočtu výrazu:

Y = A * B + (C – D) * E / F,

kde A,B,C.D,E,F,Y - proměnné, uložené v buňkách s adresami a,b,c,d,e,f,y.

Budeme předpokládat, že výsledek libovolné operace se ukládá v registru R operační jednotky, a může být použit v roli operandu v následující operaci. Pro uložení mezivýsledků P1, P2, ... máme k dispozici pracovní paměťové buňky p1, p2, ...

Nejpřirozenější pro výpočet aritmetických a logických výrazů je tříadresní systém instrukcí.

OK	A1	A2	A3

Struktura tříadresní instrukce

Zde OK = operační kód,
A1, A2 = adresa dvou operandů,
A3 = adresa výsledku.
Program výpočtu výrazu bude:

	Instrukce			**Význam**
Násob	a	b	p1	A * B → P1
Odečti	c	d	-	C - D → R
Násob	-	e	-	R * E → R
Děl	-	f	-	R / F → R
Sečti	-	p1	y	R + P1 → Y

Pomlčka v poli adresy označuje, že tato adresa se nepoužije, i když v poli adres v instrukci se nachází. Protože každá instrukce potřebuje tři m-bitová pole adres, tak pro adresaci informace je potřebné K3 = 3 x 5 = 15 adres, i když pouze 9 je jich využito efektivně. Při vykonání programu se T3 = 14 krát obracíme k paměti:

- 5x výběr instrukce,
- 9x čtení / zápis operandů.

Při použití dvouadresních instrukcí:

OK	A1	A2

Struktura dvouadresní instrukce

Proces výpočtu výrazu je popsán následujícím programem:

Instrukce			**Význam**
Násob	a	b	A * B → R
Zapiš	p1	-	R → P1
Odečti	c	d	R - D → R
Násob	-	**e**	R * E → R
Děl	-	f	R / F → R
Sečti	P1	y	R + P1 → Y

Zde **Zapiš** je operace zápisu výsledku R do paměti. Program je sestaven ze šesti instrukcí, zabírajících K2 = 2x6 = 12 adres a 9 z nich je využito efektivně. Při provádění programu se T2 = 15x obracíme k paměti:

- 6x výběr instrukce,
- 9x čtení / zápis operandů.

Při práci s jednoadresní instrukcí:

OK	A1

Struktura jednoadresní instrukce

Program výpočtu výrazu bude:

	Instrukce	**Význam**
Ulož	a	A → R
Násob	b	R * B → R
Zapiš	p1	R → P1
Ulož	c	C → R
Odečti	d	R - D → R
Násob	e	R * E → R
Děl	f	R / F → R
Sečti	p1	R + P1 → R
Zapiš	y	R → Y

Program je sestaven z 9 instrukcí zabírajících 9 adres. Při provádění programu T1 = 18x obrací k paměti:

- 9x výběr instrukce,
- 9x čtení / zápis operandů.

Počet adres instrukcí k	1	2	3
Počet adres v programu K	9	12	15
Počet přístupů k paměti T	18	15	14

Z tabulky je vidět, že s větším počtem adres v instrukci se zvětšují požadavky na velikost paměti pro adresu, ale snižuje se počet přístupů k paměti, tj. snižuje se celková doba výpočtů. Se zvyšováním počtu operandů ve výrazu odpadá nutnost každou operaci provádět třemi adresami; výsledek operace může být uložen v procesoru a může být použit jako operand v následující operaci. Ve vědecko-technických úlohách, na jednu operaci, přísluší v průměru 1,2 až 1,4 operandů, uložených v operační paměti a v úlohách zpracování hromadných dat 1,6 až 1,8 operandů. Proto pro vědecko-technické úlohy jsou výhodné 1-adresní instrukce a pro zpracováni dat 2-adresní instrukce.

1.3.4 Zobrazení abecedně-číslicových znaků

Současné počítače (až na výjimky) mohou pracovat s úplnou abecedou písmen, číslic a různých symbolů, (tzn. 26 písmen mezinárodní abecedy, 10 číslic a interpunkční znaménka, panelové znaky pro některé vstupní a výstupní zařízení). Základní abeceda zejména ve slovanských zemích bývá doplněna písmeny národní abecedy. V paměťových médiích samočinného počítače se zpravidla uchovává až dvakrát tolik číselné informace jako alfabetické informace. Proto je žádoucí, aby vybraný kód byl efektivnější pro vyjadřování číslic. Tuto podmínku splňuje např. osmibitový kód EBCDIC (Extended Binary Coded Decimal Interctange Code), jenž umožňuje vyjádřit

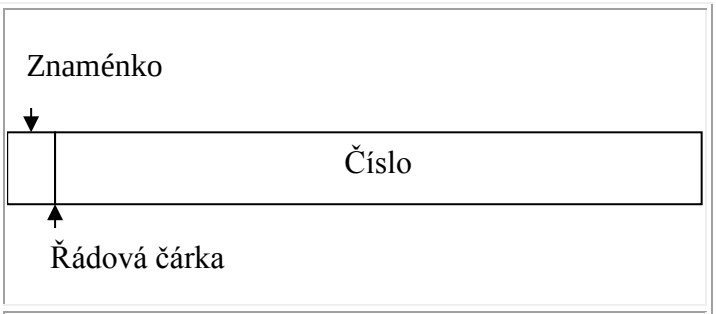
206 různých znaků. Každý znak tedy představuje skupina osmi bitů. Výhoda osmibitového kódu spočívá v tom, že umožňuje v jedné osmibitové skupině uchovávat kódy dvou desítkových číslic. Tyto číslice vstupují do stroje zakódovány osmi bity; čtyři bity levé části tvoří tzv. zónu, která svou hodnotou (1111) určuje, že znak je číslice. Znaky se dají zřetězit a v poslední osmibitové kombinaci se místo zóny udává znaménko čísla. Desítkové číslice 0, 1, 2, ... 9 jsou zakódovány v přímém dvojkovém kódu 8421. Uvedený tvar je "rozvinutý" a používá se pro styk s periferními zařízeními. Číselná informace se před uložením do paměti přeskupí do zhuštěného tvaru. Zhuštěný tvar se uplatňuje při vnitřních operacích s desítkovými čísly.

	0-	1-	2-	3-	4-	5-	6-	7-
-0	NUL	DLE	SP	0	@	P	`	p
-1	SOH	DC1	!	1	A	Q	a	q
-2	STX	DC2	"	2	B	R	b	r
-3	ETX	DC3	#	3	C	S	c	s
-4	EOT	DC4	$	4	D	T	d	t
-5	ENQ	NAK	%	5	E	U	e	u
-6	ACK	SYN	&	6	F	V	f	v
-7	BEL	ETB	'	7	G	W	g	w
-8	BS	CAN	(	8	H	X	h	x
-9	HT	EM	)	9	I	Y	I	y
-A	LF	SUB	*	:	J	Z	j	z
-B	VT	ESC	+	;	K	[	k	{
-C	FF	FS	,	<	L	\	l	|
-D	CR	GS	-	=	M	]	m	}
-E	SO	RS	.	>	N	^	n	~
-F	SI	US	/	?	O	_	o	DEL

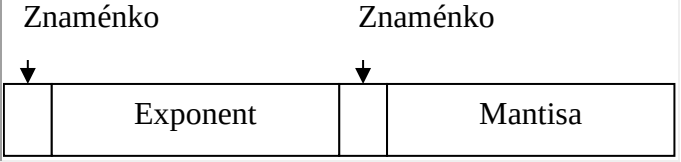
Znaková sada ASCII

1.3.5 Pevná a pohyblivá řádová čárka

Většina moderních počítačů umožňuje zpracovávat čísla ve dvojím tvaru, a to v pevné a pohyblivé řádové čárce, viz následující obrázek.



Obecné uspořádání čísla v pevné řádové čárce



Obecné uspořádání čísla v pohyblivé řádové čárce

Pro zobrazení čísel v pevné řádové čárce je charakteristické pevné umístění řádové čárky na určité místo. Zpravidla se umísťuje před nejvyšším řádovým místem. Můžeme tedy zobrazit čísla v intervalu <0,1>.

Zápis čísla v pohyblivé řádové čárce má dvě části: mantisu a exponent.
Obecný zápis je $x = m.\ z^e$
kde

x - zobrazované číslo,
m – mantisa,
e – exponent,
z - základ použité číselné soustavy.

Lze tak zobrazit čísla v rozsahu $<z^{-(emax+m)}; z^{emax}>$
kde emax je maximální hodnota exponentu, která je dána počtem řádových míst exponentu v daném zobrazení, m je počet řadových míst mantisy. Rozsah zobrazení informace v pevné řádové čárce úzce souvisí s délkou slova počítače.

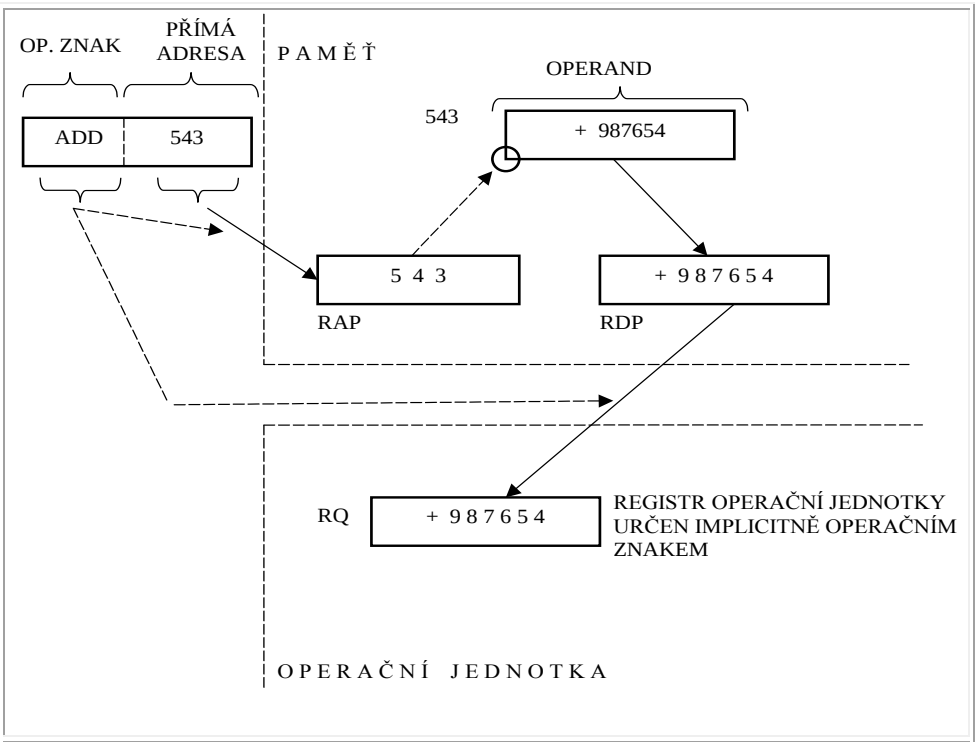
1.4 Metody určování operandů

V číslicových počítačích se setkáváme s různými typy určování operandů. Jsou to:

- Přímé adresování.
- Přímý operand.
- Nepřímé adresování.
- Implicitní adresování.
- Implicitní operand.
- Modifikované adresování.
- Relativní adresování.
- Asociativní adresování.

1.4.1 Přímé adresování

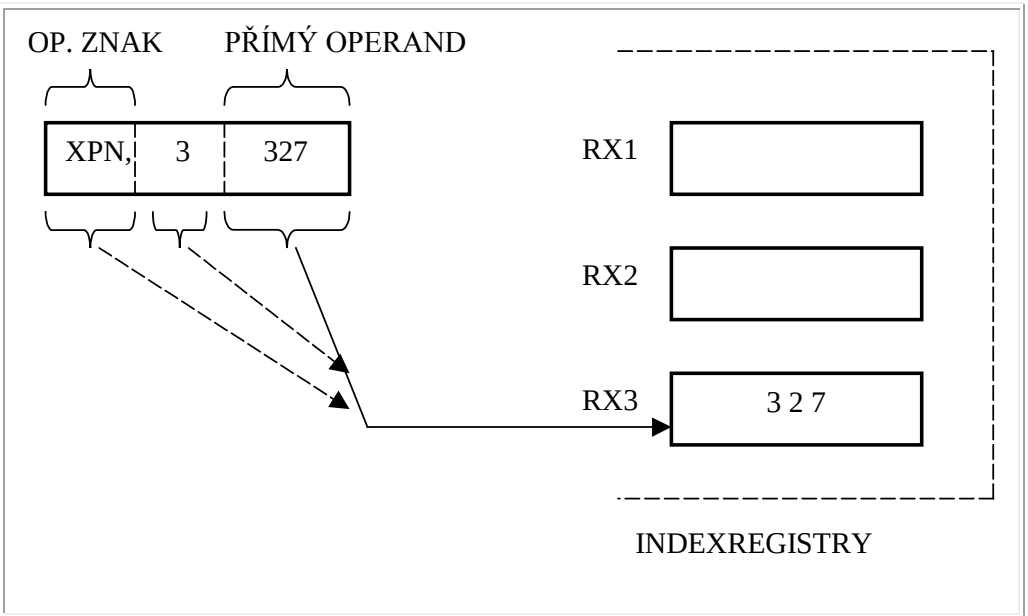
Přímé adresování je základní způsob adresování, kdy se pracuje s operandem na adrese uvedené přímo v instrukci.



Přímé adresování

1.4.2 Přímý operand

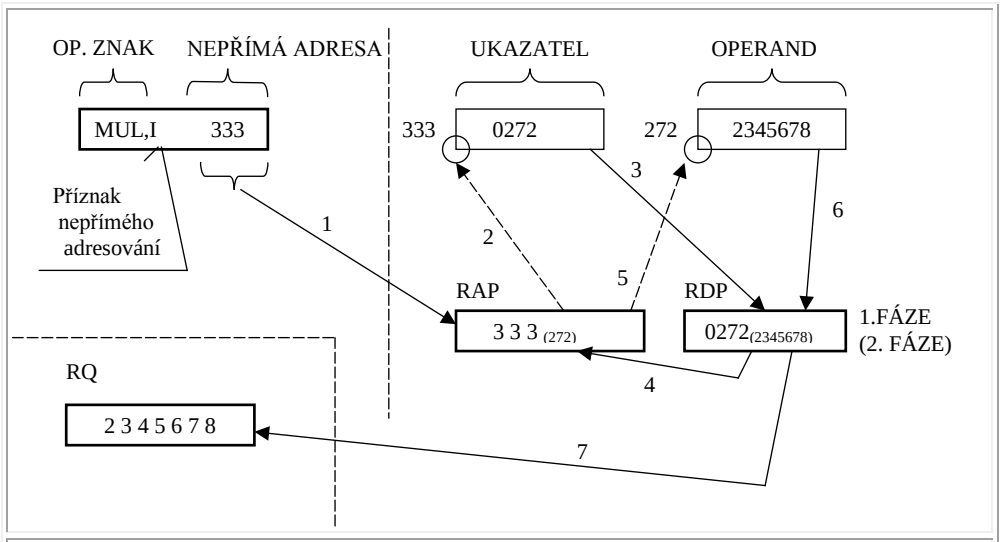
V tomto případě je (obvykle na místě adresy) v instrukci zapsán vlastní operand. Rozlišení, zda jde o adresu nebo operand, musí být dáno operačním kódem. Provedením instrukce na následujícím obrázku se přesune přímý operand 327 do indexregistru číslo 3.



Přímý operand

1.4.3 Nepřímé adresování

Adresová část instrukce udává adresu, na níž je zaznamenána adresa vlastního operandu. Ve srovnání s přímým adresováním to vyžaduje přídavný cyklus paměti, což je nevýhodné. Uplatní se jen tehdy, jestliže řada instrukcí odkazuje na týž operand, jehož umístění v paměti se však během zpracování programu mění.



Nepřímé adresování

1.4.4 Implicitní adresování

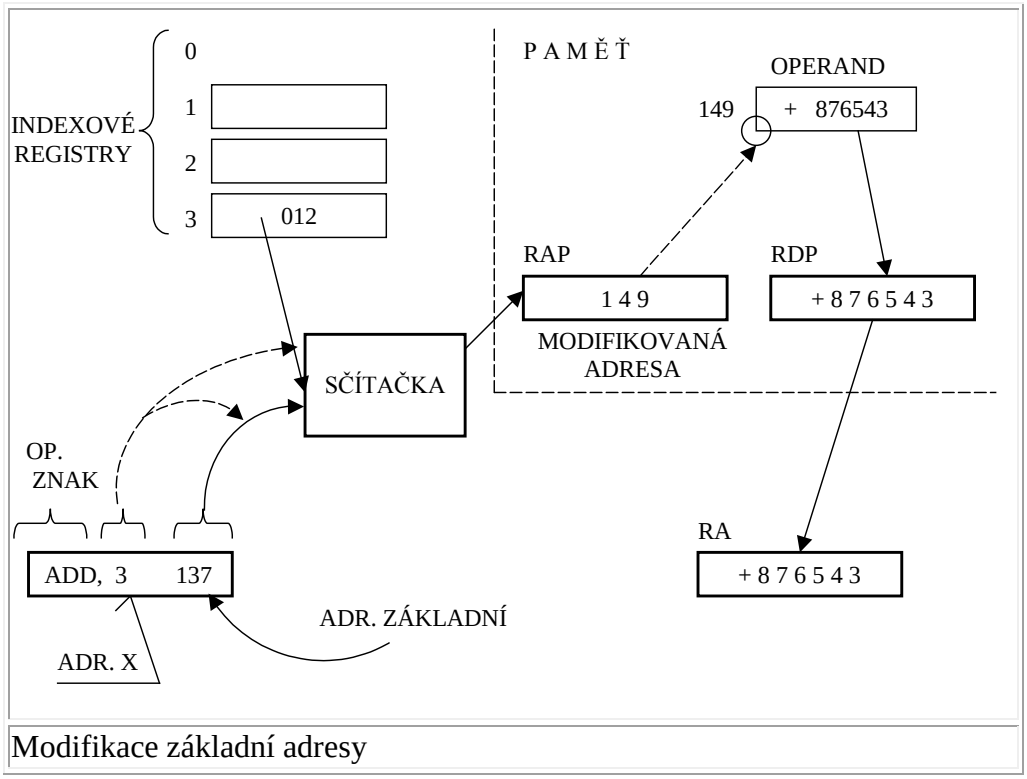
S implicitním adresováním se setkáváme u všech jednoadresových počítačů, které vyžadují, aby pro všechny operace pracující se dvěma operandy bylo umístění jednoho z nich dáno vlastní operací. Místo pro výsledek je ve většině případů určeno také implicitně. Příkladem může být instrukce přenosu mezi dvěma registry, kdy adresy obou registrů jsou implicitně určeny operačním kódem.

1.4.5 Implicitní operand

Jde o případ, kdy operand je určen operačním kódem, např. kód operace „posuň řádovou čárku o jeden bit“ v sobě zahrnuje operaci násobení dvěma (u dvojkového počítače s pevnou řádovou čárkou).

1.4.6 Modifikované adresování

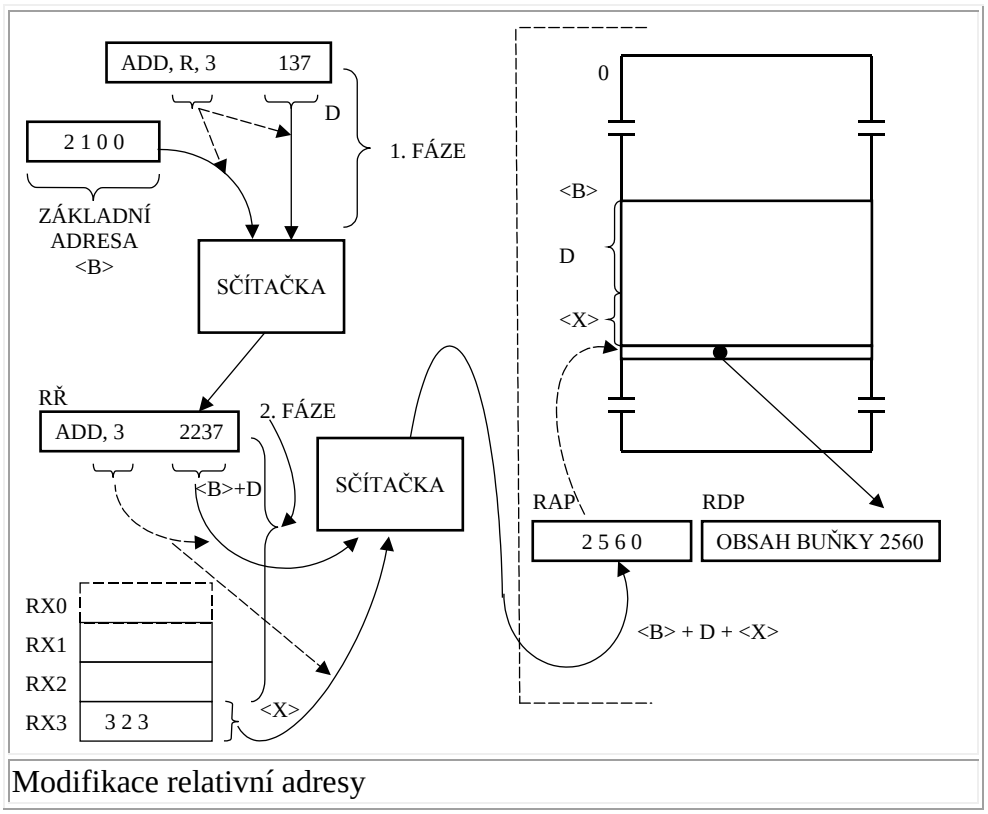
Adresová část instrukce se před vložením do adresového registru paměti RAP modifikuje konstantou uloženou v indexregistru, jehož číslo je dáno příznakem instrukce.



Modifikace základní adresy

1.4.7 Relativní adresování

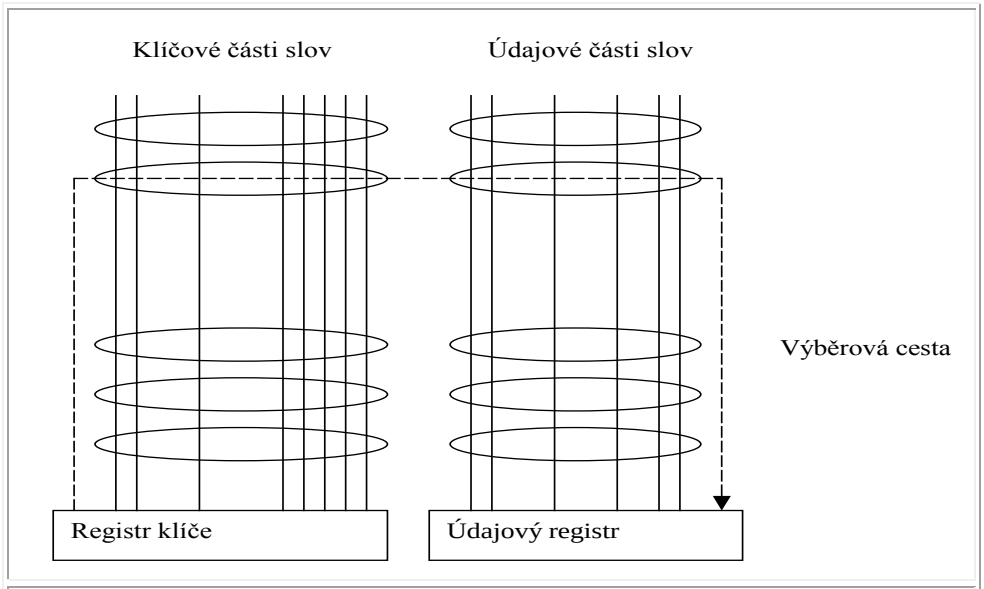
Adresová část instrukce neurčuje při relativním adresování přímo místo v paměti, ale adresu uloženou k nějaké jiné adrese, označené jako uložená adresa (bázová adresa).



Modifikace relativní adresy

1.4.8 Výběr podle obsahu (asociativní výběr)

Metody adresování, které jsme probrali, se týkají paměti, které jsou adresovány pozičně, tj. každé paměťové místo má svou adresu, což odpovídá Von Neumannově modelu počítače. U asociativních pamětí se vyhledává informace s určitou vlastností. Tato vlastnost bývá určena tzv. klíčem, což je určitá podmnožina bitů slova obsahující uvedený vzorek jedniček a nul. Zbylé bity jsou lhostejné a jsou při hledání maskovány. Každá buňka paměti má údajovou část (uložená informace) a klíčovou část. Obsah registru klíče se najednou srovná se všemi uloženými klíči. V té buňce, kde nastane shoda klíčů, se vyšle obsah údajové části buňky do údajového registru asociativní paměti.



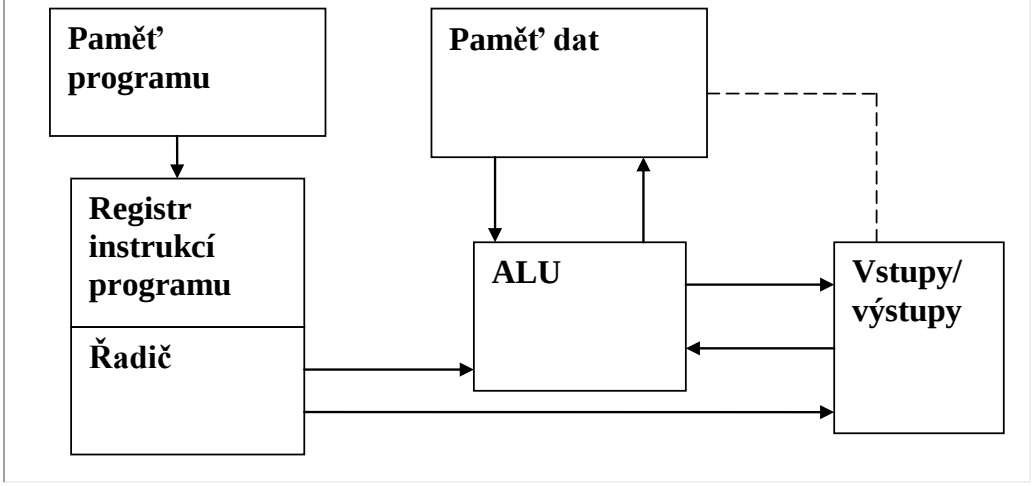
Asociativní adresování

1.5 Odlišná schémata počítačů

Základní odlišnosti dnešních počítačů od von Neumannova schématu:

- Podle von Neumannova schématu počítač pracuje vždy nad jedním programem. Toto vede k velmi špatnému využití strojového času. Je tedy obvyklé, že počítač zpracovává paralelně více programů zároveň - tzv. multitasking.
- Počítač může disponovat i více než jedním procesorem.
- Počítač podle von Neumannova schématu pracuje pouze v tzv. diskrétním režimu.
- Existují vstupní / výstupní zařízení (I/O devices), která umožňují jak vstup, tak výstup dat (programu).
- Program se do paměti nemusí zavést celý, ale je možné zavést pouze jeho část a ostatní části zavádět až v případě potřeby.

Jednou z nejpodobnějších architektur k von Neumanově je hardwardské schéma počítače, lišící se oddělenou pamětí pro program a pro data (používají některé jednočipové mikropočítače).



Hardwardské schéma počítače

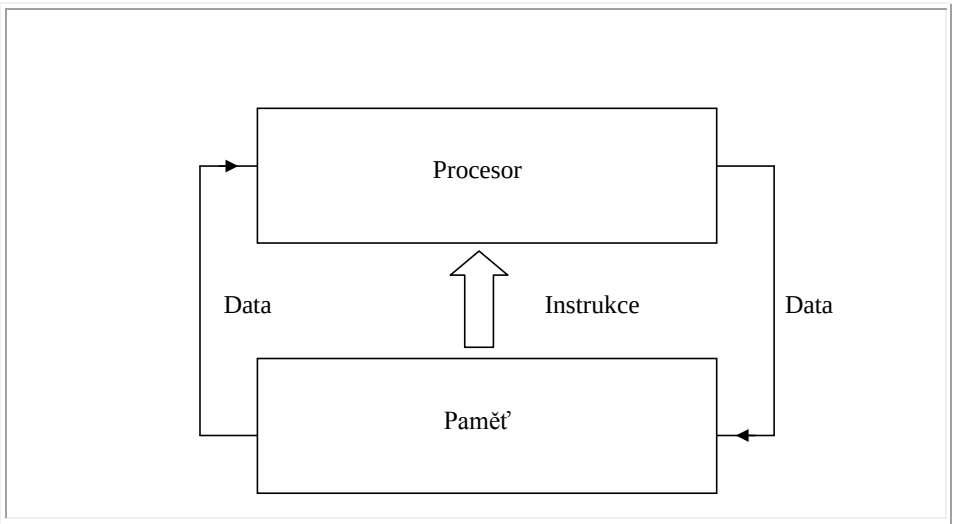
V poslední době se uplatňují víceprocesorové počítače, tj. počítače s několika CPU. Dělí se podle toho, zda mají sdílenou paměť na :

- multiprocesory (multiprocessors), které mají sdílenou paměť,
- multipočítače (multicomputers), které nemají sdílenou paměť, procesory komunikují například pomocí mechanismu zasílání zpráv.

Podle počtu instrukčních a datových proudů se počítače dělí na:

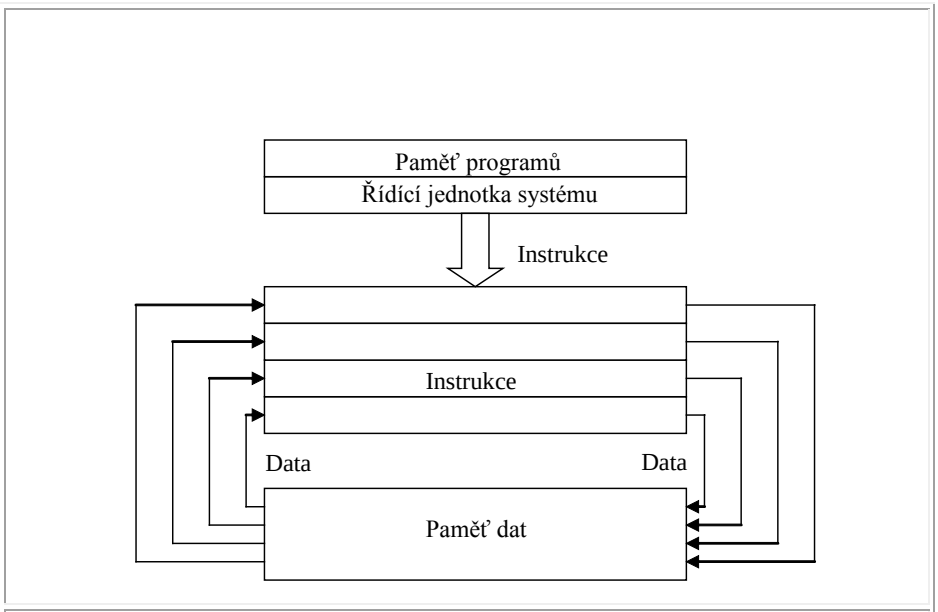
- SISD (single instruction, single data) - běžné jednoprocesorové počítače,
- SIMD (single instruction, multiple data) - jedna instrukce se provádí na větší množství dat (například sčítání dvou vektorů) - tzv. vektorové počítače, některé superpočítače jsou SIMD,
- MISD (multiple instruction, single data) - někdy označované jako řetězené (pipeline) procesory,
- MIMD (multiple instruction, multiple data) - víceprocesorové systémy; podle toho zda mají nebo nemají sdílenou paměť, se rozdělují na multiprocesory (se sdílenou pamětí) a multipočítače (multicomputers - spolupracující počítače propojené sítí).

Na následujících obrázcích jsou naznačeny principiální vazby procesoru event. procesorů s operační pamětí.



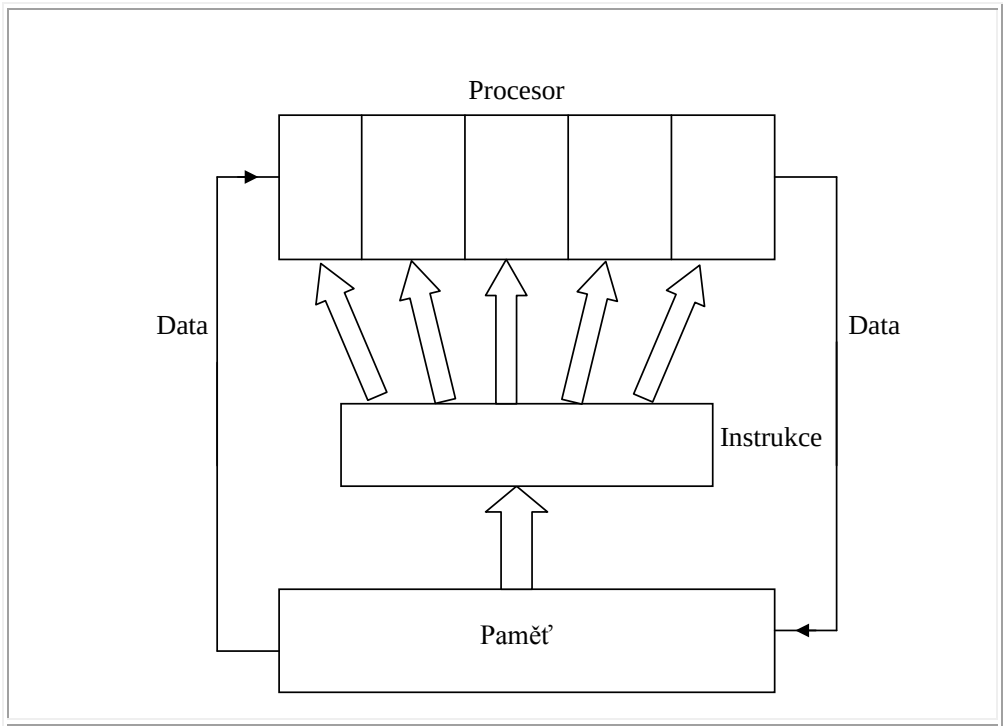
Počítačový systém SISD (single instruction, single data)

Uvedený počítač je shodný s architekturou Von Neumanna. Procesor postupně přebírá instrukce, dekóduje je a provádí.



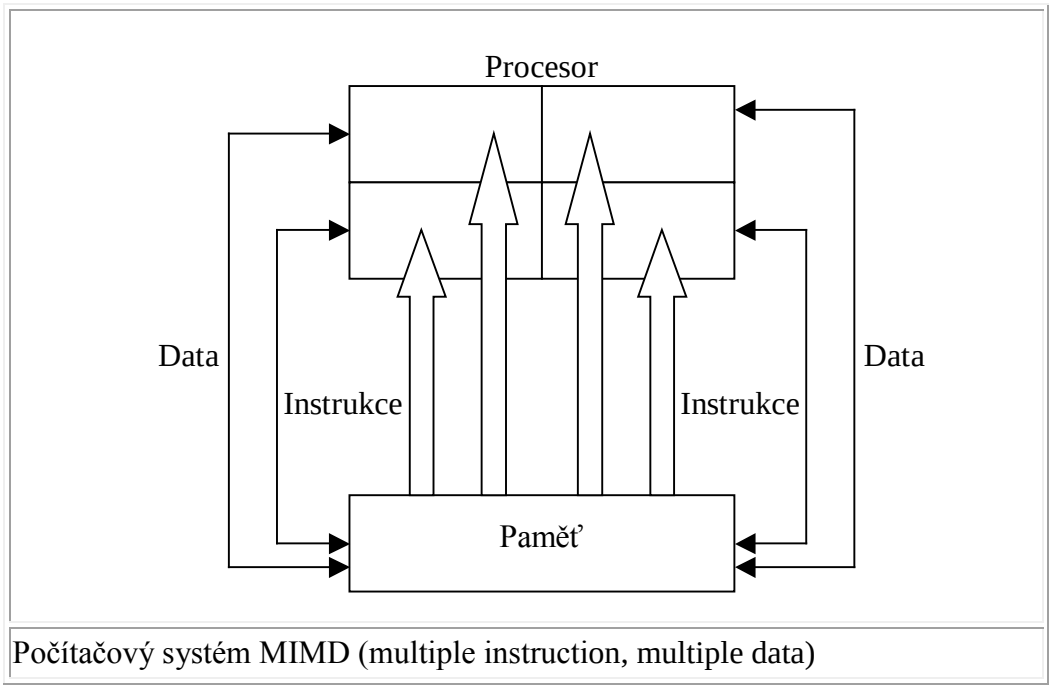
Počítačový systém SIMD (single instruction, multiple data)

Tento počítačový systém se někdy také označuje jako maticový procesor (array procesor). Operace v procesorech se provádějí podle příkazů centrálního řadiče maticového procesoru. Kvůli efektivnímu zpracování je hlavní paměť rozdělena na dva fyzicky samostatné celky: **paměť programů**, která spolupracuje s řídící jednotkou maticového procesoru a **paměť dat**, která komunikuje se všemi procesory. Všechny procesory mohou vykonávat současně tutéž operaci s daty, uloženými v paměti. Proto je maticový procesor užitečný pro operace s maticemi a vektory.



Počítačový systém MISD (multiple instruction, single data)

Tento procesor realizuje základní funkce řetězením operačních modulů, které pracují autonomně. Tím se dosahuje potřebný paralelismus.



Počítačový systém MIMD (multiple instruction, multiple data)

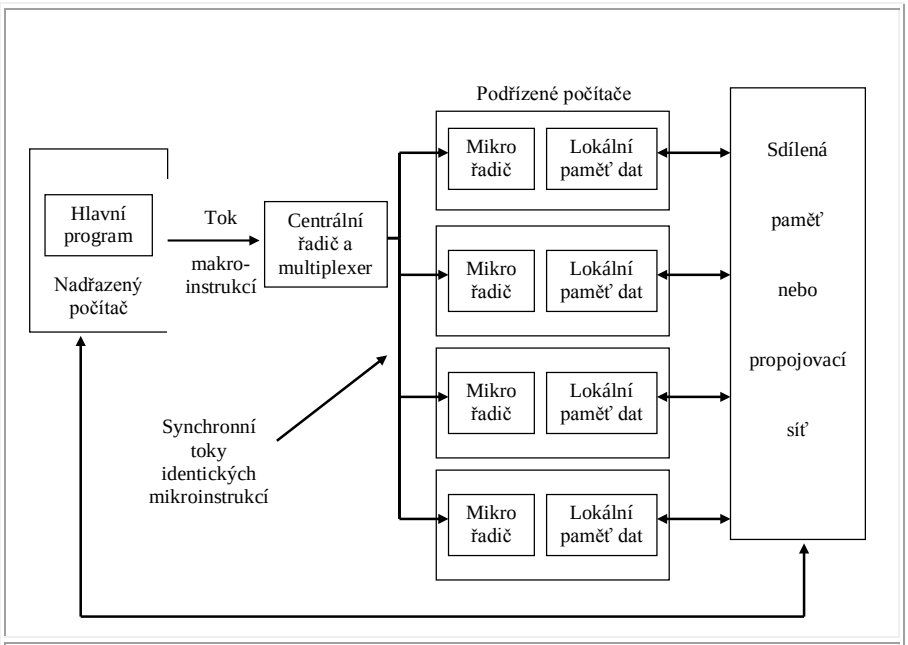
Uvedené multiprocesory mají tyto základní rysy:

- systém obsahuje dva nebo více fyzicky úplně nezávislé procesory, každý s vlastním tokem instrukcí,
- procesory sdílejí společnou operační paměť,
- systém disponuje společným souborem vstupních/výstupních zařízení,

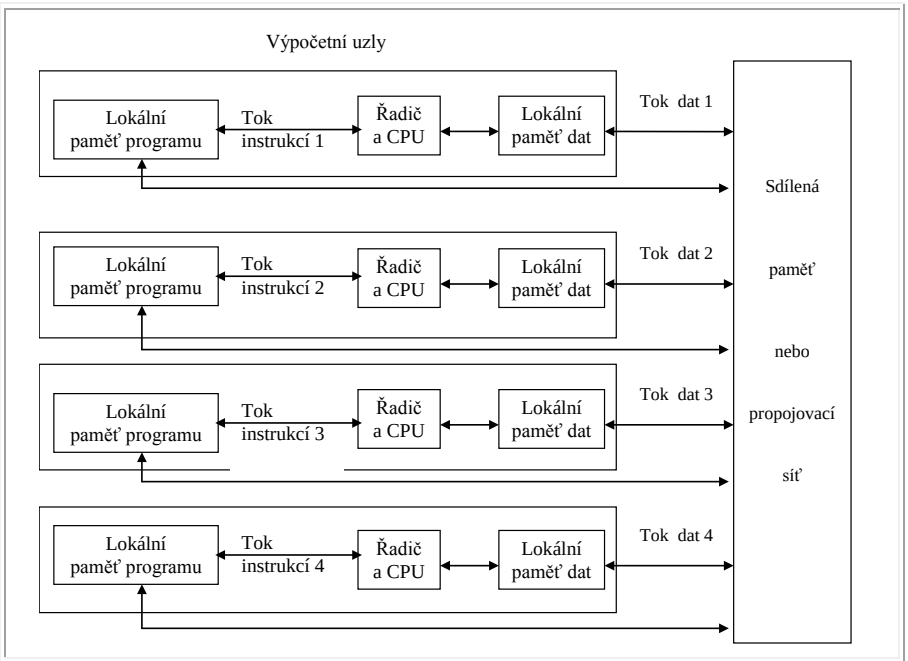
- celý počítačový systém je řízen jedním operačním systémem (jedna řídící autorita).

Každý procesor je schopen samostatně vybírat a provádět instrukce. Multiprocesor tedy může zpracovávat současně více programů, resp. samostatných částí jednoho programu, což se nazývá souběžné zpracování (multiprocessing). Přitom procesory nemusí být stejného typu – pak mluvíme o nehomogenním multiprocesoru. Sdílení hlavní paměti se rozumí ve smyslu možnosti fyzického přístupu každého procesoru do libovolného paměťového místa.

Příklady konkrétní architektury počítačů SIMD a MIMD se 4 výpočetními uzly jsou naznačeny na následujících obrázcích. SIMD a MIMD se typicky liší v složitosti a výkonu výpočetních uzlů. Uzly v MIMD jsou výkonné samostatné počítače, kdežto SIMD uzly jsou obvykle mnohem jednodušší, což je důsledkem jejich podřízenosti v systému.



Architektura SIMD se 4 výpočetními uzly



Architektura MIMD se 4 výpočetními uzly

Kontrolní otázky:

1. Čím se liší von Neumannova koncepce počítače od harwardské koncepce?
2. Kde je uložena adresa právě prováděné instrukce?
3. O kolik míst se posune řádová čárka binárního čísla násobíme-li jej číslem 16?

Úkoly k zamyšlení:

1. U jakých typů instrukcí se nezvyšuje obsah čítače instrukcí o 1?
2. Vedle relativního adresování existuje i samorelativní adresování, kdy bázovou adresou je adresa právě prováděné instrukce. Nakreslete způsob výpočtu konečné adresy.

Korespondenční úkol:

1. Napište program pro výpočet výrazu $Y = A^2 + B^3 - C * D + E / F$
   s jednoadresními, dvouadresními a tříadresními instrukcemi. Operační jednotka vykonává jen základní operace sečítání, odčítání, násobení a dělení.

Shrnutí obsahu kapitoly

V této kapitole jste se seznámili s architekturou číslicových počítačů. Důraz byl kladen na pochopení instrukčního cyklu řadiče a návaznost a hlavní funkce jednotlivých podsystémů počítače. Velká pozornost byla věnována zobrazování jednotlivých informací v počítači a metodám určování operandů instrukcí.

2 Úloha operačních systémů

V této kapitole se dozvíte:

- Proč studujeme operační systémy?
- Jaké jsou základní funkce operačních systémů?
- Jakou úlohu mají operační systémy z hlediska komunikace člověka s počítačem?
- Jaké je typické rozhraní operačních systémů s aplikačními programy?
- Z jakých generických komponent se skládají operační systémy?
- Jaké jsou hlavní funkce jednotlivých komponent operačních systémů?

Po jejím prostudování byste měli být schopni:

- Charakterizovat základní funkce operačních systémů.
- Znát způsob komunikace operačního systému s člověkem a s aplikačními programy.
- Porozumět vrstvené architektuře operačních systémů.
- Popsat oblasti zájmů operačních systémů.
- Definovat funkce jednotlivých částí operačních systémů.

Klíčová slova této kapitoly:

Správce zdrojů, virtuální počítač, multiprogramování, rozhraní člověk/stroj, rozhraní proces/operační systém, generické komponenty.

Doba potřebná ke studiu: 2 hodiny

Průvodce studiem

Studium této kapitoly je jednoduché a popisným způsobem zde nastudujete základní úlohy a funkce operačních systémů.
Na studium této části si vyhraďte 4 hodiny.Po celkovém prostudování a vyřešení všech příkladů doporučujeme vypracovat korespondenční úkol.

Operační systémy jsou jedny z nejrozsáhlejších a nejsložitějších programových systémů ve kterých se uplatňují mnohé vědecké poznatky z oblasti softwarového inženýrství, struktur dat, sítí, algoritmů apod. Během posledních let byly při konstrukci operačních systémů objeveny řady nových metod, které jsou stejně užitečné i v jiných programových aplikacích. Problémy a těžkosti, které se vyskytují při tvorbě efektivních a spolehlivých operačních systémů jsou stejné jako ty, s nimiž se setkávají programátoři či autoři jiných rozsáhlých programů. Čas od času je potřeba operační systém upravit, modifikovat či parametrizovat. Pak je ovšem potřeba jim rozumět a znát algoritmy základních funkcí. Detailní znalost principů operačních systémů je naprosto nezbytná při vytváření těch částí, které jsou závislé na funkčnosti nestandardních technických prostředků. Příkladem jsou ovladače periferních zařízení. Techniky a metodiky tvorby operačních systémů lze s výhodou uplatnit i v jiných oblastech tvorby rozsáhlých programových systémů.

Účelem vzniku operačních systémů bylo zabezpečit programové sdílení prostředků, plánování úloh, plánování a přidělování paměti, ochrana dat a programů, odhalování

chyb při běhu programů. Takto vzniklé operační systémy byly tvořeny množinou automatických a manuálních procedur, umožňující skupině lidí sdílet výpočetní systém, tj. sdílet čas procesoru(ů), operační paměti, periferních zařízení a procesů.

2.1 Služby operačního systému

Jedna z prvních definic operačního systému jej charakterizovala jako programové vybavení nezbytné pro provoz počítače. Tato definice však nic neříká, co je nezbytné pro provoz počítače. Proto si raději definujme operační systém na funkcích a to jako:

- správce zdrojů - resource manager,
- virtuální počítač - virtual machine.

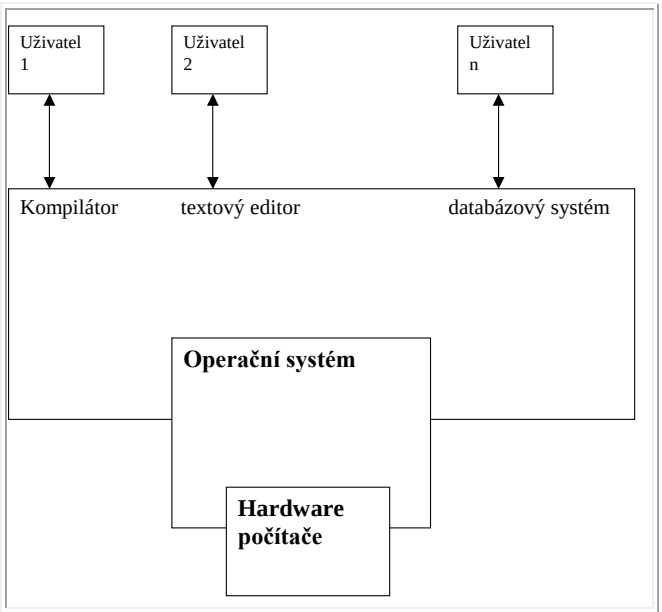
Správce zdrojů. Zdroje jsou vstupní/výstupní (I/O) zařízení, soubory, procesor, paměť apod. Operační systém vlastní jednotlivé systémové zdroje - přiděluje a odebírá je jednotlivým procesům.

Virtuální počítač. Operační systém skrývá detaily ovládání jednotlivých zařízení (transparentnost), definuje standardní rozhraní pro volání systémových služeb. Programátor se může věnovat vlastní úloze a nemusí znovu programovat I/O operace. Program může díky "odizolování" od konkrétních zařízení pracovat i se zařízeními, o kterých jeho autor v době vytváření programu neměl ani ponětí (program se o ovládání I/O nestará).

Další definice operačního systému jej charakterizuje jako:

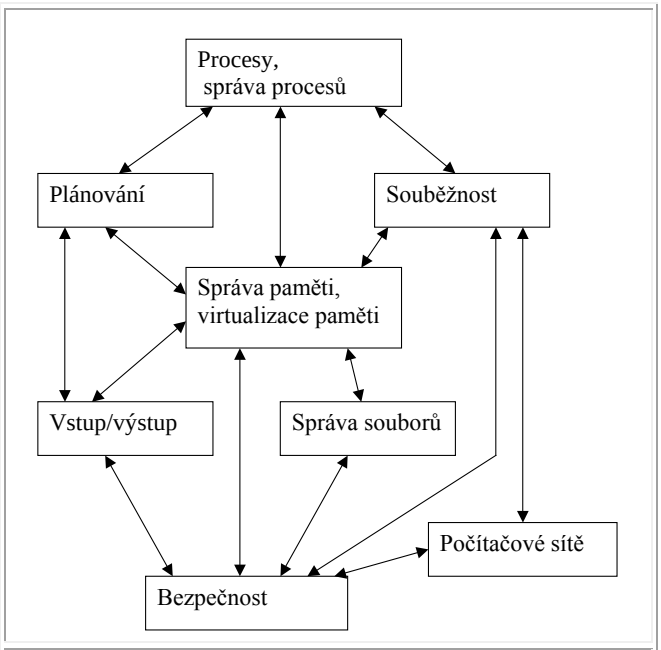
- Správce prostředků – spravuje a přiděluje zdroje systému.
- Řídicí program – řídí provádění uživatelských programů a operací I/O zařízení.
- Jádro – trvale běžící program – všechny ostatní programy lze chápat jako aplikační.

Všechny tyto definice operačního systému oddělují striktně jednotlivé komponenty výpočetního systému, jak je zřejmé z následujícího obrázku.



Abstraktní pohled na systémové komponenty

Z předchozích definic vyplývají oblasti služeb operačních systémů, které jsou zřejmé z následujícího obrázku.

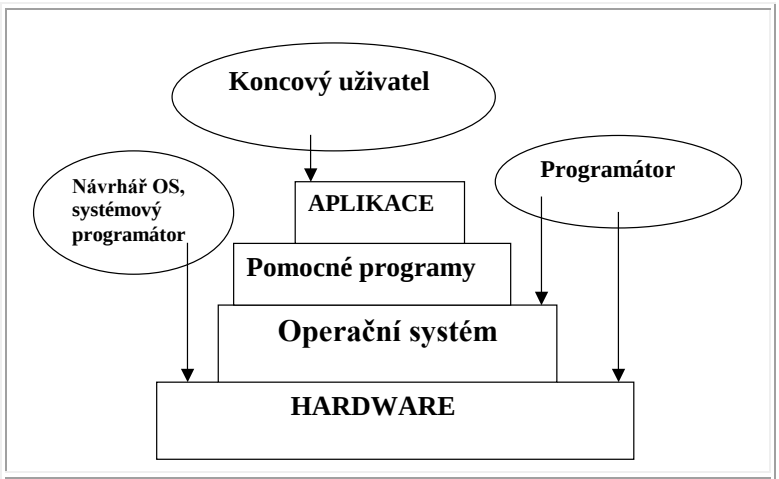


Zájmy operačních systémů

Základním účelem operačního systému je využití sdílených prostředků. Znamená to, že jeden nebo více uživatelů výpočetního systému se budou ucházet o používání fyzických prostředků, konkrétně o sdílení času procesoru event. více procesorů, operační paměti, periferních zařízení apod. Operační systém je z tohoto pohledu množina automatických a manuálních procedur umožňující skupině lidí sdílet výpočetní systém. Každý uživatel získává iluzi, že pracuje s počítačem, který umí provádět jakékoliv programy, tj. že pracuje na virtuálním počítači. Operační systém pak poskytuje každému uživateli vlastní virtuální počítač a navíc chrání každý z těchto počítačů proti destruktivnímu zásahu ostatních. Operační systém přitom nabízí uživateli daleko atraktivnější rozhraní než poskytuje vlastní hardware. Operační systémy jsou rozsáhlé programy zabezpečující multiprogramování, plánování a přidělování paměti, plánování úloh, ochranu dat a programů a odhalování chyb při běhu programů. Operační systém je z tohoto pohledu program, který řídí běh ostatních procesů tzn. ostatním procesům bezpečně a efektivně předává řízení a získává je zpět, sděluje procesoru, kdy má spouštět ostatní procesy. Přitom vytváří rozhraní mezi uživatelem a hardware a skrývá ostatním procesům detaily o hardware tj. musí zvládnout správu detailů hardware ve své režii.

2.2 Struktura operačního systému

Cílem operačního systému je rovněž zajištění pohodlnosti používání počítače, tzn. že operační systém je správcem rozhraní člověk/stroj a správcem rozhraní proces/operační systém. Operační systém je z tohoto pohledu tvůrcem virtuálního počítače a skrývá tak detailní pravdu o holém počítači (hardware).



Rozhraní člověk stroj

Operační systém z hlediska rozhraní člověk/stroj typicky poskytuje služby pro:

- vytváření programů na uživatelském rozhraní (editory, kompilátory, sestavovací programy, ladící programy, apod.),
- provádění programů, tj. zavádění programů do operační paměti a jejich spouštění,
- běh procesů, podpora komunikace a synchronizace procesů,
- zpřístupňování vstupních/výstupních zařízení a operací na nich,
- řízení přístupů k souborům,
- řízení přístupu k systému,
- detekce chyb a chybové řízení (chyby hardware, programové),
- protokolování činností.

Rozhraní proces/operační systém je typickým rozhraním služeb jádra, což jsou služby:

- unifikace vstupních a výstupních operací,
- virtualizace paměti,
- ochrana a reakce na chyby,
- protokolování a řízení přístupu ke zdrojům,
- synchronizace procesů,
- komunikace mezi procesy.

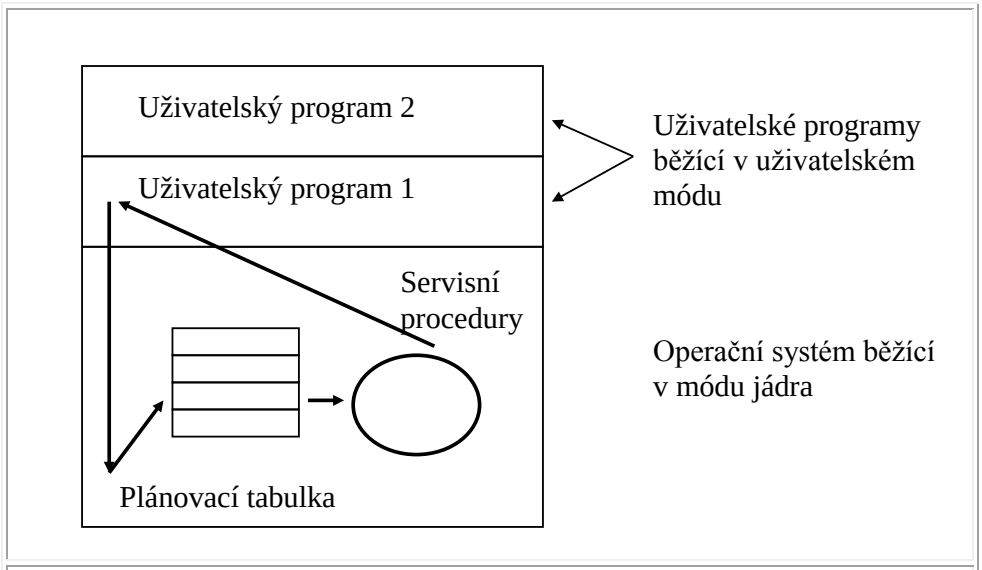
Dalším cílem operačního systému je zajištění dostatečné výkonnosti a efektivity. Operační systém je tak správcem systémových prostředků (procesorů, paměti, vstupních a výstupních zařízení a souborů) a správcem jejich užívání. Operační systém tak řídí přístup k vstupním a výstupním zařízením a souborům, provádí správu paměti a určuje, který program bude používat procesor. Operační systém konečně musí zajistit schopnost vývoje tj. doplňování a vývoj hardware, doplňování nových služeb do počítačového systému. Operační systém musí být schopen reagovat na inovace v technických prostředcích, na nové komponenty počítačů.

Operační systémy, z hlediska své struktury, lze popsat architekturou:

- monolitickou,
- víceúrovňovou,
- virtuální,
- klient-server.

2.2.1 Monolitická architektura operačního systému

Monolitickou architekturu operačního systému lze vyjádřit následujícím obrázkem.



Služby monolitického operačního systému v operační paměti

Z obrázku vyplývá určitá „vnitřní strukturovanost“, kdy hlavní program volá obslužné procedury pro jednotlivá systémová volání. Systémová volání přepínají zpracování z uživatelského módu do módu servisního (mód jádra operačního systému).
Systémová volání u obecného operačního systému lze rozdělit do několika skupin:

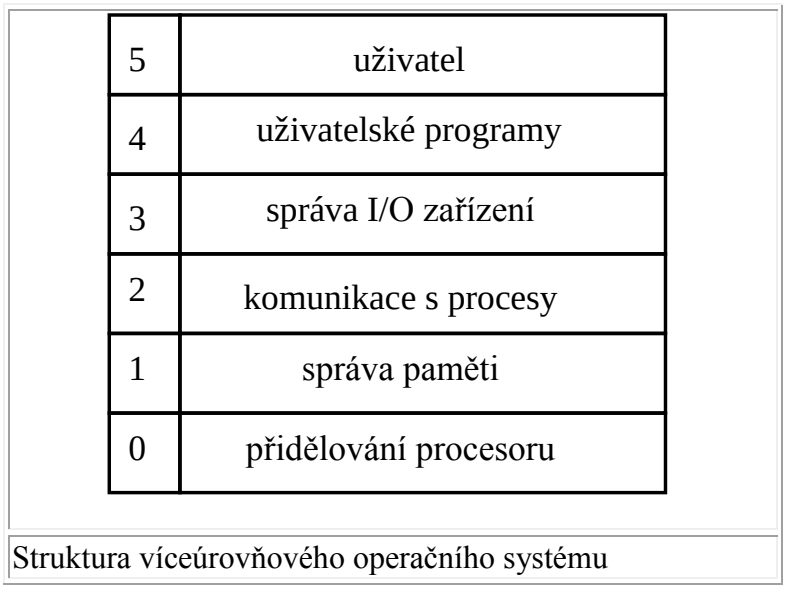
- **řízení procesů** (vytvoření procesu, výměna procesů v paměti, ukončení zpracování procesu apod.),
- **programová přerušení (**nastavení přerušení pro jeho zachycení nebo ignoraci, přerušení procesu, plánování přerušení v čase, suspendace volaného procesu až do příštího přerušení),
- **řízení souborů** (vytvoření nebo zrušení souboru, vytvoření adresáře, otevření souboru pro čtení nebo zápis, zavření souboru,čtení dat ze souboru do vyrovnávací paměti, zápis dat do souboru z vyrovnávací paměti apod.),
- **řízení adresářů** a systému řízení souborů (vytvoření nového vstupu adresáře, přemístění vstupu adresáře, připojení systému souborů, přesun diskových bloků ze zásobníku v paměti na disk, změna pracovního adresáře, změna kořenového adresáře),
- **ochrana** (změna bitů ochrany souboru, získání identifikace uživatele, získání identifikace skupiny, nastavení identifikace uživatele, nastavení identifikace skupiny, změna vlastníka souboru nebo skupiny, nastavení nebo zrušení masky ochrany),
- **řízení v čase** (nastavení času ukončení, získání času ukončení, nastavení času posledního přístupu k souboru, získání uživatelského a systémového času pro další použití).

Nevýhodou monolitické architektury operačního systému je realizace jádra jako velkého programového celku, ve kterém lze snadno implementovat chyby, tím je obtížné ladění funkčnosti a složitá rozhraní procedur mající jednoznačnou vazbu na

použitý hardware počítače. Výhodou je relativně snadná implementace operačního systému a zabezpečení velkého výkonu i na poměrně nevýkonném hardware.

2.2.2 Víceúrovňová architektura operačních systémů

Řada moderních operačních systémů je vytvářena na principu hierarchického uspořádání služeb dle příkladu na následujícím obrázku.



Struktura víceúrovňového operačního systému

Hierarchické uspořádání služeb operačního systému je založeno na následujících principech:

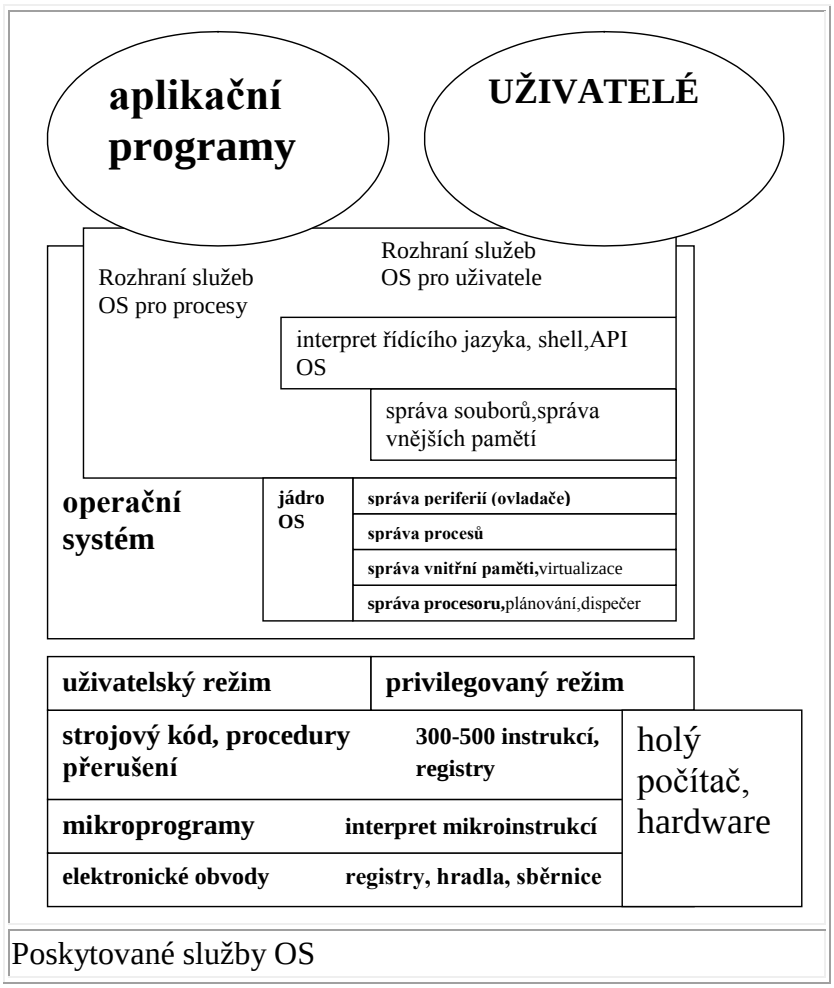
- operační systém se dělí do jistého počtu vrstev (úrovní),
- každá vrstva je budována na funkcionalitě nižších vrstev,
- nejnižší vrstva (0) je hardware,
- nejvyšší vrstva je uživatelské rozhraní,
- pomocí principu modularity jsou vrstvy navrženy tak, aby každá používala funkcí (operací) a služeb pouze vrstvy $n - 1$ tj. vyšší vrstva využívá pouze služeb nejbližší nižší vrstvy a nabízí služby nejbližší vyšší vrstvě.

Pomocí této vrstvené architektury lze řešit problém přílišné složitosti velkého operačního systému a to tak, že se realizuje dekompozice velkého problému na několik menších zvládnutelných problémů. Přitom každá úroveň řeší konzistentní podmnožinu funkcí, nižší vrstva nabízí vyšší vrstvě „primitivní“ funkce (služby) a nižší vrstva nemůže požadovat provedení služeb vyšší vrstvy. Používají se přesně definovaná rozhraní a tím lze každou vrstvu uvnitř modifikovat, aniž to ovlivní ostatní vrstvy (principy modularity).

Nevýhodou takové architektury je především vyšší systémová režie a tím pomalejší vykonávání systémových volání. Protože efektivita hraje v jádře operačního systému významnou roli, je třeba volit kompromis v počtu vrstev tj. definovat pouze omezený počet úrovní pokrývající vyšší funkcionalitu. Výhodou takové architektury je přehlednost a oddělené datové struktury.

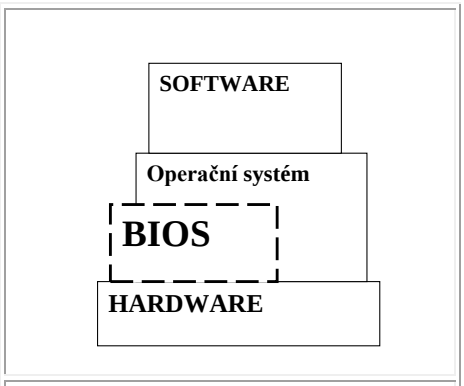
Typická struktura operačních systémů je hierarchická, čímž se řeší problém přílišné složitosti. Dekompozice velkého problému na několik menších umožňuje zvládnout

řešení složitého operačního systému. Každá takto dekomponovaná úroveň řeší konzistentní podmnožinu funkcí, kde nižší vrstva nabízí vyšší vrstvě primitivní funkce (služby) a přitom nižší vrstva nemůže požadovat provedení služeb vyšší vrstvy. Rozhraní mezi vrstvami musí být přesně definovaná, což umožní modifikovat každou vrstvu uvnitř, aniž to ovlivní ostatní vrstvy. Typická struktura vrstev operačního systému je zřejmá na následujícím obrázku.



Poskytované služby OS

Typickým příkladem hierarchické struktury podsystémů je návaznost obecného operačního systému na hardware počítače typu PC. Struktura návaznosti je zřejmá z následujícího obrázku.



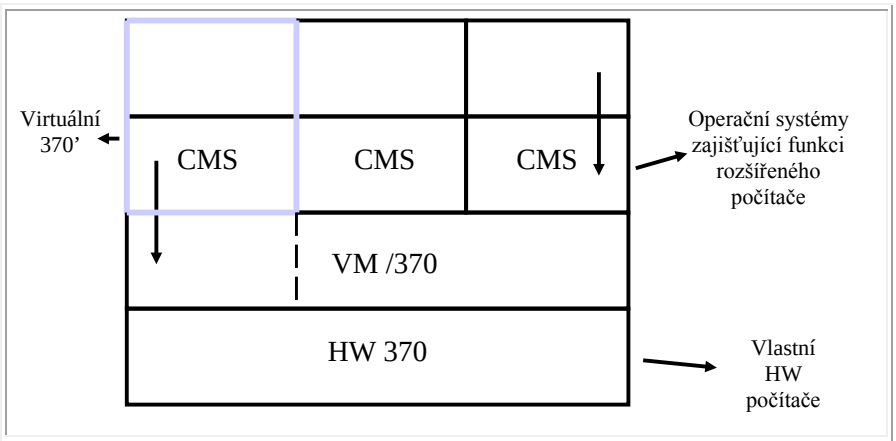
Rozhraní v PC

BIOS - Basic Input Output System, jak už vyplývá z názvu, zabezpečuje základní služby při přístupu k periferním zařízením. Jinými slovy, tvoří rozhraní mezi hardwarem a vyššími vrstvami programového vybavení. Je to rozhraní standardizované, tzn. že vstupní body a parametry obslužných procedur operačního systému nezávisí na typu hardware a je jednoznačně navázáno na funkce operačního systému. BIOS zabezpečuje v počítači ještě další úkoly:

- provádí úvodní test po spuštění počítače,
- umožňuje nastavit základní parametry počítače,
- zavádí operační systém,
- poskytuje operačnímu systému prostředky pro realizaci víceúlohového prostředí.

2.2.3 Architektura operačního systému s virtuálními počítači

Operační systémy navrhované na principech virtuálních počítačů jsou převážně určeny pro režim sdílení času (time-sharing). Operační systém tak emuluje existenci více jednouživatelských systémů s vlastním hardwarem (virtual machine monitor - VM), které jsou přesnou kopií skutečného hardware HW (VM vrstva emuluje HW). Pro realizaci je však třeba **rozšířených vlastností rozhraní** procesoru a oddělení funkce multiprogramování od funkce rozšířeného stroje (extended machine). Na následujícím obrázku je naznačena činnost systému zajišťující funkci rozšířeného počítače pomocí *CMS (Conversational Monitor System)*, což je interaktivní systém pro sdílení času mezi uživateli. Když *CMS* zpracovává systémová volání převede jej do operačního systému vlastního virtuálního počítače. Tyto instrukce jsou pak převáděny do OS jako část vlastních simulací na skutečném hardwaru.



Struktura operačního systému s virtuálními počítači

Jádro systému tvoří monitor virtuálního počítače, který pracuje s prostým hardwarem a zajišťuje multiprogramování s použitím ne jednoho, ale několika virtuálních počítačů na vyšší úrovni. Virtuální počítače nejsou počítače se systémem souborů a ostatními možnostmi. Jsou pouze přesnými kopiemi prostého hardwaru, zahrnující mód jádra a mód uživatele, vstupů a výstupů, přerušení a vše čím je skutečný počítač vybaven.

Každý virtuální počítač je identický se skutečným hardwarem, každý z nich může pracovat s nějakým operačním systémem, který běží přímo na skutečném hardwaru. Nevýhodou této architektury může být zpomalení poskytovaných služeb operačním systémem a to z důvodu emulace. Výhodou je možnost koexistence více operačních

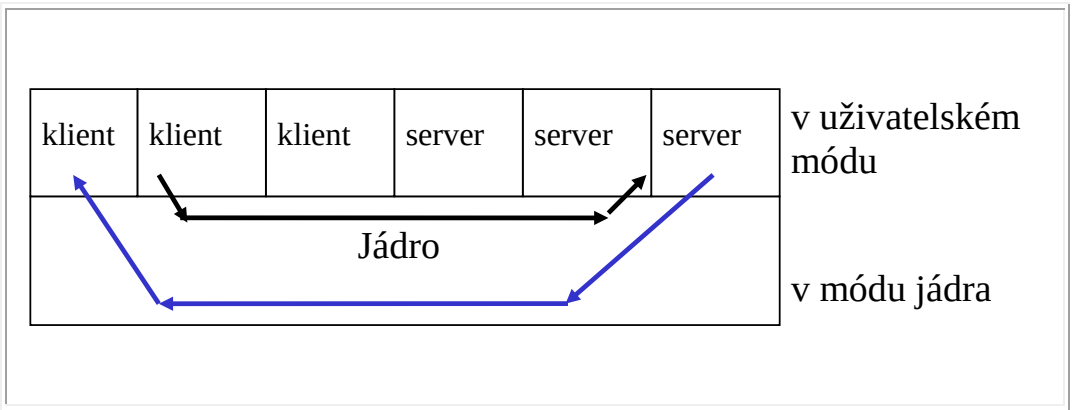
systémů na jednom hardware a využití programového vybavení z jiného hardwarového systému.

2.2.4 Architektura operačního systému s modelem klient-server

Architektura operačních systémů na principu komunikace klient-server vznikla z potřeby minimalizace rozsahu jádra. Cílem je implementovat většinu funkcí operačního systému v uživatelských procesech, tedy v uživatelském módu. Například při požadavku na službu, jako je čtení bloku dat ze souboru, zašle uživatelský proces (nyní *klient* proces) požadavek na obsluhu souborů (*file server*), která vykoná práci a zašle zpět přečtená data jako odpověď na požadavek. Tzn. že mikrojádro zajišťuje zejména komunikaci.
Základní struktura takového operačního systému může být následující:

- **obsluha klienta (***klient process),*
- **obsluha procesů** *(process server),*
- **obsluha terminálů** *(terminal server),*
- **obsluha souborů** *(file server),*
- **obsluha paměti** *(memory server).*

Na následujícím obrázku je naznačena struktura takového operačního systému a způsob komunikace.



Struktura operačního systému klient-server

Skutečné rozdělení na jednotlivé obsluhy – servery může být samozřejmě v každém systému jiné, jednotlivé servery mohou být vytvářeny i uživatelem. Po rozdělení operačního systému na jednotlivé části se pak při vyvolání požadované služby aktivuje vždy pouze obsluha příslušné části operačního systému. Jednotlivé části operačního systému nejsou rozsáhlé a jsou takto snadno řiditelné. Většina služeb operačního systému pak pracuje v uživatelském módu a ne v módu jádra systému.
Použití systému *klient-server* je velmi výhodné zejména pro distribuované operační systémy. U takového systému jádro příslušného systému řídí pouze přenos zpráv od klienta k příslušné obsluze – serveru. Ve skutečnosti to není možné, neboť některé funkce operačního systému, např. práce s vstupními a výstupními jednotkami, nelze provádět z uživatelského programu. Řeší se to pak tak, že tyto kritické procesy se zpracovávají v módu jádra.

Výhodou architektury klient-server je v oddělení adresních prostorů procesů a tím dobrá strukturovanost operačního systému, odolnost systému v případě selhání serverového procesu a použitelnost v síťovém a distribuovaném prostředí.

2.2.5 Charakteristiky moderních operačních systémů

Moderní operační systémy musí reagovat na rozvoj hardware, zvýrazněný multiprocesorovými stroji, připojením na vysokorychlostní sítě, rychlejšími procesory a větší kapacitou paměti.

Při vytváření operačních systémů se vyžaduje aplikovat nové softwarové technologie podporující multimédia, webovské aplikace, internet a technologie klient/server. Dalším směrem vývoje jsou distribuované operační systémy, vytvářející ilusi jedné společné paměti, jednoho paralelního stroje, distribuovaného systému souborů s jednotným pohledem na ně. Zde je směr vývoje pomocí objektově orientovaných nástrojů s cílem modulárního rozšiřování funkcionality celého operačního systému, postaveného na malém jádře a přizpůsobování se konkrétním potřebám bez porušení integrity celého systému.

Pro uživatele to znamená, aby operační systém byl snadno použitelný, snadno naučitelný, spolehlivý, bezpečný a rychlý. Z hlediska systémového návrhu se jedná jeho snadný návrh, implementaci, udržovatelnost, přizpůsobivost, spolehlivost a bezchybnost.

Tradičně bývaly operační systémy napsaný v symbolickém strojovém jazyku (assembleru), v současnosti se stále častěji píší v běžných programovacích jazycích vysoké úrovně (obvykle C/C++). Tím se docílí rychlejšího naprogramování, kompaktnějšího výsledku, srozumitelnějšího a tím snadnějšího ladění a snadnější přenositelnosti na jinou architekturu

2.3 Historie operačních systémů

V dřívějších dobách operační systémy jako takové neexistovaly a programátor musel komunikovat s počítačem pomocí strojového kódu (na úrovni 1-0). Navíc ještě musel znát přesnou konfiguraci počítače a jednotlivých připojených zařízení. Postupem času s rozvojem výpočetní techniky se však tento postup stal neúnosným, a tak začaly vznikat první programovací jazyky. Programovací jazyky mohou být buď vázány na konkrétní hardware (např. assembler), nebo jsou na hardware nezávislé (tzv. vyšší programovací jazyky). Nejstarším vyšším programovacím jazykem byl Short Code z roku 1949, později vznikl Fortran (1956, vyvinut IBM), COBOL (1959), BASIC (1965, později standardní jazyk pro PC), Pascal (1971), C (1972) atd.

S dalším rozvojem však bylo třeba programu, který by sám zvládal základní funkce systému a ulehčil programátorům práci. Počátkem 60. let tak pomalu začaly vznikat operační systémy a v polovině 60. let (s příchodem minipočítačů) již vyvstává potřeba takových operačních systémů, jaké známe dnes. Mezi nejznámější a nejrozšířenější platformy patří především operační systém Windows od firmy Microsoft a operační systémy na bázi UNIXu, vyvinutého původně firmou AT&T. Mezi nejznámější operační systémy na bázi UNIXu patří Linux a Mac OS X.
Významným prvkem ve vývoji operačních systémů bylo vytvoření grafického uživatelského rozhraní (GUI - graphics unit interface), které má své kořeny v padesátých letech, ale rozvinulo se až v sedmdesátých letech, kdy skupina v Xerox Palo Alto Research Center (PARC) vyvinula Alto, počítač založený na GUI. Skutečným mezníkem byl ale až GUI vyvinutý společností Apple pro její počítač Lisa,

resp. Macintosh, který jako první tuto myšlenku již v roce 1983, resp. 1984 zpřístupnil masám. Pro kancelářské využití je dále účelné se zaměřit pouze na OS s GUI.

2.3.1 Operační systémy Windows

Historie operačního systému Windows sahá do roku 1981, kdy firma IBM uvedla na trh první PC spolu se 16bitovým operačním systémem MS-DOS (Microsoft Disk Operating System). Pro další vývoj počítačového průmyslu a šíření operačních systémů od Microsoftu se ukázalo jako rozhodující, že firma IBM umožnila výrobu klonů svých počítačů - velmi rychle se pak rozšířily do celého světa.

Ačkoliv MS-DOS v podstatě umožnil masivní rozvoj mikropočítačů, byl již v době svého vzniku nepohodlný a z hlediska návrhu nedostatečný a v podstatě zastaralý. MS-DOS podporoval pouze jednoho připojeného uživatele, který mohl v čase pracovat s pouze jedním programem (chybějící podpora víceúlohového režimu). MS-DOS měl hardwarová omezení, např. nedokázal pracovat s pamětí větší než 640 kB nebo s disky většími než 30 MB. Většina nedostatků MS-DOSu, především uživatelská nepřívětivost a absence víceúlohového režimu, byla postupně překonána pozdějšími verzemi systému Windows.
Významným mezníkem ve vývoji počítačů bylo uvedení počítače Apple Lisa s operačním systémem LisaDesk na bázi GUI v lednu 1983. U Microsoftu začal vznikat OS Windows jako GUI nadstavba MS-DOS. První verze OS s GUI od Microsoftu - Windows 1.0 byla uvedena na trh 20. listopadu 1985. Tato nadstavba OS DOS sice nebyla příliš použitelná, ale např. již nenutila uživatele ukončovat a znovu spouštět programy. Pokud chtěl s programy pracovat současně, mohl se mezi nimi přepínat, avšak okna se nemohla překrývat. Tento nedostatek byl odstraněn ve verzi 2.0, která byla uvedena na trh v roce 1987. Zde již bylo možno okna překrývat jedno přes druhé, bez nutnosti je mozaikovitě skládat vedle sebe.

Komerčního úspěchu a reálné použitelnosti se ale dočkal až OS MS Windows 3, uvedený na trh v roce 1990, který lze označovat za první reálně použitelný Windows s GUI. Windows verze 3.0 se velmi rychle rozšířily, především díky velké hardwarové i softwarové podpoře významných nezávislých výrobců a předinstalováváním na nová PC. V roce 1991 byla vydána rozšířená verze Multimedia Extension pro práci Windows s multimédii. Na jádře pracujícím pod DOSem byly na trh později uvedeny další OS Windows:

- Windows 3.1 z dubna 1992 s řadou vylepšení (jako podpora OLE, vylepšení správce souborů, vylepšení podpory tiskáren, nových ovladačů podporujících MS-DOS grafiku v okně, virtuální paměť lze měnit v ovládacím panelu, atd.) a odstraněním některých chyb v uživatelském rozhraní. Následně byly vydány i Windows 3.11, kde byla přidána síťová podpora.
- Windows 95 uvedený v srpnu 1995. Tento systém byl opatřen řadou vylepšení, například částečně 32bitovým jádrem (hybridní 16/32bitové), podporou dlouhých názvů souborů, lepší podporou sítí (byl integrován protokol TCP/IP) a zcela novým grafickým rozhraním (GUI Windows 95 bylo tak úspěšné, že se jej Microsoft rozhodl využít i ve verzi Windows NT 4.0). Lze jej označit za povedenou, klíčovou a přelomovou verzi Windows (bývá označován za první uživatelsky přívětivý operační systém od Microsoftu).

- Windows 98 - již 32bitový, uvedený v červnu 1998. Tento systém přinesl hezčí grafické prostředí, integraci Webu do oken průzkumníka, již plně funkční podporu USB, nový Microsoft Explorer 4.
- Windows 98 SE, uvedený v roce 1999 coby druhé vydání Windows 98 (Second Edition), byl již stabilním a kvalitním systémem.
- Windows Me (Millenium Edition) - je posledním zástupcem této kategorie, který již restart do režimu čistého MS-DOSu neumožňuje (obsahuje nové grafické prostředí z Windows 2000 Profesional, Windows Media Player 7 a nové DirectX).

Operační systémy na základě DOSu však měly koncepční nedostatky. Již v roce 1987 proto IBM ve spolupráci s Microsoftem začal vytvářet nový operační systém pro PC, tentokrát již nezatížený nedostatky DOSu a nazvaný OS/2. Spolupráce obou firem se ale rozpadla a každá z nich si vyvíjela svou vlastní verzi OS/2 (Microsoft tu svou záhy přejmenoval na Windows NT a IBM na OS/2 Warp). V polovině roku 1993 se tak dostávají na scénu Windows NT, která jsou zaměřená především na náročné uživatele a servery (zkratka NT znamená New Technology). Pátou verzi Windows NT uvedl Microsoft na trh 17. února 2000. Přesto, že je tento systém volným pokračováním Windows NT 4.0, obsahoval tolik změn, že byl přejmenován na Windows 2000. Pro většinu domácích uživatelů je však tento systém příliš robustní a náročný na hardware. Bývá nasazován především na výkonných serverech.

Až v roce 2001 Microsoft konečně uvedl na trh operační systém pro běžné uživatele postavený nikoliv na DOSu, ale na přizpůsobené technologii NT, nazvaný Windows XP (eXPerience). V tomto případě Microsoft oznámil, že se jedná o nástupce operačních systémů, který si z obou větví má vzít jen to nejlepší. Z větve NT to má být vysoká spolehlivost, z větve 9x pak kompatibilitu a podporu multimédií (až do verze XP nebyly aplikace pro řadu NT kompatibilní s aplikacemi pro řadu 9x a naopak). Jeho další odlišností oproti předchozím verzím je nová podoba grafického rozhraní, a možnost jeho změny (skinovatelnost); celý systém je multimediální, umožňuje práci s videem, zvuky a Internetem. V některých operacích je Windows XP až o 30% pomalejší než na starém jádře pracující Windows Me, přesto je to však (za předpokladu doinstalování service packů) významný krok směrem vpřed na poli OS Windows.
Operační systém Windows je dnes zejména díky masivním marketingovým kampaním a podpoře nezávislých výrobců HW v 90. letech nejrozšířenějším OS v oblasti kancelářských počítačů. Systémy Windows různých verzí jsou instalovány na cca 90% všech kancelářských počítačů. Mezi největší výhody MS Windows patří právě jejich masivní rozšíření. To je však zároveň pro Microsoft i omezením - vývoj trpí velkou setrvačností (viz např. problém roku 2000 "Y2K" - pouze dvouciferné ukládání letopočtu) a omezenými možnostmi Microsoftu zavádět moderní technologie ještě v době, kdy jsou nové (viz například setrvávání u BIOSu). Součástí MS Windows je množství aplikací pro běžnou kancelářskou práci (MS Explorer, základní multimediální aplikace, atd.). Vývoj aplikací pro platformu Windows je co do míry standardizace a chování jednotlivých aplikací různorodý.
Od verze Windows 2000, resp. Windows XP, se nevyskytují výrazné problémy se stabilitou systému. Pro dostatečnou bezpečnost systému je potřeba důsledně dbát na pravidelné aktualizace a jeho dobrou správu více než u méně rozšířených systémů. Aktualizace systému bývá obvykle v intervalech 1 měsíce.

2.3.2 Operační systémy UNIX

UNIX byl poprvé představen Kenem Thompsonem a Dennisem Ritchiem v článku publikovaném v roce 1974 v „Communications of the ACM“. Na trh byl však uveden již v roce 1969. V roce 1964 Bell Telephone Laboratories zahájily projekt MULTICS (Multiplexed Information and Computing Service). Cílem projektu bylo vytvořit OS pro rozsáhlý počítač s velkým množstvím uživatelů. Projekt MULTICS byl v sice roce 1969 ukončen, ale jeho bývalí programátoři (Ken Thompson, Dennis Ritchie a Brian Kernighan) přišli s myšlenkou jednoduchého, elegantního a snadno ovladatelného OS a již o rok později byl nový OS dokončen (v jazyce assembler) a nazván UNICS. (Tento název vznikl z "Universal Information and Computing Service", je to slovní hříčka na název MULTICS.) Později byl název pozměněn na UNIX.

V roce 1973 byl UNIX přepsán do jazyka C, aby byl snadno přenositelný mezi platformami. Zdrojové texty UNIXu byly poskytnuty universitám (např. universitě v Berkely). Vznikly dvě hlavní větve UNIXu:

- AT&T (původní verze od American Telephone and Telegraph company) - dnes UNIX System V (od Unix System Laboratories)
- BSD (vytvořený na universitě v Berkeley; BSD = Berkeley Software Distribution) - dnes UNIX BSD 4.4

V roce 1984 se začaly objevovat první pokusy o standardizaci UNIXu. Vzniklo sdružení GNU (název GNU je rekursivní zkratka pro "GNU is Not UNIX"). Toto sdružení podporuje "svobodný software" a vytvořilo GPL (General Public Licence), která je legislativním prostředkem pro zaručení "svobody" softwaru.

Později vznikla řada standardů POSIX (Portable Operating System Interface), která popisuje obecný OS (standardy např. definují systémová volání, knihovní funkce a chování programů v „POSIX kompatibilním“ operačním systému). I přes tyto standardizační snahy je ve světě Unixu mnoho různých odchylek a „vylepšení“. Přesto lze konstatovat, že POSIX spolu s dalšími standardy přinesl potřebný řád.

V roce 1989 byl na bázi UNIX BSD uveden nový plně objektový operační systém NeXTStep 1.0 s jádrem Mach (tento operační systém byl úzce svázán s počítači NeXT). V roce 1996 firma NeXT vytvořila na základě částečně přepracovaného NeXTStepu otevřený standard objektového API OpenStep, který koncepčně vychází z původního NeXTStepu, výrazně však rozšiřuje jeho služby. Na OpenStep pak navázala iniciativa GNUStep, jejíž snahou je vytvořit API, konformní se standardem OpenStep, které by chodilo prakticky kdekoli a bylo volně k dispozici v rámci licence GNU. V roce 1997 nakonec došlo k odkoupení firmy NeXT firmou Apple, která následně použila NeXTStep/OpenStep 4.2 pro vývoj nového operačního systému pro počítače Apple Macintosh.

V devadesátých letech začaly vznikat nekomerční systémy na bázi UNIXU. V těchto letech vznikaly NetBSD, FreeBSD, OpenBSD (nekomerční UNIXy typu BSD) a také LINUX (reprezentující dnes mezi nekomerčnímy systémy na bázi UNIXU druhou větev Unixu zvanou System V).

V roce 1999, resp. 2000 firma Apple Computer uvedla nový komerční operační systém Mac OS X server, resp. Mac OS X navazující na technologie NextStepu/OpenStepu, který má unixové jádro Darwin (postavené na technologiích

jako FreeBSD nebo kernel Mach 3.0). Části Darwinu jsou Open Source s licencí APL (Apple Public Licence) nebo GPL.

V současnosti se UNIX využívá na universitách a v komerční informatice (internetové aplikace, mobilní komunikace atd.), ale mezi běžnými uživateli není rozšířen. Šíří se spíše UNIXové klony, resp. operační systémy typu UNIX. Zde mezi nejvýznamnější patří Linux a Mac OS X.

2.3.3 Operační systémy Mac OS a Mac OS X

Historie operačního systému Mac OS sahá do roku 1977, kdy firma Apple Computers vyprodukovala komerčně úspěšný počítač Apple II, první all-in osobní počítač (v plastové krabici a s barevnou grafikou), resp. do roku 1983, kdy byl uveden počítač Apple Lisa s OS LisaDesk na bázi GUI. Skutečný Mac OS 1.0 však byl uveden na trh až v lednu roku 1984 spolu s prvním počítačem Macintosh. Mac OS byl velmi pokrokovým operačním systémem a kromě GUI obsahoval i další moderní prvky - ovládání myší, multitasking, multimedia, podporu práce v sítích, atd.

Mac OS byl do roku 2002 postupně uveden v 9 verzích (poslední verze 9.2). V roce 1991 vyšel v té době velmi pokročilý Mac OS 7 (jednalo se mimo jiné o plný 32bitový systém). V roce 1994 Apple oznámil práce na zcela novém operačním systému s kódovým označením Copland. Na svou dobu měl Copland mnoho převratných designových prvků jako skutečné mikrojádro a hardwarovou abstrakci. Projekt se však dostal po řadě peripetií do slepé uličky a v srpnu 96 byl zrušen vývoj.

V té době však již byla značná potřeba nového OS - Mac OS 7 po 5 letech sice procházel drobnými updaty (až k verzi systému 7.6), ale již nevyhovoval v mnoha ohledech, od neplnohodnotného multi-taskingu po nestabilitu posledních systémů 7.x (neuvedením Coplandu a uvedením Windows 95 od Microsoftu Apple přišel o své vůdčí postavení u OS s GUI a u řady vlastností OS byl Microsoftem předstižen). V roce 1997 byl proto Mac OS 7 nahrazen Mac OS 8, do kterého byly začleněny některé technologie Coplandu a který do značné míry vyřešil potřeby uživatelů. Mac OS 8.6 představil multitasking na úrovni kernelu (jádra). V roce 1998 byl uveden Mac OS 9, který je ve formě emulační vrstvy Classic spustitelný na Mac OS X dodnes. Mac OS 9 byl vyvíjen až do roku 2002, kdy byla jeho poslední verze 9.2 nahrazena zcela novým operačním systémem Mac OS X.

Historie Mac OS X sahá do roku 1997, kdy Apple odkoupil společnost NeXT a rozhodl se použít její OS NeXTSTEP (objektově orinetovaný operační systém na bázi Unixu, vybavený vlastním grafickým rozhraním) jako základ pro svůj nový OS Mac OS X. Mac OS X - na trhu od roku 2000 (Mac OS X server již od roku 1999) - je moderní objektově orientovaný systém založený na kvalitním a stabilním základu BSD Unix, vybavený novým vektorovým grafickým rozhraním Aqua GUI. Uvedením Mac OS X Apple navázal na někdejší úspěchy GUI Mac OS (i v současné době je vzhled rozhraní Mac OS X - Aqua GUI napodobován dalšími výrobci operačních systémů a aplikací, včetně Microsoftu). Mac OS X zajišťuje plnou kompatibilitu s aplikacemi napsanými pro původní Mac OS 9.2 (aplikace lze nativně spouštět).

V pozadí nového uživatelského rozhraní stojí jádro OS Darwin, otevřená základna na bázi UNIXu, postavená na takových technologiích jako Mach nebo FreeBSD. Nad

Darwinem/XNU stojí set služeb a knihoven, převzatých většinou z NextStepu, které se starají o grafické rozhraní a uživatelské aplikace. Mac OS X nabízí kompletní implementaci systému X Window pro aplikace založené na X11 (umožňuje instalaci běžných Linuxových aplikací pro prostředí X11).

Technologické vlastnosti Mac OS X umožňují emulovat chod původního Mac OS 9.2 (nativní emulační vrstva Classic) i chod běžných emulátorů ostatních platforem (Virtual PC; iEmulator; Workstation 2.1 od Parallels). Počítače Apple Macintosh umožňují nativní provoz Linuxu a díky aktuálně probíhajícímu přechodu na procesory Intel a technologii Boot Camp od Apple umožňují rovněž nativní provoz Windows XP. Lze také očekávat spouštění PC aplikací pod MacOS X (již existuje první kompilace WINE pro Intelovské Macy). Mac OS X 10.4.4 lze nativně provozovat - na počítačích Apple Macintosh - jak na PowerPC (IBM, Motorola) tak na x86 procesorech (Intel). Z počítačů Apple Macintosh se tak postupně stává nejuniverzálnější platforma pro běžného i profesionálního uživatele.
Po ukončení distribuce Mac OS 9 byl Mac OS X dosud vydán v následujících verzích:

- Mac OS X 10.2 Jaguar (uveden v roce 2002)
- Mac OS X 10.3 Panther (uveden v roce 2003); označovaný běžnými uživateli za první plně použitelný Mac OS X
- Mac OS X 10.4 Tiger (uveden v roce 2005); tento OS přinesl řadu vylepšení, mimo jiné vyřešení některých problémů českých specifik (např. přepínání klávesnice…).
- Mac OS, resp. MacOS X je možné spustit pouze na HW Apple, což znamená dokonalé provázání SW a HW, neexistuje zde problém s nekompatibilitou HW a SW. Apple zatím popírá záměr umožnit provoz Mac OS X na běžných PC (nicméně přechodem na procesory Intel již tomu nebrání žádná technická omezení a zůstávají pouze omezení definovaná samotným Applem a BIOSem na PC).

Základní myšlenkou systému je jednoduchost a intuitivnost uživatelského rozhraní - ergonomie Mac OS X je ve srovnání s jinými OS na vysoké úrovni; výrazně jednodušší je mimo jiné i customizace OS, konfigurace např. internetu a sítí obecně (většinu úkonů je schopen provádět uživatel sám / není třeba zásahu správce sítě).

Jednou z největších výhod tohoto OS je rovněž prakticky nulové množství virů a spyware (Mac OS X je vůči virům z PC zcela imunní; specielní viry pro Mac OS X prakticky neexistují, mimo jiné i zásluhou nezajímavostí platformy pro strůjce virů; jedinou známou výjimkou mohou být pouze macroviry v MS Office). Reakce na bezpečnostní rizika v systému je velice rychlá – ve formě updatů systému. Tento systém je tedy v současnosti velmi bezpečný – zejména díky svému UNIX jádru. Podobně jako ostatní moderní systémy umožňuje několik stupňů zabezpečení dat.

Nevýhodou je menší podpora ze strany výrobců a prodejců HW, nicméně největší výrobci periferií tuto platformu standardně podporují (HP, Epson…) a podpora se od doby uvedení Mac OS X rozšiřuje.

2.3.4 Operační systémy Linux

Situace okolo PC byla v 80. a 90. letech pro mnoho uživatelů natolik neuspokojivá, že se některé firmy (a stejně tak vývojářská komunita) pustily do vývoje vlastních OS

nebo alespoň GUI pro DOS, které by odstranilo alespoň ty největší nedostatky Dosu a Windows - uživatelskou nepřívětivost, množství chyb, složitá síťová řešení, nepřítomnost multitaskingu a chybějící podpora multimédií. Vznikaly různá řešení - řešení založená na MS-DOS, řešení nová (např. OS/2) a řešení na bázi UNIXu (X Window System; GNU/Hurd;...). Jak se však později ukázalo i po téměř 30 letech vývoje osobních počítačů je dodnes stále jedním z nejlepších systém UNIX, resp. řešení na něm založená, tzv. systémy typu UNIX. Nejnadějnějším systémem typu UNIX pro PC se stal Linux, výtvor finského studenta Linuse Torvaldse, datovaný na 25.8.1991, kdy Torvalds zaslal do diskuzní skupiny comp.os.minix příspěvek s předmětem "What would you like to see most in minix?" o tom, že vyrábí free operační systém.

Torvaldsův Linux v poslední době pomalu dospěl do fáze komerční použitelnosti, zatím především jako síťový server, ale pomalu také jako systém pro běžného uživatele. Již v roce 2000 začínají hlavní komerční prodejci PC HW (Compaq, IBM, Dell, SGI, Fujitsu) prodávat desktop a laptop počítače s předinstalovaným Linuxem. Linux se postupně stává úspěšným konkurentem jak na serverech, kde úspěšně nahrazuje starší a hlavně drahé unixy, tak na desktopových stanicích, kde začíná úspěšně konkurovat MS Windows.

Zatím ho však částečně deklasuje, že jeho vývojářská základna je naprosto decentralizovaná. Fakt, že je Linux vyvíjen nepřehlédnutelným zástupem programátorů ze všech koutů planety, sice umožňuje, že veškerá vylepšení jsou rychle hotova a k dispozici, ale na druhé straně způsobuje nepřítomnost nějaké přesnější vývojové linie. Linux tedy trpí nemocí, známou z Unixu - každý si ho upravuje pro sebe, takže vzniká přehršel verzí až zmatek. Toto si uvědomují dokonce i sami tvůrci Linuxu, takže se objevily snahy o jeho standardizaci - zde existují dva koncepty: UnitedLinux a Linux Standard Base (LSB). Mimo OpenSource řešení existují také komerční distribuce Linuxu, které se vyznačují vyšší podporou ze strany výrobce.

Dnes tedy existuje celá řada na kancelářských stanicích a serverech využitelných distribucí OS Linuxu, jako např.:

- **SuSE Linux.** Distributor Linuxu s oficiální pobočkou v České republice. SuSE Linux se dodává v několika jazykových verzích včetně české. Distribuce je placená a zahrnuje rozsáhlou českou dokumentaci a instalační podporu. Tuto distribuci převzala a dále vyvíjí fa Novell.
- **RedHat Linux (a CZ verze).** Začátkem roku 2001 nejrozšířenější distribuce. Uživatelsky příjemná instalační procedura, program na konfiguraci systému X-Window, prostředky pro administraci systému přes X11, velké množství softwarových balíků.
- **Debian GNU/Linux.** Tato distribuce není vyvíjena jednou firmou - na vývoji jednotlivých balíků této distribuce se podílejí lidé po celém světě. Jde o jednu z mála nekomerčních distribucí Linuxu.
- **Ubuntu Linux.** Kvalitní distribuce, kde Ubuntu tým vydává novou verzi Ubuntu každých šest měsíců. Vždy obsahuje poslední verze jádra, X-Window systém, Gnome a dalších klíčových aplikací, a každá verze má bezpečnostní podporu 18 měsíců. Tato distribuce vychází z distribuce Debian, je však více zaměřena na běžné uživatele, přecházející také z jiných OS.
- **Další distribuce** (Fedora; Linspire; Slackware Linux; Turbo Linux; Linux Mandrake; Caldera Network Desktop, OpenLinux).

Jednou z největších výhod mnoha distribucí Linuxu je jeho nulová pořizovací cena, stejně jako velké množství aplikací, které jsou pro tyto OS v rámci opensource a GNU licencí vyvíjeny.

Linuxové distribuce nedosahují z hlediska uživatelského rozhraní takového komfortu, jako Mac OS X, či Windows XP, nicméně při řádném zaškolení, případně nenáročného způsobu použití (např. na obsluhu konkrétního procesu u stolních počítačů) je pro kancelářské nasazení dobře použitelný, tento stav se ovšem velmi rychle mění směrem k většímu uživatelskému komfortu.
Velkou výhodou tohoto systému je možnost takřka libovolného přizpůsobení potřebám uživatele. Toto přizpůsobení si ovšem není schopen uživatel provádět sám. Pro potřeby podpory je možné využití komerčních služeb mnoha firem.

Nevýhodou tohoto OS je nižší podpora komerčních aplikací (např. absence podpory Microsoft Office), tato nevýhoda je ovšem vyvažována velkým množstvím alternativního SW zdarma, který je často technologicky vyspělejší a založen na uznávaných standardech. Další nevýhodou je také nižší podpora ze strany výrobců periferií, obvykle však existují „neautorizované“ ovladače také pro HW jejichž výrobci Linux přímo nepodporují.

Kontrolní otázky:

4. Vyjmenujte alespoň dva důvodu, proč znalost operačních systémů je potřebná pro práci programátora?
5. Proč je operační systém rozhraním mezi uživatelem a hardware počítače?
6. Jaké jsou funkce BIOSu?
7. Ze kterých generických komponent se skládá operační systém?

Úkoly k zamyšlení:

3. Zamyslete se nad výhodami a nevýhodami grafického (Windows) a textového (MS DOS) rozhraní mezi uživatelem a operačním systémem?

Korespondenční úkol:

2. Představte si rozhraní člověka s počítačem řízeným hlasem. Napište některé příkazy, které by musel operační systém interpretovat, aby mohl vykonávat nejzákladnější funkce.

Shrnutí obsahu kapitoly

V této kapitole jste se seznámili s důvody obecných znalostí principů operačních systémů. Důraz v této kapitole byl kladen na pochopení rozhraní člověk/stroj a proces/operační systém. Velká pozornost byla věnována vysvětlení zájmů operačních systémů.

3 Architektura operačních systémů

V této kapitole se dozvíte:

- Z jakých generických komponent se skládají operační systémy?
- Jaké jsou hlavní funkce jednotlivých komponent operačních systémů?
- Jaké jsou základní algoritmy činnosti jednotlivých správců operačního systému?
- Jak jsou jednotlivé komponenty operačních systémů vzájemně propojeny?

Po jejím prostudování byste měli být schopni:

- Charakterizovat funkce jednotlivých částí operačních systémů.
- Znát základní algoritmy činnosti jednotlivých správců operačního systému.
- Porozumět způsobům komunikace jednotlivých částí operačního systému.
- Popsat činnost operačního systému z hlediska funkcí jednotlivých částí operačního systému.

Klíčová slova této kapitoly:

Správa procesoru, správa procesů, správa paměti, správa I/O systému, správa sekundární paměti, správa souborů, networking, interpret příkazů, systém ochran, multithreading, stavový model, plánovač, preemptivní plánování, nepreemptivní plánování, kooperativní multitasking, ochrana paměti, stránkování, segmentace, přerušení, DMA, sběrnice.

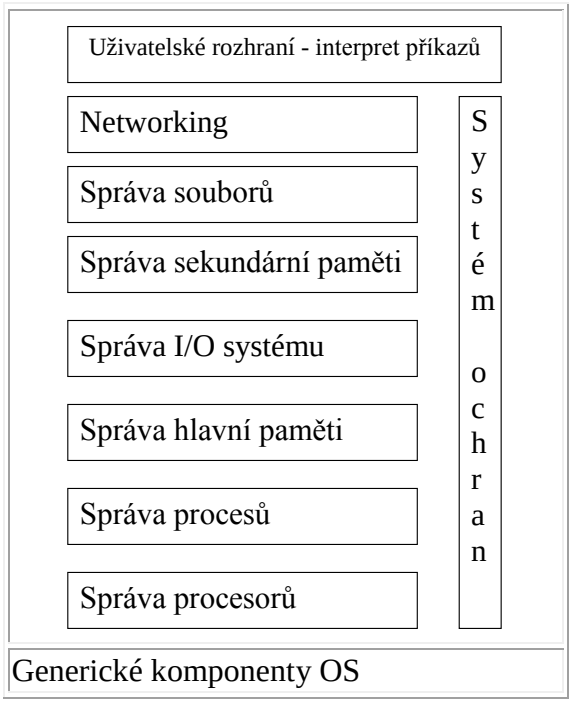
Doba potřebná ke studiu: 8 hodin

Průvodce studiem

Tato kapitola je nejnáročnějším tématem studijního textu a jsou v ní popsány principy činnosti operačních systémů. Poměrně náročné téma je zejména pro ty z Vás, kteří dosud nemají žádné znalosti z oblasti architektury operačních systémů. V takovém případě Vám zřejmě některé principy funkcí operačních systémů budou připadat obtížně pochopitelné, ovšem nenechte se tím odradit, neboť pochopením této části jste pochopili, jak operační systémy pracují.
Na studium této části si vyhraďte alespoň 8 hodin. Doporučujeme studovat s přestávkami vždy po pochopení jednotlivých podkapitol. Po celkovém prostudování a vyřešení všech příkladů doporučujeme dát si pauzu, třeba 1 den, a pak se pusťte do vypracování korespondenčních úkolů.

Pro popis obecné architektury operačních systémů použijme model vrstvené architektury uvedené v předchozím odstavci. Při studiu architektury operačních systémů vycházíme z principu, že operační systém je „správce prostředků“, je to soubor programů (algoritmů) vytvořených k ovládání systémových prostředků tj. paměti, procesorů, periferních zařízení a souborů informací (tj. programů a dat). Funkcí operačního systému je dbát, aby tyto prostředky byly efektivně využívány, řešit konflikty vzniklé při „soutěžení“ o jednotlivé prostředky mezi různými uživateli (mezi jejich programy). Operační systém musí sledovat stav každého prostředku, rozhodovat, kterému procesu bude prostředek přidělen (v jakém rozsahu a na jak dlouho), prostředek přidělit a případně žádat jeho navracení. Podle toho, jak dělíme

prostředky, se dělí i jejich správci. Na následujícím obrázku je naznačeno rozdělení operačního systému na jednotlivé správce.



Obecná architektura operačních systémů je budovaná na principech hierarchických vrstev. Operační systém se dělí do jistého počtu vrstev (úrovní). Každá vrstva je budována na funkcionalitě nižších vrstev tzn. že nejnižší vrstva je tvořena hardwarem počítače a nejvyšší vrstva je vrstva uživatelského rozhraní. Tím se řeší problém přílišné složitosti velkého systému. Nižší vrstva nabízí vyšší vrstvě „primitivní“ funkce (služby) a přitom nižší vrstva nemůže požadovat provedení služeb vyšší vrstvy. Používají se přesně definovaná rozhraní umožňující v rámci jedné vrstvy jejich vlastnosti a funkčnosti modifikovat, aniž to ovlivní ostatní vrstvy.
V klasickém operačním systému (z pohledu historického) jsou procesy jen uživatelské programy a vlastní operační systém je prováděn jako samostatná entita v privilegovaném režimu. K přepínání kontextu procesů dochází jen tehdy, je-li to nutné z hlediska plánování.

V procesově konstruovaném operačním systému je vlastní operační systém kolekcí systémových procesů. Funkcí jádra je procesy separovat a přitom jim umožnit kooperovat. Minimum funkcí je třeba realizovat v privilegovaném režimu, kdy jádro je pouze ústředna pro přepojování zpráv.
Samostatnou variantou operačního systému je realizace architekturou tzv. mikrojádra, kde malé jádro plní pouze několik málo nezbytných funkcí, jako je primitivní správa paměti (adresového prostoru), komunikaci mezi procesy a základní plánování a správu I/O zařízení a přerušení. Ostatní služby jádra řeší procesy (servery) běžící v uživatelském režimu. Jsou to ovladače, služby systému souborů a virtualizace paměti. Výhody v architektuře mikrojádra jsou v pružnějším a snadněji rozšiřitelném řešení, lze doplňovat nové služby, odstraňovat nepotřebné služby a všechny služby jsou poskytovány jednotně (výměnou zpráv). Toto řešení je jednoduše přenositelné, tj. při implementaci na nový procesor stačí změnit mikrojádro. Spolehlivějším řešením jsou

různá modulární řešení. Moduly jsou snadněji testovatelné a umožňují podporu distribuovanosti tj. výměna zpráv je implementovatelná v síti i v jednom systému.

V další části si stručně popíšeme hlavní funkce jednotlivých komponent operačních systémů.

3.1 Správa procesorů/procesů

Správce procesoru má tyto funkce:
- sleduje prostředek (procesor a stav procesů),
- rozhoduje, komu bude dána možnost užít procesor,
- přiděluje procesu prostředek, tj. procesor,
- požaduje vrácení prostředku (procesoru).

Pod pojmem proces (task) chápejme provedení nějakého programu. Proces potřebuje pro svoji realizaci jisté zdroje:
- doba procesoru,
- paměť,
- I/O zařízení, atd.

Operační systém je z hlediska správy procesů zodpovědný za:
- vytváření a rušení procesů,
- potlačení a obnovení procesů,
- poskytnutí mechanismů pro synchronizaci procesů a pro komunikaci mezi procesy.

Operační systém je z hlediska správy procesorů zodpovědný za výběr procesu běžícího na volném procesoru.

3.2 Správa (hlavní, operační) paměti

Správce paměti je úložiště připravených tj. rychle dostupných dat sdílených procesorem a vstupními/výstupními zařízeními. Hlavní (operační, primární) paměť je pole samostatně adresovatelných slov nebo bytů, zpravidla energeticky závislá, tj. po výpadku napájení se data z ní ztrácí. Operační systém je z hlediska správy (hlavní) paměti odpovědný za:
- vedení přehledu kdo a kterou část paměti v daném okamžiku využívá,
- rozhodování kterému procesu uspokojit jeho požadavek na prostor paměti po uvolnění,
- přidělování a uvolňování paměti podle potřeby,
- řízení virtuální paměti.

Z hlediska těchto zodpovědností správce operační paměti:
- udržuje přehled o přidělené a volné paměti,
- ve spolupráci se správou procesů rozhoduje o tom, kterému procesu, kolik, kde a kdy má přidělit operační paměť,
- provádí přidělení volné části paměti,
- určuje strategii odnímání dříve přidělené operační paměti procesům (opět po předchozí domluvě se správou procesů).

V této souvislosti několik poznámek k řízení tzv. virtuální paměti. K základním problémům návrhu architektury počítače patří rozhodnutí, jak se bude zobrazovat tzv. logický adresový prostor – LAP, do tzv. fyzického adresového prostoru – FAP. Uživatel vidí logický adresní prostor – LAP. Programy a data z LAP jsou do fyzického adresního prostoru - FAP zaváděny podle potřeby. LAP je vymezen množinou adres určenou počtem bitů vnitřní adresní sběrnice. Obvykle je jednodimenzionální. S dvoudimenzionální strukturalizací LAP se setkáme u kolekce samostatných lineárních segmentů (proměnné délky). V takovém případě bude adresa paměťového místa tvořena dvěma složkami. Fyzický adresový prostor FAP je vymezen množinou adres určenou počtem bitů vnější adresní sběrnice počítače. FAP je určen velikostí operační paměti. Zobrazení LAP do dostupného FAP se provádí pomocí hardware (Dynamic Address Tranlation – DAT, Memory Management Unit - MMU). Tyto prostředky musí řešit různé konflikty. Např. při odkázání místa s adresou LAP, které není zobrazeno ve FAP se ve FAP nalezne (vytvoří) volný blok a na toto místo se zavede blok z obrazu LAP s požadovanou informací.

3.3 Správa I/O systému

Správce periferních zařízení (vstupního/výstupního systému) má tyto funkce:

- sleduje stav prostředků (periferních zařízení, jejich řídících jednotek),
- rozhoduje o efektivním způsobu přidělování prostředku – periferního zařízení,
- přiřazuje prostředek (periferní zařízení) a zahajuje I/O operaci,
- požaduje navracení prostředku.

Z hlediska funkce operačního systému lze správce I/O systému chápat jako:

- úložiště vyrovnávacích pamětí,
- univerzální rozhraní ovladače I/O zařízení,
- ovladače jednotlivých hardwarových I/O zařízení.

Do správy I/O systému patří i správa vnější (sekundární) paměti. Počítačový systém musí poskytnout pro zálohování hlavní paměti sekundární paměť, v poslední době nejvíce používané pevné disky (hard disky) s perspektivou přechodu na velkokapacitní paměti typu „flash“. Operační systém je z hlediska správy vnější (sekundární) paměti odpovědný za:

- správu volné paměti,
- přidělování paměti,
- plánování činnosti disku.

3.4 Správa souborů

Správce souborů má tyto funkce:

- sleduje prostředek (soubor), jeho umístění, užití, stav atd.,
- rozhoduje, komu budou prostředky přiděleny, realizuje požadavky na ochranu informací uložených v souborech a realizuje operace přístupu k souborům,
- přiděluje prostředek, tj. otevírá soubor,
- uvolňuje prostředek, tj. uzavírá soubor.

Pod pojmem soubor chápeme jak programy, tak data. Operační systém je z hlediska správy souborů odpovědný za:

- vytváření a rušení souborů,

- vytváření a rušení adresářů (katalogů, složek),
- podporu primitivních operací pro manipulaci se soubory a s adresáři,
- zobrazení souborů do sekundární paměti,
- archivování souborů na energeticky nezávislá média.

3.5 Networking, distribuované systémy

Pod pojmem distribuovaný systém chápeme kolekci procesorů, které nesdílejí ani fyzickou paměť ani hodiny, synchronizující činnost procesoru. Každý procesor má svoji lokální paměť a lokální hodiny. Procesory distribuovaného systému jsou propojeny komunikační sítí. Komunikace jsou řízeny protokoly. Distribuovaný systém uživateli zprostředkovává přístup k různým zdrojům systému.

3.6 Systém ochran

Pod pojmem systém ochran rozumíme mechanismy pro řízení přístupu k systémovým a uživatelským zdrojům. Systém ochran musí:

- rozlišovat mezi autorizovaným a neautorizovaným použitím,
- specifikovat problém vnucovaného řízení,
- poskytnou prostředky pro své prosazení.

3.7 Uživatelské rozhraní - interpret příkazů

Interpret příkazů je program, umožňující vykonávat příkazy pro:

- správu a vytváření procesů - služby operačního systému poskytované interpretem příkazů slouží k provedení programu, tj. k schopnosti operačního systému zavést program do hlavní paměti a spustit jeho běh,
- ovládání I/O zařízení - uživatelský program nesmí provádět I/O operace přímo, operační systém musí poskytovat prostředky k provádění I/O operací,
- správu sekundární paměti - manipulace se systémem souborů, schopnost číst, zapisovat, vytvářet a rušit soubory,
- správu hlavní paměti,
- zpřístupňování souborů,
- ochranu – tj. detekci chyb v procesoru a paměti, I/O zařízeních a v programech uživatelů pro zajištění správnosti výpočtu,
- práci v síti - výměna informací mezi procesy realizovaná buďto v rámci jednoho počítače nebo mezi různými počítači pomocí sítě, tj. implementace sdílenou pamětí nebo předáváním zpráv.

Tento program se nazývá příkazový interpret. Jeho funkcí je získávat a provádět příští příkaz.

Uživatelská rozhraní jsou realizovaná znakově (někdy označované řádkově) nebo graficky. Znakově orientovaným interpretům zadáváme příkazy pomocí klíčových slov, graficky orientovaným pomocí poklepání myši nebo dotykem na dotykové obrazovce na ikonu, pomocí dialogů apod.

3.8 Vnitřní služby operačního systému

Vnitřní služby operačního systému nejsou určeny k tomu, aby pomáhaly uživateli, v prvé řadě slouží pro zabezpečení efektivního provozu systému, tj. slouží pro:

- Přidělování prostředků (zdrojů) mezi více souběžně operujících uživatelů nebo úloh.
- Účtování a udržování přehledu o tom, kolik jakých zdrojů systému který uživatel používá. Cílem je účtování za služby a sběr statistik pro plánování.
- Ochranu tj. péči o to, aby veškerý přístup k systémovým zdrojům byl pod kontrolou.

Vnitřní služby operačního systému jsou obecně realizovány souborem systémových programů vytvářejících určité systémové struktury tzv. virtuální stroje. Typickými službami jsou programy pro:

- práci se soubory, editaci souborů, katalogizaci souborů, modifikaci souborů,
- získávání, definování a údržbu systémových informací,
- podporu jazykových prostředí,
- zavádění a provádění programů,
- komunikace a řízení aplikačních programů.

Kontrolní otázky:

1. Ze jakých generických komponent se skládají operační systémy?
2. Jaké jsou hlavní funkce jednotlivých komponent operačních systémů?
3. Jaké jsou základní algoritmy činnosti jednotlivých správců operačního systému?
4. Jak jsou jednotlivé komponenty operačních systémů vzájemně propojeny?
5. Které zdroje potřebuje operační systém pro svoji činnost?

Úkoly k zamyšlení:

1. Rozeznáváme tři druhy plánování procesů: krátkodobé, střednědobé a dlouhodobé. Pokuste se rozhodnout ve které úrovni 5-stavového diagramu jednotlivé plánovače pracují.

Korespondenční úkol:

Prostudujte příručku nějakého operačního systému a odpovězte na tyto otázky:

a. Za jakých základních komponent se operační systém skládá?
b. Jakými metodami obsluhuje vybraný OS periferní zařízení?
c. Jaký systém souborů využívá a jaké atributy souborů uchovává?
d. V jakém jazyku je OS napsán?

Shrnutí obsahu kapitoly

V této kapitole jste se seznámili s jednotlivými generickými komponentami operačních systémů, s jejich hlavními funkcemi a základními algoritmy činnosti jednotlivých správců operačních systémů. Důraz byl kladen na pochopení vzájemné komunikace jednotlivých částí operačních systémů. Velká pozornost byla věnována komplexnímu pochopení činnosti operačních systémů.

4 Správa procesů

V této kapitole se dozvíte:

- Jak jsou plánovány procesy v operačním systému?
- Jakými mechanismy spolupracují mezi sebou procesy?
- Jak se řeší problém uváznutí v operačních systémech?
- Co je to multithreading?

Po jejím prostudování byste měli být schopni:

- Charakterizovat druhy operačních systémů z hlediska plánování procesů.
- Znát způsob komunikace procesů v operačních systémech.
- Porozumět základním algoritmům synchronizace procesů.
- Popsat základní plánovací algoritmy přidělování procesoru procesům..

Klíčová slova této kapitoly:

Proces, procesor, stavový model, plánovač, kooperativní multitasking, preemptivní a nepreemptivní plánování procesů, kontext, souběh, uváznutí, multithreading.

Doba potřebná ke studiu: 6 hodin

Průvodce studiem

Studium této kapitoly je poměrně náročné zejména pro ty z Vás, kteří dosud nemají žádné znalosti z oblasti algoritmů paralelních procesů. V takovém případě Vám zřejmě některé algoritmy budou připadat obtížně pochopitelné, ovšem nenechte se tím odradit, neboť pochopením této části se Vám usnadní studium následujících kapitol.
Na studium této části si vyhraďte alespoň 6 hodin. Doporučujeme studovat s přestávkami vždy po pochopení jednotlivých podkapitol. Po celkovém prostudování a vyřešení všech příkladů doporučujeme dát si pauzu, třeba 1 den, a pak se pusťte do vypracování korespondenčních úkolů.

Správa procesů a s tím související správa procesorů patří mezi nejkomplikovanější části operačního systému. Pro pochopení funkcí správce si nejdříve vysvětlíme některé pojmy.

4.1 Základní pojmy

Pod pojmem program budeme chápat zápis algoritmu v nějakém programovacím jazyce (například ve strojovém kódu). Předpokládejme, že je statický, neměnný (neuvažujeme-li vývoj nových verzí programů).

Proces (úloha, aplikace) je pak běžící program, tvořený neměnným kódem, konstantami a proměnnými daty, jako jsou stav procesoru, data na zásobníku, globální proměnné, halda, soubory atd.

Pro běh procesu jsou nutné následující zdroje systému:

- procesor,
- vnitřní paměť,
- další prostředky (vstupní/výstupní zařízení, soubory apod.).

Základní členění operačních systémů z hlediska počtu pracujících uživatelů a počtu paralelně pracujících úloh (procesů) je zřejmé z následující tabulky.

Systém	Jednouživatelský	Víceuživatelský
Jednoúlohový	MS-DOS, CP/M	(stanice v Novellu), Intellec SIV
Víceúlohový	Windows, Mac OS X)	Windows Server, Unix, VM/S

Klasifikace operačních systémů z hlediska stupně paralelnosti práce:

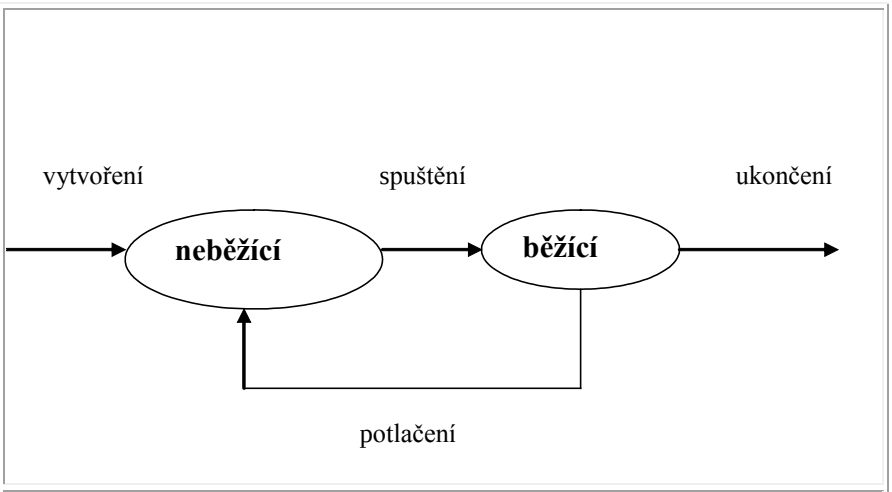
- **Jednouživatelské jednoúlohové** - s podporou operačního systému se zpracovává pouze jeden proces a to trvale.
- **Jednouživatelské víceúlohové** - jeden uživatel má současně spuštěno více aplikací (např. na pozadí probíhá náročný výpočet, hraje hudba, probíhá komunikace po internetu a současně s tím uživatel edituje dokument).
- **Víceuživatelské víceúlohové** - více uživatelů sdílí tytéž prostředky. Někdy se označují jako operační systémy se sdílením času.
- **Systémy s reálným časem** - je to vlastně varianta předchozích dvou typů určená pro řízení technologických procesů.

Víceúlohový systém zpravidla vytváří iluzi, že jednotlivé procesy mají celý systém pro sebe; odizolovává procesy od sebe navzájem. Na druhou stranu je někdy potřeba, aby spolu mohly procesy spolupracovat. Klasické procesy mají oddělené adresní prostory. Pokud spolu chtějí komunikovat, musí použít prostředky poskytované operačním systémem.

Multitasking může usnadnit programování - příkladem jsou například síťové servery, které obsluhují několik klientů současně. Při klasickém naprogramování pomocí jednoho procesu by tento proces byl tvořen velkou smyčkou, ve které by se přijímaly požadavky klientů a postupně se vyřizovaly. Umožňuje-li systém vytváření podřízených procesů (child process, potomek), může server fungovat tak, že při příchodu požadavku od klienta server odštěpí (fork) podproces. Původní proces bude nadále čekat na další požadavky klientů. Potomek obslouží klienta a skončí.
Některé systémy podporující tzv. vícevláknový řežim (multithreading) umožňují, aby se jeden těžký proces skládal z více vláken. Vlákna (thread, lightweight process) jednoho procesu sdílejí adresní prostor paměti a mohou spolu komunikovat pomocí sdílené paměti. Nepodporuje-li systém multithreading, znamená to, že každý proces je tvořen právě jedním vláknem. Výhodou vláken je nižší režie při jejich přepínání a snadnější spolupráce mezi vlákny než mezi procesy. Každé vlákno má samostatný zásobník a udržuje se pro něj stav procesoru (včetně programového čítače).

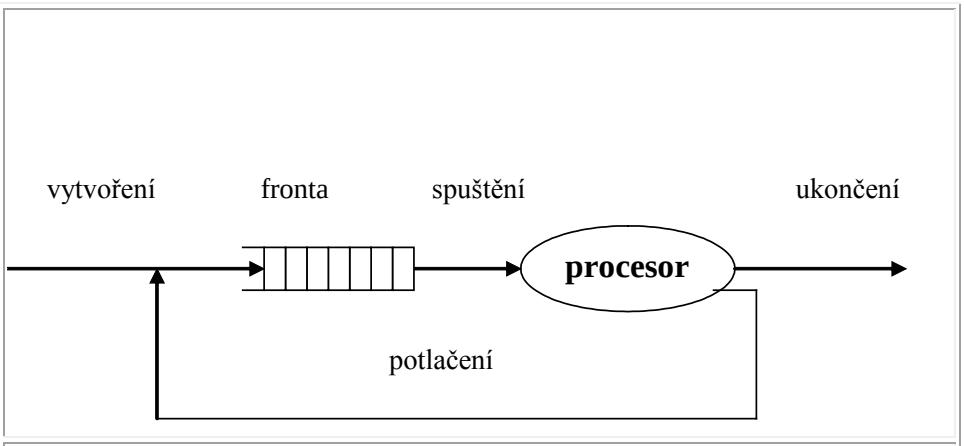
4.2 Stavový model procesů

Životní cyklus procesu v operačním systému lze charakterizovat pomocí přechodů mezi stavy procesu. Grafickým vyjádřením těchto přechodů mezi stavy jsou stavové modely. Základním vyjádřením cyklu procesu je dvoustavový model, zobrazený na následujícím obrázku.



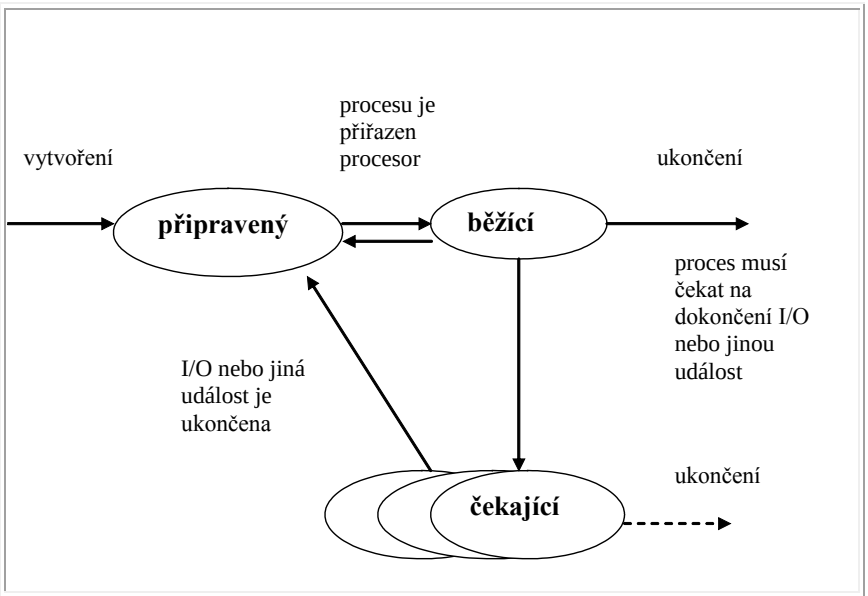
2 – stavový model procesů

Proces je vytvořen buď příkazem uživatele (u terminálu) nebo na žádost operačního systému o provedení služby či na žádost jiného procesu (rodiče). Takto vytvořený proces je ve stavu „neběžícím“. Spuštěním procesu, na základě plánovacího algoritmu přechází proces do stavu „běžící“. Tento proces může být ukončen normálně, tj. byl celý proveden, nebo násilně vypršením časového limitu či uživatelem, provedením chybné instrukce, chybou I/O zařízení, porušením ochrany paměti, nebo na žádost rodiče apod. „Běžící“ proces může být potlačen na základě časového limitu, vyšší prioritou apod. a přechází do stavu „neběžící“. Z hlediska implementace je zřejmé, že ve stavu „běžící“ může být jen jeden proces realizován jedním procesorem, kdežto ve stavu „neběžícím“ může být více procesů zařazených do fronty, jak je naznačeno na následujícím obrázku.



2 stavový model s frontou připravených procesů

Z tohoto pohledu je přesnější vyjádření stavového modelu pomocí tří stavů, jak je zřejmé z následujícího obrázku.



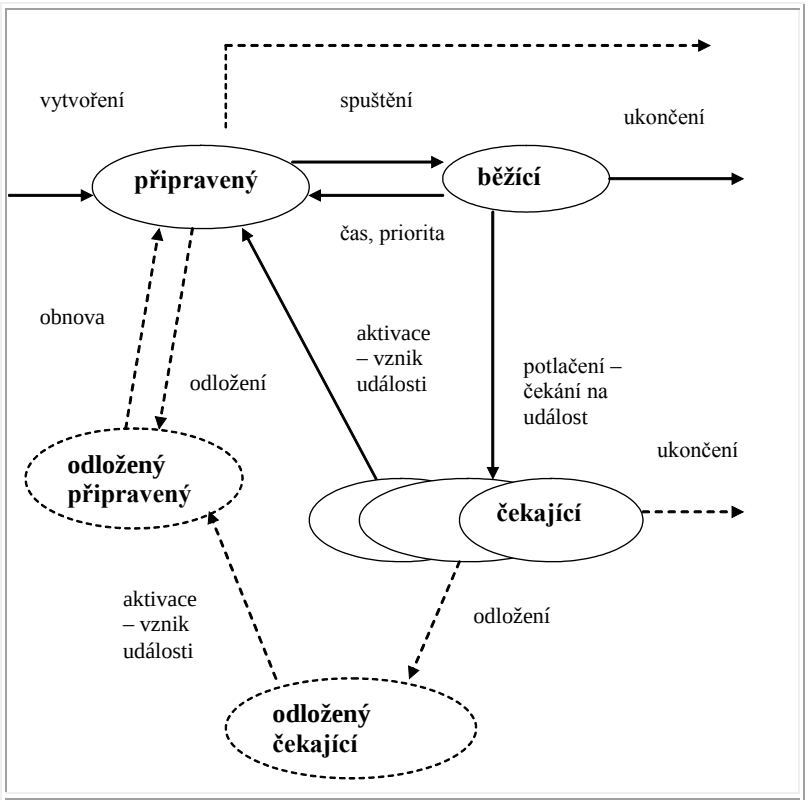
3 – stavový model

Ve stavu „běžící“ (running) je procesu přidělen procesor a právě se provádí příslušné programy. Stav „čekající“ (waiting) vyjadřuje, že proces čeká na určitou událost, např. dokončení vstupní/výstupní (I/O – Input/Output) operace. Stav „připravený“ (ready) charakterizuje proces připravený k vykonání a čeká pouze na přidělení procesoru.

Popišme si nyní průchod stavovým modelem úlohy, využívající I/O operaci. Je-li úloha ve stavu „běžící“ a žádá o čtení ze souboru (nebo jinou I/O operaci), modul správy souborů volá modul přidělování periferií, aby zahájil čtení (nebo jinou I/O operaci). Modul přidělování periferií ji zahájí a zároveň požádá plánovač procesu, aby proces převedl do stavu „čekající“. Je-li I/O operace dokončena, je patřičný signál přerušení vyhodnocen jako žádost o navrácení úlohy do stavu připraven. Pokud je úloha ve stavu „běžící“ dokončena, modul přidělování periferií ji odebere přidělené periferie, modul přidělování paměti uvolní paměť, která byla úloze alokována a plánovač procesu jí odebere procesor. Tím je výpočet úlohy ukončen.
Pro definici a správu procesů využívá operační systém tabulky, obsahující potřebné informace o stavu procesu. Každý běžící proces je realizací programu, který je interpretací instrukcí uložených v paměti. Adresa právě prováděné instrukce je zapsána v čítači instrukcí a v návaznosti na předchozí instrukce je mnohdy modifikována obsahem registrů procesoru. V nejjednodušším případě pak stačí o každém procesu uchovat právě čítač instrukcí a registry procesoru.
Pro plánování procesů je třeba uchovávat informace potřebné pro spuštění procesu na procesoru, informace potřebné pro správu paměti a informace potřebné pro správu I/O. Informace se vkládají do front pro plánování procesů. Fronta úloh je množina všech procesů v systému. Těchto front může být více:

- fronta připravených procesů (ready), což je množina procesů sídlících v hlavní paměti a připravených k běhu,
- fronta na zařízení což je množina procesů čekajících I/O zařízení,
- fronta odložených procesů charakterizovaná množinou procesů čekajících na přidělení místa v hlavní paměti,
- fronta na semafor realizovaná množinou procesů čekajících synchronizační událost.

V souvislosti s plánováním procesů se setkáváme ještě s jedním modelem umožňujícím odkládání procesů (swapping). Každý proces, aby byl procesem, se jednou musí dostat do operační paměti (alespoň částečně). S omezenou kapacitou operační paměti musí existovat nástroj který umožňuje umístit do paměti více procesů. Jedním nástrojem je virtuální paměť, tj. rozšíření fyzického adresního prostoru na logický. Příliš mnoho procesů v operační paměti však snižuje výkonnost. Proto operační systémy umožňují provádění některých procesů odložit mimo adresní prostor operační paměti např. na disk. Stavový model se pak rozšíří o další dva stavy – „odložený čekající“, „odložený připravený“ – viz následující obrázek.



5-stavový model procesů

Vyjděme z předpokladu, že všechny procesy jsou „čekající“ a operační systém vytváří prostor pro přidělení „běžícímu“ procesu. „Odložené“ procesy uvolňují operační paměť v případě, že je mnoho „čekajících“ procesů nebo vlastník procesu si to přeje nebo to předpisuje časový plán, či si to přeje rodič z důvodu synchronizace sourozenců. Pak procesy přecházejí do stavu „odložený čekající“. V případě, že se stala očekávaná událost (např. stavovou informaci má operační systém přístupnou) přechází proces ze stavu „odložený čekající“ do stavu „odložený připravený“. V případě, že se fronta připravených vyprázdnila (nebo alespoň téměř vyprázdnila), pak přechází proces ze stavu „odložený připravený“ do stavu „připravený“.

Plánovač procesoru je program, který vybírá z procesů, sídlících v hlavní paměti a těmi, které jsou připravené k běhu – ve stavu „připravený“. Tento plánovač může vydat plánovací rozhodnutí v okamžiku, kdy proces:

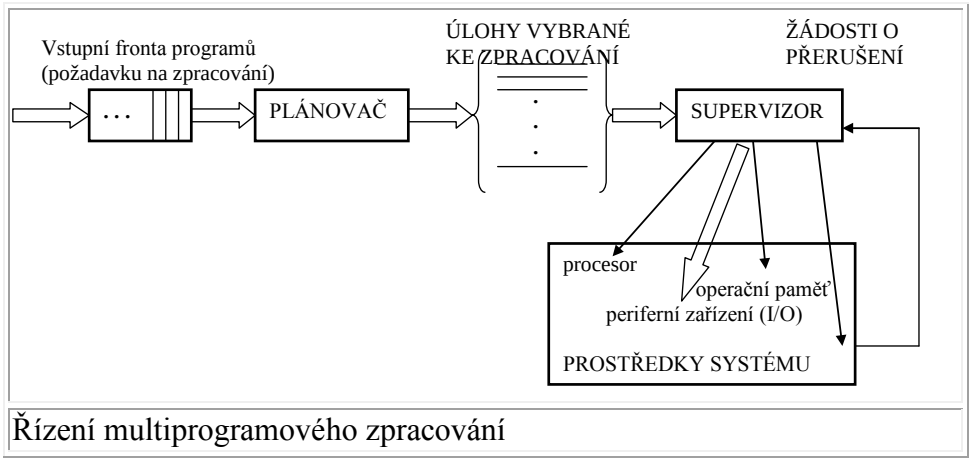
1. Přechází ze stavu „běžící“ do stavu „čekající“.
2. Přechází ze stavu „běžící“ do stavu „připravený“.
3. Přechází ze stavu „čekající“ do stavu „připravený“.
4. Končí.

Případy 1 a 4 se označují jako „nepreemptivní“ plánování (plánování bez předbíhání).
Případy 2 a 3 se označují jako „preemptivní“ plánování (plánování s předbíháním).
Ve víceúlohových systémech při plánování procesoru mohou nastat následující situace:

- Pokud některý proces přejde ze stavu „běžící“ do stavu „čekající“ (čekání na I/O operaci, semafor, čekání na uplynutí zadaného časového intervalu, čekání na ukončení procesu-potomka), pak hovoříme o nepreemptivním plánování CPU (procesů).
- Pokud některý proces skončí, pak hovoříme taktéž o nepreemptivní plánování CPU (procesů).
- Pokud je některý proces převeden ze stavu „běžící“ do stavu „připravený“, pak hovoříme o preemptivní plánování CPU.
- Pokud některý proces přejde ze stavu „čekající“ do stavu „připravený“, pak hovoříme o systémech reálného času.

4.3 Plánování procesů

Operační systém koordinuje činnost všech prostředků počítačového systému při současné práci na několika procesech (programech). Tato činnost je naznačena na následujícím obrázku.



Řízení multiprogramového zpracování

Supervizor (správce prostředků) udržuje přehled o stavu všech systémových prostředků (procesoru, operační paměti, vnějších pamětí, vstupních a výstupních zařízení), tj. zda jsou připojena nebo odpojena, mají-li poruchu, jsou-li volné nebo obsazené a řídí jejích činnost. U paměti vede tabulku volných a obsazených oblastí. Volné zařízení a oblasti pamětí přiděluje programům vybraným ke zpracování. Po dokončeném zpracování, o němž dostává informaci signálem přerušení, zajišťuje obsluhu požadavku na přerušení s příp. uvolněním systémového (-vých) prostředku(ů) a jeho (jejich) přidělení jinému procesu (programu), čekajícímu na obsluhu. Kromě toho obsluhuje/řeší i mimořádné stavy v systému.

Úkolem plánovače je vybírat vstupní požadavky (programy) ke zpracování na základě zjištěných pro ně potřebných prostředků a dávat podnět k zavádění vybraných požadavků do operační paměti, čímž se z nich vytvářejí úlohy připravené ke zpracování procesorem.

Plánovač vybírá požadavky podle zvolené strategie a zaznamenává pro ně podle

zadání potřebné prostředky k jejich provádění: programy, data, vyhrazené oblasti operační i vnější paměti, vstupní a výstupní zařízení. Informace o přidělených prostředcích zaznamenává plánovač do tabulek. Program, který získá všechny potřebné prostředky, se stává úlohou vybranou ke zpracování v počítači pod řízením supervizoru.

Supervizor spouští úlohu tím, že odpovídající program zavede do operační paměti. Uvolní-1i se procesor, zahájí se výpočet úlohy, který probíhá dokud se nevyskytne potřeba vstupu nebo výstupu informace. Takový požadavek dostane supervizor a obslouží ho tak, že zahájí činnost potřebného přiděleného vnějšího zařízení. Úloha čeká na dokončení I/O a supervizor přidělí procesor jiné úloze, která čeká na výpočet. Ukončení vstupní/výstupní operace se oznamuje supervizoru, jenž převede úlohu do stavu čekání na výpočet v procesoru. Když skončí zpracování nějaké úlohy, supervizor žádá plánovač o přípravu dalšího požadavku ke zpracování. Plánovací algoritmy se vybírají vždy s jistým cílem. V daných podmínkách mají zajišťovat optimální využití prostředků systému, minimální dobu odpovědi ap. Proto je v řadě případů nutno, kromě přidělení prostředků a výběru ú1ohy ke zpracování, také stanovit časový interval vymezený pro zpracování.

Všimneme si nejjednodušších plánovacích algoritmů a s nimi souvisejících charakteristických veličin a vlastností procesů, na základě kterých plánovač rozhoduje, kterému procesu přidělí procesor. Charakteristické veličiny mohou být:

t_{oi} - okamžik příchodu (vytvoření) i-tého procesu,

t_i - doba potřebná k provedení i-tého procesu,

t_{di} -doba zbývající na dokončení i-tého procesu,

P_i - statická priorita i-tého procesu (nejnižší hodnota nejvyšší přednost),

t_{bi} - doba dosavadního běhu programu ($t_{bi} \leq t_i$),

Δt_i - přidělený časový interval.

Plánovací algoritmy, které využívají výše uvedené charakteristické veličiny jsou shrnuty v tabulce:

Typ	Využívá charakteristickou veličinu	Princip
FCFS, také FIFO (first come - first served / First in - first out)	t_{oi}	řádný frontový režim
SJF (shortest job first),	t_i	proces s nejkratší dobou provádění, je první obsloužen
LCFS (least completed - first served)	t_{bi}	přednostně se obsluhuje proces, který zatím běžel nejkratší dobu

EDFS (earliest - due-time first served)	t_{di}	přednostně se obsluhuje proces, kterému zbývá nejméně času na dokončení, tj. do okamžiku, kdy musí být dokončen
HSFS (highest static priority first served)	P_i	přednostně se obsluhuje proces s nejvyšší statickou prioritou
RR (round-robin)	Δt_i	cyklická obsluha procesů po časových intervalech

Rozeznáváme tři druhy plánování procesů (process scheduling):

- **krátkodobé** (short-term), CPU scheduling (plánování procesoru): výběr, kterému z připravených procesů bude přidělen procesor,
- **střednědobé** (medium-term): výběr, který čekající (blokovaný) proces bude odsunut z vnitřní paměti na disk, je-li vnitřní paměti nedostatek (swap out, roll out),
- **dlouhodobé** (long-term), job scheduling (plánování prací, úloh): výběr, která úloha bude spuštěna (má význam zejména při dávkovém zpracování). Účelem je namixovat úlohy tak, aby byl počítač co nejvíce vytížen (třídy úloh dle náročnosti).

V jednotlivých operačních systémech se nemusí nutně používat všechny tři druhy plánování procesů. V některých případech je například plánování úloh zjednodušeno na pravidlo: **pokud je dostatek zdrojů operačního systému, spusť proces.**

4.3.1 Plánování procesoru

Předpokládejme, že výpočetní systém je vybaven jedním procesorem. Není tedy technicky možné, aby na jednom procesoru mohlo najednou běžet několik procesů. Operační systém musí současný běh programů simulovat, a tak plánovat přidělování jednoho procesoru několika procesům. Plánování procesoru se používá ve víceúlohových systémech. Může nastat v následujících situacích:

- pokud některý proces přejde ze stavu „běžící“ do stavu „čekající“ (čekání na I/O operaci, semafor, čekání na uplynutí zadaného časového intervalu, čekání na ukončení procesu-potomka),
- pokud některý proces skončí,
- pokud je některý proces převeden ze stavu „běžící“ do stavu „připravený“,
- pokud některý proces přejde ze stavu „čekající“ do stavu „připravený“.

Jestliže k plánování procesoru dochází pouze v prvních dvou výše uvedených případech, říkáme, že OS používá nepreemptivní plánování CPU (procesů). Jinak říkáme, že OS používá preemptivní plánování CPU. Přeplánování procesoru v posledním uvedeném případě se používá zřídka (například u systémů reálného času).

4.3.1.1 Přepínání programů

Multitaskový operační systém umožňuje současný běh několika programů. Přepínání programů (task switching) je předchůdcem kooperativního multitaskingu a může být realizován dvěma způsoby:

S omezeným přepínáním programů je možné přepínat jen mezi jedním „normálním“ programem - říká se mu hlavní program - a několika speciálními programy vytvořenými zvláštním způsobem výhradně pro přepínání.

S neomezeným přepínáním umožňují spuštění několika „normálních“ programů a přepínání mezi nimi.

4.3.1.2 Kooperativní multitasking

Proces, který právě běží, musí pravidelně volat systémovou službu, dává tím najevo, že může být přerušen (tato služba bývá kombinována s jinými službami systému). Dokud si uživatel nevyžádá přepnutí na jiný proces, nedělá tato služba nic (nebo pouze provede systémovou službu, se kterou je kombinována). Vyžádá-li si uživatel přepnutí procesů, zajistí zmíněná služba při nejbližším vyvolání přepnutí kontextu. Po jejím ukončení již běží nový proces a dosud aktivní proces čeká, až bude znovu aktivován.

Kooperativní multitaskový operační systém využívá přepínání mezi procesem na popředí (foreground) a procesy na pozadí (background). Výhodou je lepší využití procesoru, tzn. doba, kdy procesor čeká, vyplní zpracováním jiného procesu. Nevýhodou je zpomalení procesu na popředí a proto je nepoužitelný na realizaci paralelních úloh (správa sítě, komunikace prostřednictvím sériového rozhraní apod.). Chyba v aktivním procesu vede totiž k nekonečné smyčce a uváznutí systému.

4.3.1.3 Nepreemptivní plánování

V případě nepreemptivního plánování se proces musí procesoru sám vzdát. Pokud má být doba, po kterou je proces ve stavu „běžící“, omezená, je nutné, aby proces kontroloval časovač a po překročení stanovené doby se dobrovolně vzdal procesoru vyvoláním služby operačního systému, která je k tomuto účelu určena. Výhodou je, že proces nemůže být přerušen, pokud nechce (například v kritické sekci). Nevýhodou je, že špatně chovající se proces může zablokovat celý operační systém. Takto byl vytvořen například MS-Windows ver. 3.11.

4.3.1.4 Preemptivní plánování

V případě preemptivního plánování operační systém může odebrat procesu procesor. Zpravidla se tak děje při uplynutí časového kvanta určeného pro běh procesu a celá akce je vyvolána přerušením od časovače. Příkladem operačního systému, který používá preemptivní plánování je Unix, Windows či Vista.

4.3.2 Strategie plánování procesů

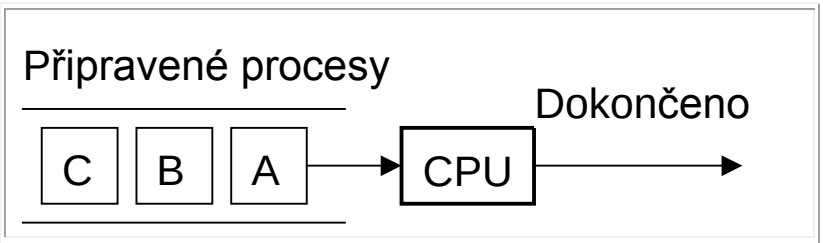
Strategie použitá pro výběr, kterému z připravených procesů bude přidělen procesor, bývá tvůrci operačního systému vybírána podle těchto kritérií:

- spravedlnost: každý proces dostane spravedlivé kvantum času procesoru,
- efektivita: udržovat maximální vytížení procesoru, příp. jiné části systému,
- čas odezvy: minimalizovat dobu odezvy pro interaktivní uživatele,
- doba obrátky: minimalizovat dobu zpracování každé dávkové úlohy,
- průchodnost: maximalizovat množství úloh zpracovaných za jednotku času.

Podle toho, které z těchto vlastností brali tvůrci systému v úvahu a jakou váhu jim přikládali, používají různé operační systému různé strategie plánování procesů.

4.3.2.1 Plánování v řádném frontovém režimu

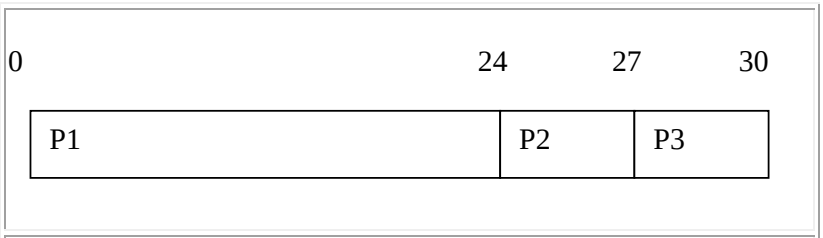
Při plánování v řádném frontovém režimu (FCFS - first come, first served - kdo dřív přijde, ten je dříve obsloužen) procesy přicházející do stavu „připravený“ jsou umísťovány na konec fronty typu FIFO (first in first out). Při plánování se procesor (CPU) přidělí tomu procesu, který je ve frontě první. Tuto strategii je možné používat při preemptivním i nepreemptivním plánování procesů. Při preemptivním plánování se někdy nazývá cyklickým plánováním (RR - round robin scheduling).



Plánování FCFS

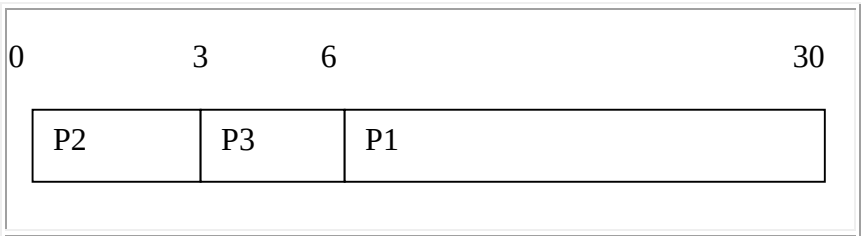
Vlastnosti tohoto způsobu plánování procesů si objasněme na následujících příkladech.

Příklad 1:
Proces P1 = 24
Proces P2=3
Proces P3=3
Procesy vznikly v pořadí P1,P2,P3
Doby čekání – P1=0, P2=24, P3=27
Průměrná doba čekání (0+24+27)/3=17



Příklad 1

Příklad 2:
Proces P1 = 24
Proces P2=3
Proces P3=3
Procesy vznikly v pořadí P2,P3,P1
Doby čekání – P2=0, P3=3, P1=6
Průměrná doba čekání (0+3+6)/3=3



Příklad 2

Závěry k plánování FCFS
Příklad 2 vykazuje lepší výsledek než Příklad 1, i když se jedná o stejné procesy. Krátké procesy následující po dlouhém procesu ovlivňuje „konvojový efekt“.

4.3.2.2 Plánování procesů s nejkratší dobou provádění

Při plánování procesů s nejkratší dobou provádění (SJF – shortest job first) se s každým procesem spojí délka jeho příští dávky procesoru. Vybírá se proces s nejkratší dobou dávky procesoru, přičemž docílíme dvou variant:
- **Nepreemptivní, bez předbíhání** – jakmile se procesor předá vybranému procesu, tento nemůže být předběhnut žádným jiným procesem, dokud dávku procesoru nedokončí.
- **Preemptivní, s předbíháním** – jakmile se ve frontě připravených objeví proces s délkou dávky procesoru kratší než je doba zbývající k dokončení dávky právě běžícího procesu, právě běžící proces je ve využívání procesoru předběhnut novým procesem (Shortest-Remaining-Time-First – SRTF).

SJF je optimální algoritmus, pro danou množinu procesů dává minimální průměrnou dobu čekání.

4.3.2.3 Prioritní plánování

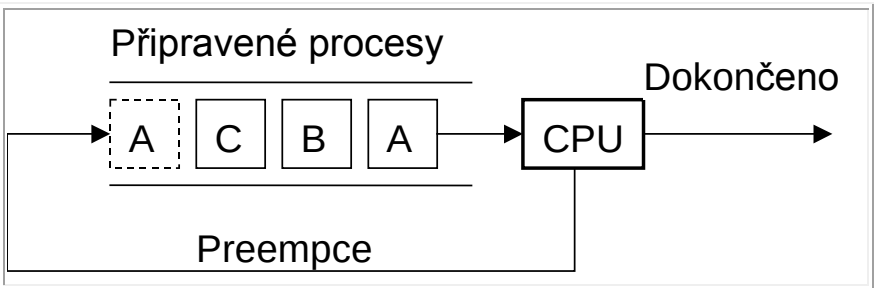
Při prioritním plánování (HSFS - highest static priority first served) je s každým procesem spojeno prioritní číslo, přičemž prioritní číslo vyjadřuje preference procesu pro výběr příště běžícího procesu. Procesor se přiděluje procesu s nejvyšší prioritou a této nejvyšší prioritě odpovídá nejnižší prioritní číslo. Opět lze pozorovat dvě varianty:

- nepreemptivní, bez předbíhání,
- preemptivní, s předbíháním.

Variantou je takové prioritní plánování, kde prioritou je předpovídaná délka příští dávky procesoru. Zde nastává tzv. problém **stárnutí, kdy** procesy s nižší prioritou se nemusí nikdy provést. Řešením je metoda **zrání, kdy** priorita se s postupem času zvyšuje.

4.3.2.4 Cyklická obsluha procesů

Plánování cyklickou obsluhou (RR - round robin scheduling) je preemptivním plánováním.



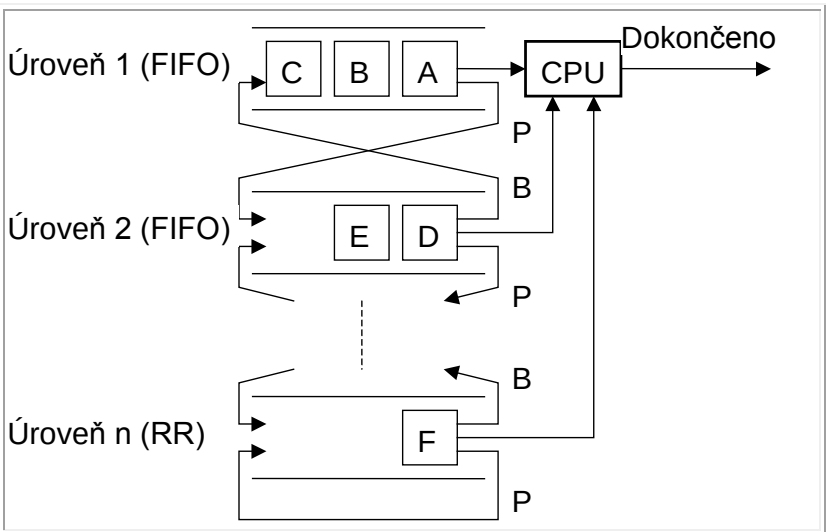
Plánování RR

Každý proces dostává procesor (CPU) na malou jednotku času – časové kvantum – např. desítky až stovky ms. Po uplynutí této doby je běžící proces předběhnut nejstarším procesem ve frontě připravených procesů a zařazuje se na konec této fronty. Je-li ve frontě připravených **n** procesů a časové kvantum je **q**, pak každý proces získává **1/n**-tinu doby CPU, najednou nejvýše po **q** časových jednotkách. Žádný proces nečeká na přidělení CPU déle než **(n-1)q** časových jednotek.

Výkonnostní hodnocení záleží v prvé řadě na velikosti přidělovaného časového kvanta **q**:

- jestliže **q** je velké, pak je plánování podobné typu FCFS,
- jestliže **q** je malé, pak může být neefektivní – **q** musí být dostatečně velké s ohledem na režii přepínání kontextu.

Určitou modifikací metody RR je začlenění více front ve zpětné vazbě, jak je naznačeno na následujícím obrázku.



Plánování s více frontami se zpětnou vazbou

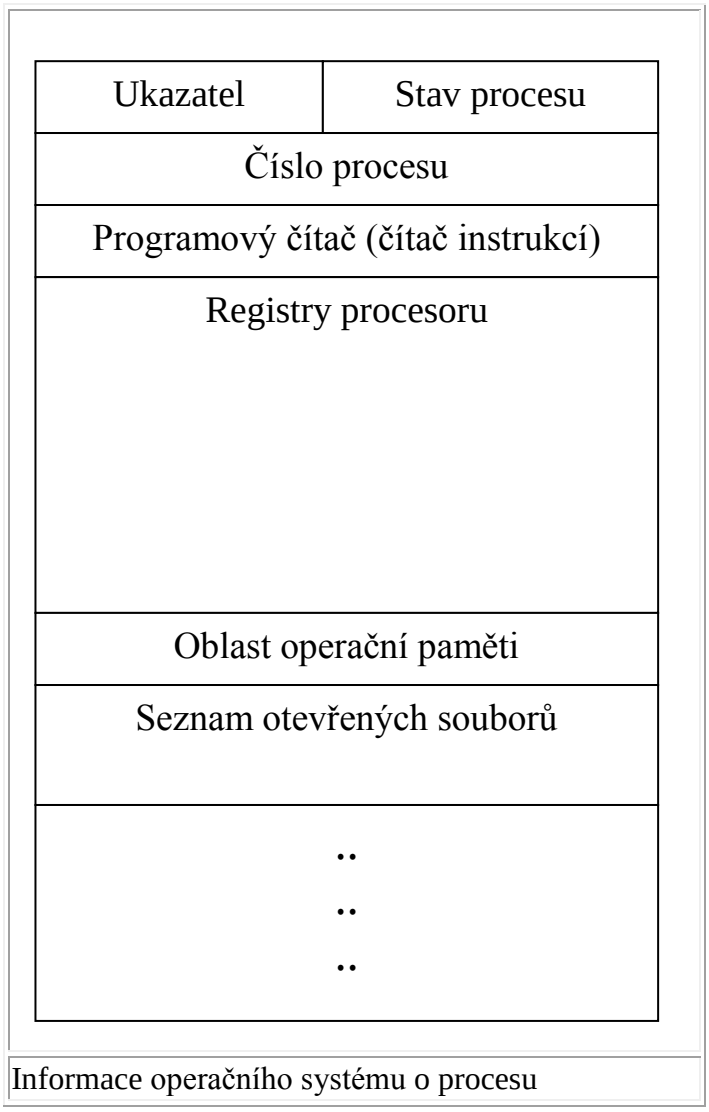
4.3.3 Kontext procesu

Informace o procesech jsou ukládány v tabulce (PCB - Process Control Block) obsahující informace potřebné pro definici a správu procesu, paměti a IO. Předně se jedná o:

- stav procesu (běžící, připravený, …)

- čítač instrukcí
- registry procesoru
- účtovací informace

Struktura PCB tabulky je zřejmá z následujícího obrázku.



Informace operačního systému o procesu

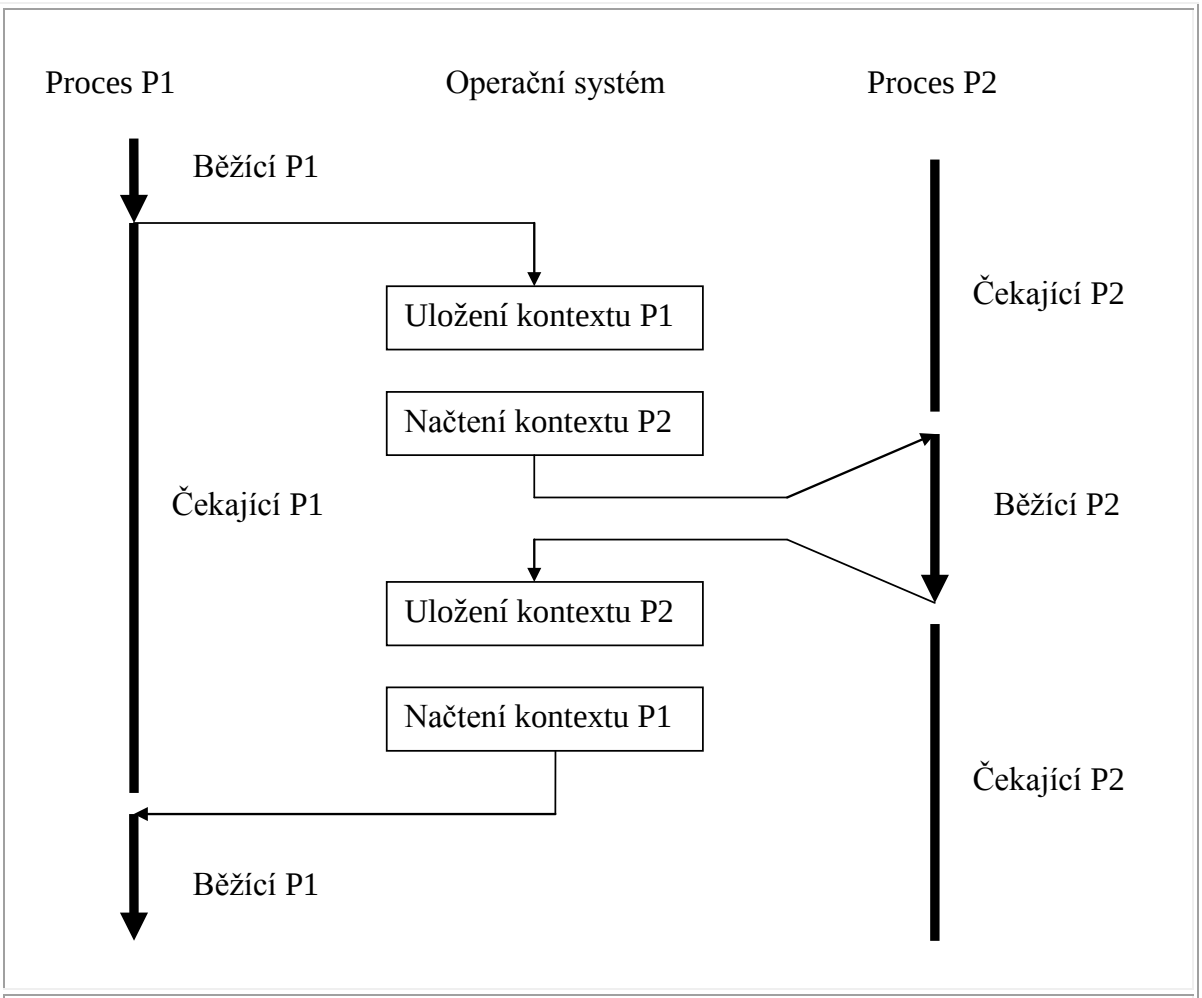
Uvedená tabulka se nazývá kontext procesu

U operačních systémů s preemtivním plánováním procesoru může být proces přerušen mezi libovolnými dvěma strojovými instrukcemi v programu a řízení může být předáno jinému procesu. Programy jsou však psány tak, že nepředpokládají, že by došlo ke změně obsahu registrů procesoru případně některých dalších oblastí mezi dvěma instrukcemi. Při přepnutí na jiný proces musí být také změněny další registry (ukazatel na tabulku stránek, klíč pro ochranu paměti, registr udávající, zda je procesor v privilegovaném stavu apod.).

Proto se při přepínání mezi procesy provádí tzv. uložení kontextu (context save) původně běžícího procesu a obnovení kontextu (context restore) procesu, kterému se přiděluje procesor. Pod pojmem kontext je myšlen stav procesoru (obsah registrů),

stav případného koprocesoru, případně i stav dalších zařízení. Tento kontext se ukládá buď do zásobníku procesu, nebo do předem připravené oblasti dat v adresním prostoru procesu.

Při přepnutí kontextu procesu se přepojuje CPU z procesu P1 na proces P2. Přitom se musí uchovat stav původně běžícího procesu (uložit v PCB procesu P1) a zavést stav nově běžícího procesu (z PCB procesu P2). Přepnutí kontextu představuje pro operační systém režijní ztrátu (zátěž), během přepínání operační systém nevykonává nic efektivního. Doba přepnutí kontextu závisí na konkrétní HW platformě, tj. počtu registrů procesoru. Přepnutí se děje pomocí přerušení, kdy se musí uchovat minimálně čítač instrukcí a zavést do čítače instrukcí hodnotou adresy vstupního bodu ovladače přerušení z vektoru přerušení. Přepnutí kontextu je zřejmé z následujícího obrázku.



Přepnutí kontextu procesu

4.4 Komunikace mezi procesy

Procesy mezi sebou komunikují pomocí dvou používaných mechanismů:

- zasílání zpráv,
- sdílená paměť.

Některé operační systémy podporují oba mechanismy.

Mechanismus **sdílené paměti** je jednodušší na programování a mocnější. Programátor má k dispozici více prostředků a implementace je i zpravidla jednodušší.

Mechanismus **zasílání zpráv** je flexibilnější a je možné jej použít i pro komunikaci mezi procesy běžícími na různých procesorech nebo počítačích.

4.4.1 Zasílání zpráv

Zaslání zprávy může být realizováno předáním ukazatele na zprávu ve sdílené paměti, umístěním zprávy do vyrovnávací paměti nebo do fronty zpráv buď ve sdílené paměti nebo v paměti jádra systému. Systém pak zajišťuje kopírování zpráv do paměťového prostoru příjemce zprávy, možná je i komunikace prostřednictvím sítě. Implementace komunikačního spojení pro zasílání zpráv se v různých systémech mohou lišit v následujících detailech:

- způsobu vytváření spojení,
- počtu procesů využívajících spojení: 2 nebo více,
- počtu spojení mezi dvěma procesy: 1 nebo více,
- kapacitou spojení tj. kolik nezpracovaných zpráv může spojení obsahovat:
  - nulová kapacita: buď musí odesilatel čekat na příjemce (rendezvous), nebo se zpráva, na níž nikdo nečeká ztratí (aby k tomu nedošlo, je nutné použít nějaký synchronizační prostředek);
  - omezená kapacita: pokud je linka zaplněná, musí odesilatel čekat,
  - neomezená kapacita: odesilatel nikdy nečeká,
- velikosti zpráv: pevná nebo proměnná,
- jednosměrnosti nebo obousměrnosti spojení,
- přímé (zpráva se zasílá konkrétnímu procesu, je přijímána od konkrétního procesu) nebo nepřímé (zpráva se zasílá a přijímá prostřednictvím poštovní schránky) komunikaci,
- symetrické nebo nesymetrické komunikaci,
- asynchronní nebo synchronní komunikaci: odesilatel musí při synchronní komunikaci čekat na odpověď (použito např. pro RPC - remote procedure call),
- automatické nebo explicitní („ruční“) použití vyrovnávacích pamětí,
- zasílání kopie nebo reference.

Ošetření chyb při zasílání zpráv musí zahrnovat tyto situace:

- jeden ze spolupracujících procesů skončil,
- ztráta zprávy,
- duplicita zprávy,
- zkomolení zprávy.

Pro předávání zpráv jsou k dispozici následující programové prostředky:

- **SEND** zašle zprávu. Nikdy se nezablokuje. Pokud na zprávu čeká příjemce operací RECEIVE, je mu zpráva doručena a příjemce odblokován.
- **RECEIVE** zprávu přijímá. Pokud žádná zpráva není dostupná, přijímající proces je zablokován.

Je třeba vyřešit problém **adresace**, tj. určení odesilatele a příjemce. Lze řešit např. identifikací procesu na daném počítači. Jiným řešením je zavedení **schránek (mailboxů)** a adresace pomocí identifikace schránky. Schránka je vyrovnávací paměť typicky pevné délky. Do ní jsou zprávy ukládány operací **SEND** a z ní vybírány

operací **RECEIVE**. Schránka modifikuje operaci **SEND**: pokud je schránka plná, zablokuje se i **SEND**.
Zajímavým případem je situace, kdy je schránka nulové velikosti. Pokud je odesilatelem proveden **SEND** a ještě tam není žádný příjemce čekající operací **RECEIVE**, musí odesilatel počkat. Podobně příjemce čeká na odesilatele, až něco zašle. Po předání zprávy, tj. je provedena operace **SEND** i **RECEIVE** na jednu schránku, se oba procesy opět rozběhnou. Tento případ se nazývá **dostaveníčko (randezvous)**.

4.4.2 Sdílená paměť

Sdílená paměť je paměť, do které má přístup více procesů. Může být použita pro komunikaci mezi procesy.

4.4.2.1 Souběh - race condition

Souběh (race condition) je situace, kdy při přístupu dvou nebo více procesů ke sdíleným datům dojde k chybě, přestože každý z procesů samostatně se chová korektně. K chybě dochází tím, že data jsou modifikována některým procesem v době, kdy s nimi jiný proces provádí několik operací, o kterých se předpokládalo, že budou provedeny jako jeden nedělitelný celek.

Datům, která jsou sdílena několika procesy tak, že při přístupu k nim by mohlo dojít k souběhu, se říká kritická oblast. Kritická sekce je nejmenší část programu, ve které se pracuje s daty v nějaké kritické oblasti, a která musí být provedena jako jeden celek. Zejména u složitějších datových struktur (obousměrné spojové seznamy, složité dynamické struktury uložené v souborech apod.) dochází často k tomu, že v určitém stadiu zpracování jsou data dočasně nekonzistentní. Pokud v tom okamžiku dojde k přepnutí kontextu na proces, který tato data také používá, může nastat souběh.

Příklady souběhu:

1) Při "současně" provedeném vkladu a výběru peněz v bance realizovaném na víceúlohovém počítači může dojít vlivem souběhu ke ztrátě vkladu:

```
1. proces (výběr)                    2. proces (vklad)
pom:=konto;
pom:=pom-10000;
                -> (context switch) ->
                                     pom:=konto;
                                     pom:=pom+20000;
                                     konto:=pom;
                <- (context switch) <-
konto:=pom;
```

Příklad souběhu

2) Dva procesy se snaží vytvořit soubor ve stejném adresáři. Pokud si zvolí stejné jméno souboru, může dojít k tomu, že první proces zjistí, že soubor tohoto jména neexistuje. Pak je přerušen druhým procesem, který také zjistí, že soubor neexistuje, vytvoří jej a zapíše do něj nějaká data. První proces potom provede operaci vytvoření

souboru, čímž data zapsaná do souboru prvním procesem smaže. Při použití jednoduchých datových struktur se souběhu dá zabránit takovým naprogramováním, že data jsou po dokončení každé strojové instrukce konzistentní:

```
1. proces (výběr)                    2. proces (vklad)
konto:=konto-10000;
                -> (context switch) ->
                                     konto:=konto+20000;
                <- (context switch) <-
```

Příklad řešení souběhu

V případě vytváření souborů může operační systém poskytovat službu, která vytvoří soubor. Pokud však soubor daného jména existuje, pokus o jeho vytvoření skončí chybou. Jestliže je provedení této služby nepřerušitelné, k souběhu nemůže dojít.

U složitějších datových struktur (například obousměrný spojový seznam) však není možné dodržet podmínku, že každá modifikace dat se provádí jedinou strojovou instrukci nebo jediným nepřerušitelným voláním systému.

4.4.2.2 Problém kritické sekce

Problémem kritické sekce je umožnit přístup ke kritické oblasti procesům, které o to usilují, při dodržení následujících podmínek:

- **výhradní přístup**; v každém okamžiku smí být v kritické sekci nejvýše jeden proces,
- **vývoj**; rozhodování o tom, který proces vstoupí do kritické sekce, ovlivňují pouze procesy, které o vstup do kritické sekce usilují; toto rozhodnutí pro žádný proces nemůže být odkládáno do nekonečna; nedodržení této podmínky může vést například k tomu, že je umožněna pouze striktní alternace (dva procesy se při průchodu kritickou sekcí musí pravidelně střídat),
- **omezené čekání**; pokud jeden proces usiluje o vstup do kritické sekce, nemohou ostatní procesy tomuto vstupu zabránit tím, že se v kritické sekci neustále střídají - mohou do této kritické sekce vstoupit pouze omezený počet krát (zpravidla pouze jednou).

Pokud o přístup do kritické sekce usiluje některý proces v době, kdy je v ní jiný proces, případně o přístup usiluje v jednom okamžiku více procesů, je nutné některé z nich pozdržet. Toto pozdržení je možné realizovat smyčkou. Toto tzv. aktivní čekání (busy waiting) však zbytečně spotřebovává čas CPU - je možné čekající proces zablokovat a obnovit jeho běh až v okamžiku, kdy proces, který je v kritické sekci, tuto sekci opustí.

4.4.3 Prostředky pro zajištění výlučného přístupu

K zajištění vzájemného vyloučení postačí dosažení toho, že žádné dva procesy nejsou ve stejném časovém okamžiku v kritické sekci.

Tato podmínka je postačující k dosažení vzájemného vyloučení, neříká však nic o efektivitě a korektnosti. K dosažení dobrých výsledků je zapotřebí dodržet následující čtyři pravidla:

- Žádné dva procesy nejsou současně ve svých kritických sekcích.
- Nemohou být udělány žádné předpoklady o rychlostech nebo počtu CPU.
- Žádný proces běžící mimo svoji kritickou sekci nesmí blokovat jiný proces.
- Žádný proces nesmí čekat nekonečně dlouho ve své kritické sekci.

4.4.3.1 Zákaz přerušení

Zákaz přerušení znemožní přepnutí kontextu. To může být nebezpečné - proces může zakázat přerušení a zhavarovat nebo se dostat do nekonečného cyklu a celý systém je pak „mrtvý“. U většiny systémů může přerušení zakázat pouze jádro operačního systému, což je zajištěno tak, že zákaz přerušení je privilegovaná instrukce. Operační systém zpravidla vnitřně používá zákaz přerušení, aby zajistil nedělitelnost posloupností instrukcí používaných při implementaci jiných synchronizačních konstrukcí.

4.4.3.2 Instrukce Test and set lock

Instrukce typu otestuj a zamkni – „test and set lock“ (TSL) nastaví proměnnou Lock (zámek) logického typu na „pravda“ (uzamčeno) a vrátí původní hodnotu této proměnné. Celá akce musí být nedělitelná (nepřerušitelná). Na počítačích, které instrukci TSL nemají, je možné ji implementovat s použitím zákazu přerušení:

```
function TestAndSet(var Lock: boolean): boolean;
begin
  DisableInterrupts;
  TestAndSet:=Lock;
  Lock:=true;
  EnableInterrupts;
end;
```

Instrukce se používá před vstupem do kritické sekce; po výstupu z kritické sekce se proměnná Lock běžným způsobem nastaví na „nepravda“ (odemčeno):

```
while TestAndSet(Lock) do { nothing } ;
... kritická sekce ...
Lock:=false;
```

Některé počítače, které instrukci TSL nemají, mohou místo ní použít instrukci, která prohodí obsah registru s obsahem proměnné v paměti (např. SWAP nebo XCHG):

```
function Swap(var a, b: boolean);
var
  Temp: boolean;
begin
  DisableInterrupts;
  Temp:=a;
  a:=b;
  b:=Temp;
  EnableInterrupts;
end;
```

Použití této instrukce vypadá takto:

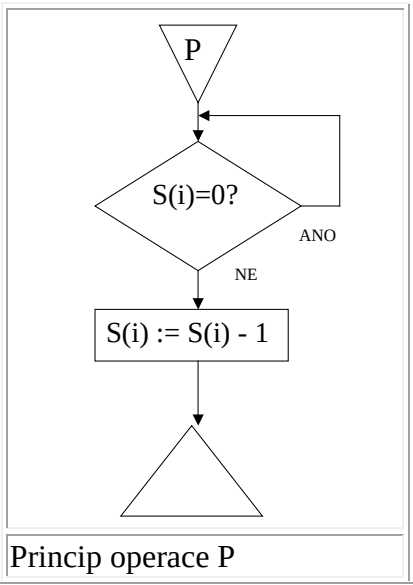
```
repeat
```

```
  x:=true;
  Swap(Lock, x);
until x=false;
... kritická sekce ...
Lock:=false;
```

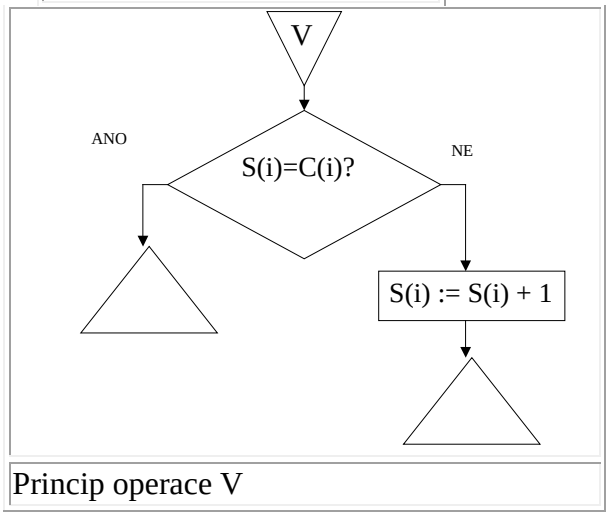
Instrukce TSL (případně SWAP a XCHG) lze použít pro zajištění výhradního přístupu. Samy o sobě však nesplňují podmínku omezeného čekání a neodstraňují aktivní čekání, mohou být však základem konstrukcí, které tyto nedostatky nemají.

4.4.3.3 Semafory

Instrukce „test and set lock“ můžeme zobecnit takovým způsobem, že dvoustavovou proměnnou typu boolean nahradíme čítačem (proměnnou typu integer). Toto zobecnění popsal E. W. Dijkstra v roce 1965. Operace byly původně nazvány P a V, podle dánských slov proberen (testovat) a verhogen (zvětšit). Některé prameny tyto operace nazývají down a up.



Princip operace P



Princip operace V

Existují dvě sémantiky vzhledem k hodnotám čítače:

1. čítač >= 0:
Operace **DOWN** zkontroluje hodnotu semaforu. Jestliže je větší než 0, sníží hodnotu semaforu o 1 a operace skončí. Jestliže je rovna 0, operace DOWN se zablokuje a příslušný proces je přidán do fronty čekajících procesů na daném semaforu.

Operace **UP** zjistí, zda je fronta čekajících procesů na daném semaforu neprázdná. Pokud ano, vybere jeden z čekajících procesů (např. nejdéle čekající) a ten odblokuje (tj. pokračuje za svou operací DOWN). Pokud je fronta prázdná, zvětší čítač semaforu o 1.

2. čítač libovolný (záporná hodnota je počet zablokovaných procesů):
Operace **DOWN** sníží hodnotu semaforu o 1. Jestliže je větší nebo roven 0, operace skončí. Jestliže je menší než 0, operace DOWN se zablokuje a příslušný proces je přidán do fronty čekajících procesů na daném semaforu.
Operace **UP** zvětší čítač semaforu o 1. Pokud je hodnota menší rovna 0, zkontroluje, zda je fronta čekajících procesů na daném semaforu neprázdná. Pokud ano, vybere jeden z čekajících procesů a ten odblokuje.

Implementace v jádře systému (s použitím zákazu přerušení) v případě použití aktivního čekání vypadá takto:

```
procedure Down(var S: semaphore);
begin
  DisableInterrupts;
  while S<=0 do begin
    EnableInterrupts;
    DisableInterrupts;
  end;
  S:=S-1;
  EnableInterrupts;
end;
procedure Up(var S:semaphore);
begin
  DisableInterrupts;
  S:=S+1;
  EnableInterrupts;
end;
```

Semafory se používají podobně jako instrukce „test and set lock“ - před vstupem do kritické sekce se vyvolá Down, po výstupu Up. Semafory popsané výše také nesplňují podmínku omezeného čekání a neodstraňují aktivní čekání.

4.4.3.4 Synchronizační prostředky odstraňující aktivní čekání

Odstranit aktivní čekání je možné tím způsobem, že pokud proces nemůže vstoupit do kritické sekce, je mu odebrán procesor a běh procesu je dočasně blokován. Aby při uvolnění kritické sekce bylo známo, zda a který proces odblokovat, přiřadí se ke každému semaforu nebo proměnné která funguje jako zámek používaný instrukcí „test and set lock“, fronta čekajících procesů:

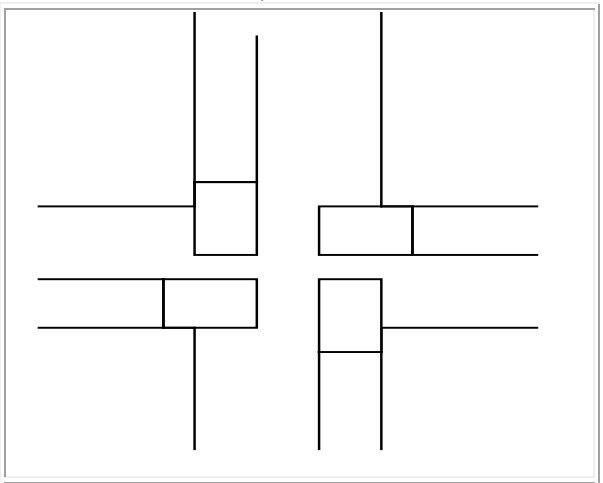
```
type Semaphore = record
  value: integer;
  queue: list of process;
end;
```

Předpokládejme, že jsou k dispozici operace Enqueue pro zařazení objektu do fronty a Dequeue pro vyřazení objektu z fronty. Operace Sleep zajistí, že aktuální proces bude zablokován (tj. jeho převeden do stavu blocked). Operace Wakeup zajistí odblokování procesu (převedení do stavu ready).

```
procedure Wait(var S: Semaphore);
begin
  DisableInterrupts;
  S.value:=S.value-1;
  if S.value<0 do begin
    Enqueue(S.queue, CurrentTask);
    Sleep;
  end;
  EnableInterrupts;
end;
procedure Signal(var S:semaphore);
begin
  DisableInterrupts;
  S.value:=S.value+1;
  P:=Dequeue(S.queue);
  if P<>nil Wakeup(P);
  EnableInterrupts;
end;
```

4.5 Uváznutí - deadlock

Pokud je ve víceúlohovém systému jednomu procesu přidělena tiskárna a druhému magnetická páska a potom první z procesů požádá o přidělení pásky a druhý tiskárny, žádný z těchto procesů nemůže pokračovat v běhu. Této situaci říkáme uváznutí neboli deadlock. Řešením této situace je odebrat jednomu z procesů prostředek, který požaduje druhý proces. Většina operačních systémů však násilné odebrání prostředků nedovoluje. Uváznutí (deadlock) - je situace, kdy dva nebo více procesů čekají na událost, ke které by mohlo dojít jedině v tom případě, že by jeden z těchto procesů pokračoval (vyřešení dopravní situace couváním).



Znázornění deadlocku situací na křižovatce

4.5.1 Podmínky uváznutí

K uváznutí může dojít jenom pokud jsou splněny všechny čtyři následující podmínky:

- Výlučný přístup (mutual exclusion). Existence prostředků, které jsou přidělovány pro výhradní použití jednomu procesu, tj. nesdílitelných prostředků.
- Postupné přidělování prostředků (hold and wait). Procesy nežádají o přidělení všech prostředků najednou, ale postupně. Pokud požadovaný prostředek není volný, musí proces čekat.
- Přidělování prostředků bez preempce. Přidělené prostředky nelze procesu násilím odebrat.
- Cyklické čekání.

4.5.2 Řešení otázky uváznutí

Operační systém se může vyrovnat s uváznutím využitím následujících strategií:

- **Ignorovat je** - přestože tato strategie vypadá nepřijatelně, používá ji například operační systém Unix; v případě uváznutí musí zasáhnout některý z uživatelů a jeden nebo více procesů násilím ukončit; v krajním případě může být nutné znovu nastartovat celý systém.
- **Předcházet mu** - zabránit splnění aspoň jedné z podmínek nutných pro vznik uváznutí.
- **Vyhýbat se mu** - systém se vyhýbá situaci, kdy by došlo k cyklickému čekání tím způsobem, že zná maximální nároky procesů na jednotlivé prostředky. Před přidělením každého prostředku zjišťuje, zda existuje způsob, jak dokončit všechny procesy i když budou vyžadovat prostředky odpovídající maximálním nárokům. Operační systém přidělí prostředek pouze tehdy, je-li to bezpečné (existuje-li způsob, jak všechny aktivní procesy zdárně dokončit).
- **Detekovat uváznutí a zotavit se z něj.**

4.5.3 Předcházení uváznutí

Strategie předcházení uváznutí se snaží zabránit splnění alespoň jedné z podmínek uváznutí.

4.5.3.1 Výlučný přístup

Prostředky, které jsou bez omezení sdílené, nemohou způsobit uváznutí. Příkladem jsou read-only soubory.

Virtualizace prostředků - používá se například u tiskáren a snímačů děrných štítků nebo pásek. Procesy nepoužívají přímo tiskárnu, ale výstupy zapisují do diskového souboru, který operační systém vytiskne později. U vstupních zařízení systém načte požadovaná data do souborů, ze kterých je pak jednotlivé procesy čtou. Tato metoda se nazývá spooling (simultaneous peripherial output on-line).

Bohužel existují prostředky, které jsou z podstaty nesdílitelné, na které není možné tuto metodu použít (např. zapisovací zařízení CD/DVD).

4.5.3.2 Postupné přidělování prostředků

Jednorázové přidělování prostředků - každý proces si vyžádá všechny prostředky, které potřebuje při svém startu a žádné další mu nebudou později přidělovány. Pokud procesu nemohou být přiděleny všechny požadované prostředky, nejsou mu přiděleny žádné a proces musí čekat.

Alternativou je strategie, při které je možné přidělovat prostředky pouze procesu, který žádné nemá. Díky tomu může proces několikrát za dobu svého běhu vrátit všechny prostředky a pak žádat o další. Tento způsob má však řadu nevýhod:

- využití prostředků je nízké, prostředky jsou vlastně přidělovány procesům, které je ve skutečnosti zatím nepotřebují,
- možnost stárnutí procesů (starvation); nároky procesu, který potřebuje několik často používaných prostředků, nemusí být nikdy uspokojeny a proces tak může čekat neomezenou dobu.

4.5.3.3 Přidělování prostředků bez preempce

Pokud je možné snadno uschovat a následně obnovit stav prostředků, může operační systém odebrat procesu prostředky, které potřebuje pro jiné procesy. Díky nutnosti uschovat a obnovit stav prostředku, je možné tuto strategii používat pouze u prostředků, které to umožňují jako je procesor nebo operační paměť.

4.5.3.4 Cyklické čekání

Algoritmus, který zamezuje cyklickému čekání:

- Každému typu prostředku je přiděleno číslo.
- Pokud má proces přiděleny nějaké prostředky, může žádat pouze o takové další prostředky, jejichž číslo je větší než největší z čísel procesem už držených prostředků.
- O prostředky, které mají stejné číslo musí žádat najednou.

4.5.3.5 Shrnutí

Některé ze strategií, které se používají pro předcházení uváznutí, je možné použít pouze pro určité prostředky (sdílitelné prostředky a prostředky, jejichž stav je možné uschovat a obnovit). Ostatní strategie mohou vést k nízkému využití prostředků a ke stárnutí procesů, což může být považováno za tak velkou nevýhodu, že v mnoha operačních systémech nejsou tyto strategie používány.

4.5.4 Vyhýbání se uváznutí

Strategie předcházení uváznutí jsou buď příliš omezující nebo nepokrývají všechny prostředky používané v příslušném systému. Pokud není možné zabránit prvním třem podmínkám vzniku uváznutí a není schůdné ani podstatné

omezení, které klade na systém výše popsaný algoritmus předcházení cyklickému čekání, je možné použít strategii vyhýbání se uváznutí.

Při žádosti o prostředek systém kontroluje, zda existuje alespoň jeden proces, který je možné po přidělení tohoto prostředku dokončit. Po jeho dokončení opět musí existovat alespoň jeden proces, který může být dokončen a tak dále až po ukončení všech procesů. Pokud by tato podmínka nebyla splněna, systém prostředek nepřidělí.

Nejznámější strategií je **bankéřův algoritmus**. Každý proces deklaruje pro každý typ prostředku maximální množství, které bude potřebovat. Operační systém si musí při přidělování prostředků ponechat tolik prostředků, aby byl schopen uspokojit maximální požadavky alespoň jednoho procesu. Tím je zaručeno, že tento proces skončí a uvolní prostředky, které pak mohou být přiděleny dalším procesům. Operační systém nekontroluje pouze, zda je možné dokončit jeden proces, ale všechny aktuální procesy.

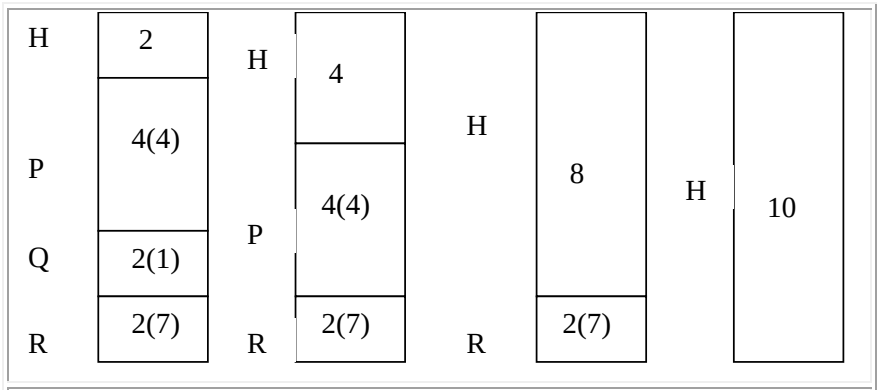
Definice problému:
Bankéř chce rozdělit pevný kapitál abstraktních peněžních jednotek mezi pevný počet zákazníků. Každý zákazník udává předem svou maximální potřebu peněžních jednotek. Bankéř vyhoví zákazníkovi, pokud jeho potřeba nepřevyšuje kapitál bankéře. Během transakcí zákazník může vracet nebo si půjčovat peněžní jednotky. Bankéř musí zaručit, že čekací doba zákazníka na přidělení peněžních jednotek bude vždy konečná. Běžná půjčka zákazníka nesmí nikdy převýšit jeho maximální potřebu. Zákazník zaručuje, že dokončí své transakce a splatí půjčku v konečném čase.

Pro popis algoritmu si definujme pojmy:

- Stav jistý – jestliže bankéř umožní všem zákazníkům dokončení všech jejich transakcí v konečném čase.
- Stav nejistý – opačná situace.

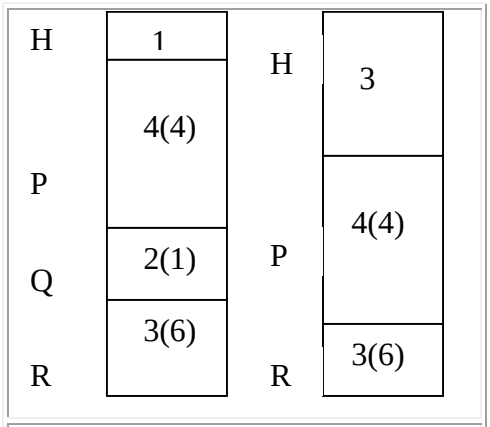
- Situace zákazníka: požadavek = potřeba – půjčka.
- Situace bankéře: hotovost = kapitál – součet půjček.

Objasnění algoritmu si ukážeme na následujícím příkladě. Bankéř má k dispozici 10 jednotek a zákazníci P, Q, R mají celkovou potřebu 20 jednotek. Některé jejich potřeby již byly realizovány půjčkou. Půjčku a požadavek zapíšeme v následujících obrázcích pomocí formalismu - (půjčka) požadavek. Tzn. zákazník P již obdržel půjčku 4 jednotky a požaduje ještě 4 jednotky, obdobně zákazník Q již obdržel 2 jednotky a požaduje ještě 1 jednotku a konečně zákazník R již obdržel 2 jednotky a požaduje ještě 7 jednotek. Zůstatek hotovosti H je na počátku tedy 2 jednotky. Algoritmus vyšetřuje zákazníky jednoho po druhém a hledá toho, jehož požadavek nepřevyšuje hotovost H.



Stav jistý – bankéř nejdříve uspokojí zákazníka Q

Z uvedeného příkladu je zřejmé, že bankéř uspokojil postupně všechny zákazníky. Následující obrázek naznačuje neřešitelnou situaci, do které se bankéř dostal tím, že si nevytvořil dostatečnou rezervu v hotovosti k postupnému uspokojení zákazníků.



Stav nejistý – bankéř přidělil R jednu jednotku a po uspokojení Q se navodí neřešitelná situace

Pro sestavení algoritmu si definujme proměnné. Stav systému je definován hodnotami:

R(i) co existuje.

C(j,i) co se požaduje pro typy prostředků **i** a procesy **j**

Počet přidělených prostředků typu **i** procesu **j**

A(j,i) pro všechna **(j,i)**

Souhrnný počet dostupných prostředků typu **i**

V(i)=R(i)- SUMA A(k,i)

Počet prostředků typu i potřebných pro dokončení procesu **j**

N(j,i)=C(j,i)-A(j,i)

Bankéř prostředek přidělí, pokud systém po přidělení zůstane v bezpečném stavu.

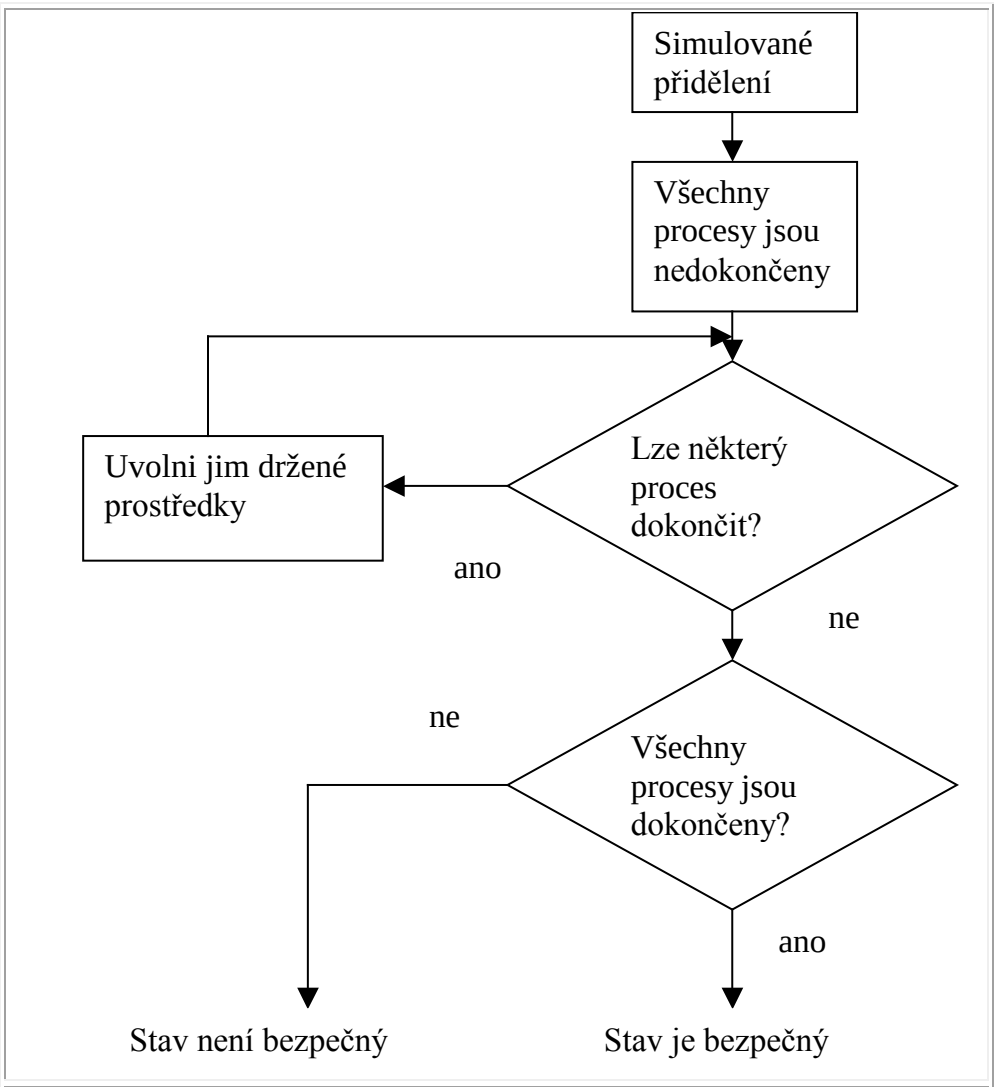
Požadovaný počet prostředků typu **i** procesem **j**

Q(j,i)

Pak **rámcový algoritmus** může být napsán:

```
IF Q(j,i) <= N(j,i) pro všechna i THEN pokračuj
      ELSE oznámení chyby procesu /*nesplnitelný požadavek*/
IF Q(j,i) <= V(j,i) pro všechna i THEN pokračuj
      ELSE čekej /*prostředek i není dostupný, čekej na jeho
uvolnění*/
      V(j,i) = V(i) - Q(j,i) pro všechna i
      A(j,i) = A(j,i) + Q(j,i) pro všechna i
      N(j,i) = N(j,i) - Q(j,i) pro všechna i
IF výsledný stav je bezpečný, THEN prostředek se procesu j
přidělí
ELSE proces j musí čekat na splnění požadavku Q(j,i) a obnoví
se původní stav
```

Vyjádření tohoto rámcového algoritmu vývojovým diagramem je zřejmé z následujícího obrázku.



Algoritmus bankéře

Detailní algoritmus pak může být popsán následujícím textem:

```
INICIALIZACE: všechny procesy se prohlásí za neukončené;
      nastaví se pracovní vektor počtu dostupných prostředků
W(i);
```

```
W(i)=V(i) pro všechna i;
REPEAT: nalezni neukončený proces j, který má počet prostředků
typu i potřebných pro dokončení N(j,i) <=W(i) pro všechna i
            N(j,i)=C(j,i)-A(j,i)   počet   prostředků   typu   i
potřebných pro dokončení procesu j
IF takový proces j neexistuje
      THEN GO TO EXIT
      ELSE označ takový proces za ukončený a uvolni jeho
prostředky:
            W(i)=W(i)+A(j,i) pro všechna i
GO TO REPEAT
EXIT: stav:=IF všechny procesy ukončené   THEN BEZPEČNÝ
                                          ELSE NENÍ BEZPEČNÝ
```

Příkladem použití bankéřova algoritmu je prověření jistého stavu pro **m** prostředků a **n** procesů. Algoritmus určuje, zda paralelní stav je jistý před zablokováním, prověřováním **m** požadavků od každého procesu, dokud není nalezen takový proces, který může být dokončen. Algoritmus opakuje tuto prověrku, dokud není eliminováno všech **n** procesů.

Maximálně je doba úměrná

$$m(n+n-1+n-2+\ldots+1)=mn(n+1)/2$$

Na závěr si shrneme prostředky a problémy synchronizace procesů:

Prostředek synchronizace
Kritické činnosti
Semafory
Vyrovnávací paměti zpráv
Fronty událostí

Problém synchronizace
Vzájemné vylučování
Výměna časovacích signálů
Výměna dat
Explicitní plánování procesů

4.6 Vlákna

Vlákno (anglicky: ***thread***) v operačních systémech označuje vlákno výpočtu, tedy posloupnost po sobě jdoucích operací. Každá spuštěná aplikace má alespoň jeden proces a každý proces má alespoň jedno vlákno, ve kterém počítá. V předchozím výkladu procesů obecně platilo, že proces měl právě jedno vlákno (přesněji, nebylo důvod tyto pojmy odlišovat). Současné operační systémy jsou vícevláknové, tedy uvnitř jednoho procesu (a v jednom adresním prostoru, tedy se sdílenou pamětí) může najednou běžet více vláken. Vlákno se pak stává elementární jednotkou pro plánovač operačního systému.

Jak jsme si již uvedli v předchozích kapitolách, multitasking je schopnost operačního systému mít spuštěno více programů současně. V zásadě operační systém používá hardwarové hodiny a každému běžícímu procesu přiděluje

„časová kvanta“. Pokud jsou tato kvanta dostatečně malá a počítač není přetížen spoustou programů, které se všechny snaží něco dělat, uživatel má dojem, jako kdyby všechny programy běžely současně.

Multithreading je schopnost programu zavést multitasking sám v sobě. Program se může rozdělit na několik samostatných prováděcích vláken, která běží současně. Tato myšlenka může na první pohled vypadat velice užitečně, ukazuje se ale, že programy používají multithreading zejména pro zpracovávání dlouhodobějších úkonů na pozadí, aniž by uživatel musel přerušit práci s programem.

Jeden z hlavních důvodů pro využití vláken je v multiprocesorových strojích, kdy vlákna jednoho procesu mohou běžet na různých procesorech. Typickým příkladem je centrální procesor CPU a grafický procesor, kdy jedno vlákno provádí uživatelem požadovaný úkol na CPU a druhé vlákno překresluje obrazovku v grafickém procesoru.

Dalším příkladem využití vláken je v síťových souborových serverech. Server musí vyřizovat řadu požadavků klientů a pro vyřízení každého požadavku vytváří samostatné vlákno, které je efektivnější než samostatný proces. Jedno vlákno zobrazuje menu a čte vstup od uživatele a současně druhé vlákno provádí příkazy uživatele.

Praktickým rozdílem mezi multithreadovým a multitaskovým operačním systémem je velikost režie při přepínání vláken nebo procesů. Přepnutí mezi vlákny je výrazně rychlejší a ve většině případů realizovatelné bez volání jádra operačního systému. Rychlejší je i vytváření a rušení vlákna a konečně vlákno spotřebuje méně paměti, což je důležité pro aplikace, které používají větší množství vláken.

Podobně jako procesy se vlákna jeví jako běžící současně. Jádro systému plánuje jejich činnost asynchronně - vlákna přerušuje, aby přidělilo čas procesoru jiným vláknům.

Koncepčně je vlákno řešeno tak, že běží uvnitř procesu. Vlákno vlastně představuje realizaci procesu na „jemnější“ úrovni. Když spustíme nějaký program, operační systém vytvoří nový proces a v tomto procesu vytvoří vlákno. Toto vlákno může vytvářet další vlákna - všechna tato vlákna realizují tentýž program a běží v tomtéž procesu. Přitom každé vlákno může realizovat jinou úlohu programu, a to v kterémkoliv daném čase.

Všechna vlákna v procesu sdílí tutéž paměť, tytéž deskriptory a ostatní systémové zdroje. Pokud nějaké vlákno změní hodnotu určité proměnné, bude tato proměnná modifikována pro všechna vlákna v procesu. To samé platí pro deskriptory souborů, pokud jedno vlákno uzavře deskriptor souboru, ostatní vlákna nemohou dále provádět operace s tímto souborem. Protože všechna vlákna běží v rámci jediného procesu, platí, že pokud jedno vlákno vyvolá přepnutí kontextu procesu, všechna ostatní vlákna se ukončí. Ukončení procesu ukončuje všechny vlákna existující v rámci tohoto procesu.

Obdobně jako u procesu si každé vlákno udržuje informace důležité pro přepnutí vlákna tj:

- zásobník,
- čítač instrukcí,
- registry procesoru,
- tabulku TCB (Thread Control Block).

Vlákna v jednom programu jsou součástí stejného procesu, takže sdílejí všechny prostředky procesu, jako je například paměť nebo otevřené soubory. Protože vlákna sdílejí paměť programu, sdílejí také jeho statické (neměnné) proměnné. Každé vlákno má ale vlastní zásobník, takže automatické proměnné jsou v rámci vlákna jedinečné. Každé vlákno také uchovává vlastní kontext procesoru uložený v tabulce TCB, který se ukládá a obnovuje při přepínání vláken.

Vlákno tak může přistupovat k paměti a ostatním zdrojům svého procesu, zdroje procesu sdílí všechna vlákna jednoho procesu. Jakmile jedno vlákno změní obsah společné paměti (nelokální – mimo zásobník), všem ostatním vláknům (téhož procesu) je tato změna k dispozici. Obdobně soubor otevřený jedním vláknem mají k dispozici všechny ostatní vlákna (téhož procesu).

Vlákno se může nacházet v jednom, ze tří stavů:

- běžící,
- připravený,
- čekající.

Oproti procesům se vlákna samostatně neodkládají na vnější paměť, všechna vlákna jednoho procesu sdílejí stejný adresový prostor v paměti.
Vlákna se dělí na:

- Vlákna na uživatelské úrovni - User-Level Threads (ULT).
- Vlákna na úrovni jádra - Kernel - Level Threads (KLT).

Správa vláken na **uživatelské úrovni** (ULT) se provádí prostřednictvím vláknové knihovny (thread library) na úrovni uživatelského/aplikačního programu. Jádro operačního systému o jejich existenci neví, přepojování mezi vlákny nepožaduje provádění funkcí jádra, nepřepíná se ani kontext procesu ani režim procesoru. Plánování přepínání vláken je specifické pro konkrétní aplikaci, kdy aplikace si volí pro sebe „nejvhodnější“ algoritmus.

Vláknová knihovna (threads library) obsahuje funkce pro:

- vytváření a rušení vláken,
- předávání zpráv a dat mezi vlákny,
- plánování běhů vláken,
- uchovávání a obnova kontextů vláken.

Jádro operačního systému pro vlákna na uživatelské úrovni neví o aktivitě vláken, proto manipuluje s celými procesy. Když některé vlákno zavolá službu jádra, je blokován celý proces dokud se služba nesplní. Pro knihovnu je takové vlákno stále ve stavu „běžící“. Stavy vláken jsou na stavech procesu nezávislé.

Výhody vláken na uživatelské úrovni:

- přepínání mezi vlákny nepožaduje řešení jádrem operačního sytému a tím se docílí vyšší rychlosti přepínání,
- nepřepíná se ani kontext procesu ani režim procesoru,
- plánování vláken je specifické pro konkrétní aplikaci,
- aplikace si volí pro sebe nejvhodnější plánovací algoritmus,
- vlákna mohou běžet pod kterýmkoliv operačním systémem,
- není vyžadována podpora na úrovní jádra operačního systému,
- vlákna mají svoji uživatelskou knihovnu ke spojení s aplikací.

Nevýhody vláken na uživatelské úrovni:

- většina volání služeb operačního systému způsobí blokování celého procesu (tj. všech vláken procesu),
- jádro může přidělovat procesor pouze procesům, dvě vlákna stejného procesu nemohou běžet na dvou procesorech.

Správu vláken na **úrovni jádra** (KLT) podporuje jádro operačního systému, nepoužívá se vláknové knihovny (thread library) ale používá se aplikační rozhraní (API) pro vláknové služby jádra. Informaci o kontextu procesů a vláken udržuje jádro, přepínání mezi vlákny aktivuje jádro a plánování na bázi vláken je řízeno jádrem operačního systému.

Výhody vláken na úrovni jádra:

- jádro může současně plánovat běh více vláken stejného procesu na více procesorech,
- k blokování dochází na úrovni vlákna (není blokován celý proces),
- programy jádra mohou mít vícevláknový charakter.

Nevýhody vláken na úrovni jádra:

- přepojování mezi vlákny stejného procesu zprostředkovává jádro operačního systému a tím je přepínání pomalejší,
- při přepnutí vlákna se dvakrát přepíná režim procesoru (tj. režie navíc).

Je tedy zřejmé že, vlákna je možné vytvořit i čistě aplikačně bez operačního systému (například pokud podporu multithreadingu nemá). Takto vzniklá vlákna mohou být spouštěna postupně v jednom vláknu operačního systému nebo v několika vláknech operačního systému může být současně spouštěn větší počet aplikačních vláken. Toto řešení sice není tak dobré jako řešení s podporou operačního systému - například volání služby operačního systému zablokuje "větší" vlákno operačního systému a ne jenom aplikační vlákno - ale pro některé úlohy může být stále rychlejší.

Kontrolní otázky:

1. Které zdroje potřebuje operační systém pro svoji činnost?
2. V čem je odlišnost způsobu řízení procesů pomocí 5-stavového model od 3-stavového?
3. Vysvětlete rozdíl mezi preemptivním a nepreemptivním plánováním procesů?

4. Jakými mechanismy spolupracují mezi sebou procesy v operačním systému?
5. Co je to uváznutí procesů?
6. Jaké znáte metody synchronizace procesů?
7. Čím se liší multitasking od multitreadingu?

Úkoly k zamyšlení:

1. Rozeznáváme tři druhy plánování procesů: krátkodobé, střednědobé a dlouhodobé. Pokuste se rozhodnout ve které úrovni 5-stavového diagramu jednotlivé plánovače pracují.

Korespondenční úkol:

1. Představte si rozhraní člověka s počítačem řízeným hlasem. Napište některé příkazy, které by musel operační systém interpretovat, aby mohl vykonávat nejzákladnější funkce.

Shrnutí obsahu kapitoly

V této kapitole jste se seznámili s principy plánování procesů operačních systémů. Důraz v této kapitole byl kladen na pochopení algoritmů synchronizace procesů. Velká pozornost byla věnována vysvětlení mechanismů spolupráce procesů.

5 Správa hlavní paměti

V této kapitole se dozvíte:

- Jaké jsou hlavní úkoly správce paměti?
- Jaké jsou strategie přidělování paměti?
- Co je to virtuální paměť?
- Jak se chrání paměť proti neoprávněným přístupům?

Po jejím prostudování byste měli být schopni:

- Charakterizovat základní funkce správce paměti.
- Znát strategie přidělování paměti více procesům.
- Porozumět principům vytváření virtuální paměti.
- Popsat způsoby ochrany paměti.

Klíčová slova této kapitoly:

Správce paměti, virtuální paměť, fragmentace paměti, ochrana mezními registry, ochrana pomocí zámků a klíčů.

Doba potřebná ke studiu: 5 hodin

Průvodce studiem

Studium této kapitoly je taktéž poměrně náročné. V případě, že jste pochopili předchozí kapitolu o procesech bude se Vám lehčeji studovat tuto kapitolu.
Na studium této části si vyhraďte alespoň 5 hodin. Doporučujeme studovat s přestávkami vždy po pochopení jednotlivých podkapitol. Po celkovém prostudování a vyřešení všech příkladů doporučujeme dát si pauzu, třeba 1 den, a pak se pusťte do vypracování korespondenčních úkolů.

Správce hlavní paměti patří mezi další velmi důležité - i když ne nejkomplikovanější moduly každého operačního systému.

Hlavním úkolem správce paměti je:

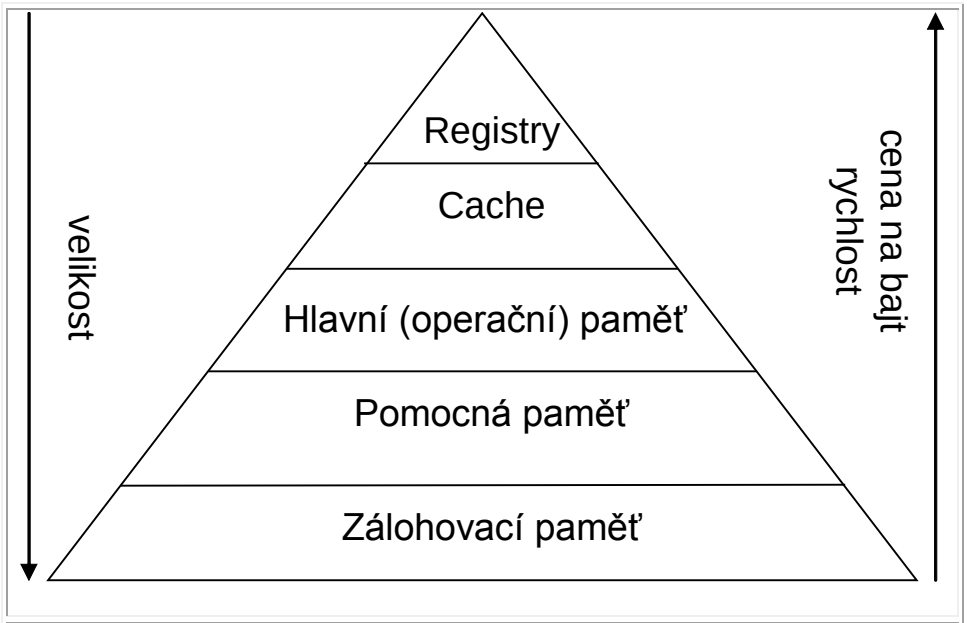
- **Přidělovat** operační paměť jednotlivým procesům, když si ji vyžádají.
- **Udržovat** informace o paměti, o tom, která část je volná a která přidělená (a komu).
- **Zařazovat** paměť, kterou procesy uvolní, opět do volné části.
- **Odebírat** paměť procesům, je-li to zapotřebí.
- **Zajistit ochranu** paměti (umožňuje-li to technické vybavení) - žádný proces by neměl mít přístup k paměti jiného procesu nebo operačního systému, jestliže mu to vlastník paměti explicitně nepovolí.

Požadavky na správu paměti:

- **Možnost relokace.** Programátor nemůže vědět, ze které části paměti bude jeho program prováděn. Relokace neumožňuje, aby se adresy kontrolovaly během kompilace. Odkazy na adresy se musí kontrolovat při běhu procesu hardwarem.

- Procesu může být **dynamicky** při výměnách (odebírání a vracení prostředku procesu) **přidělována** jiná oblast paměti – **swapping**.
- **Odkazy** na paměť v LAP (logickém adresovém prostoru) se musí dynamicky překládat na skutečné ve FAP (fyzickém adresovém prostoru).
- **Nutnost ochrany.** Procesy nesmí být schopné se bez povolení odkazovat na paměťová místa, přidělená jiným procesům.
- **Logická organizace.** Uživatelé tvoří programy jako moduly s odlišnými vlastnostmi - execute, datové read-only, read/write, některé moduly soukromé (private) a jiné veřejné (public).
- **Možnost sdílení.** Více procesů může sdílet společnou část paměti, aniž by se tím porušovala ochrana paměti. Důvodem pro sdílený přístup ke společné datové struktuře (sdílení jediného exempláře datové struktury) je lepší řešení, než udržování konzistence jejich násobných kopií vlastněných jednotlivými procesy.
- **Řízení paměti.** Existují tři různé kategorie řízení paměti:
  - **Načítání (fetch).** Jedná se o chování paměti při získávání dalších částí procesů nebo dat, např. starší na žádost (on demand), nyní v předstihu (anticipatory).
  - **Umísťování (placement).** Zde se jedná o požadavek, kde v paměti umístit požadovaný blok dat.
  - **Výměna (replacement).** Jedná se o požadavek, co z paměti odstranit pro nově příchozí data.

Požadavky, které se v počítačovém systému kladou na paměti se zatím nedají ekonomicky splnit jedinou pamětí. Čím větší je kapacita paměti, tím větší je totiž i její poměrná cena. Podobně roste cena i se zkracováním vybavovací doby. Proto rychlé paměti (s krátkou vybavovací dobou) mají obvykle malou kapacitou, pomalé paměti (s delší vybavovací dobou) mají velkou kapacitu. Začlenění jednotlivých úrovní paměti je naznačeno na obrázku.



Hierarchie paměti

Z pohledu uvedené hierarchie paměti se správa paměti operačního systému zabývá převážně řízením hlavní (operační) paměti.

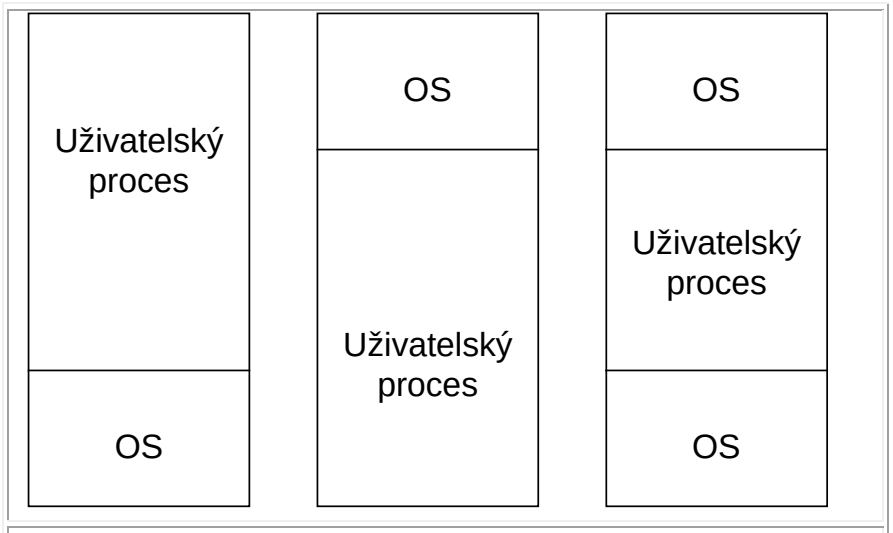
5.1 Strategie přidělování paměti

Existují různé strategie přidělování paměti:

- přidělování veškeré volné paměti,
- přidělování pevných bloků paměti,
- přidělování bloků paměti proměnné velikosti,
- segmentace paměti,
- stránkování paměti,
- stránkování na žádost,
- segmentace se stránkováním na žádost.

5.1.1 Přidělování veškeré volné paměti

Část operační paměti je obsazena operačním systémem (kódem programu, proměnnými, vyrovnávací pamětí), zbytek je k dispozici pro uživatelský program. V každém okamžiku je tedy v paměti nejvýše jeden uživatelský program.
V jednouživatelských operačních systémech je přidělování paměti naznačeno na následujícím obrázku.



Přidělování paměti v jednouživatelských OS

Tato strategie byla např. používána na začátku 60. let pro operační systémy osmibitové mikropočítače (CP/M, ISIS-2 od Intelu).

Operační paměť byla v operačním systému CP/M rozdělena následovně:

- Prvních 100 byte je určeno operačnímu systému (údaje o běžícím programu, parametry).
- V další části je paměť pro program (a volné místo).
- Na konci paměti proměnné systému a buffery, BDOS a BIOS (v ROM)
- Adresování pro programy začíná vždy na adrese větší než 100. Zde jsou ukládány přeložené programy připravené pro spuštění.

- Velikost operační paměti je maximálně 64 KB, a proto operační systém a programy musí být přizpůsobeny velikosti paměti.

Operační paměť v operačním systému ISIS-2 byla rozložena následovně:

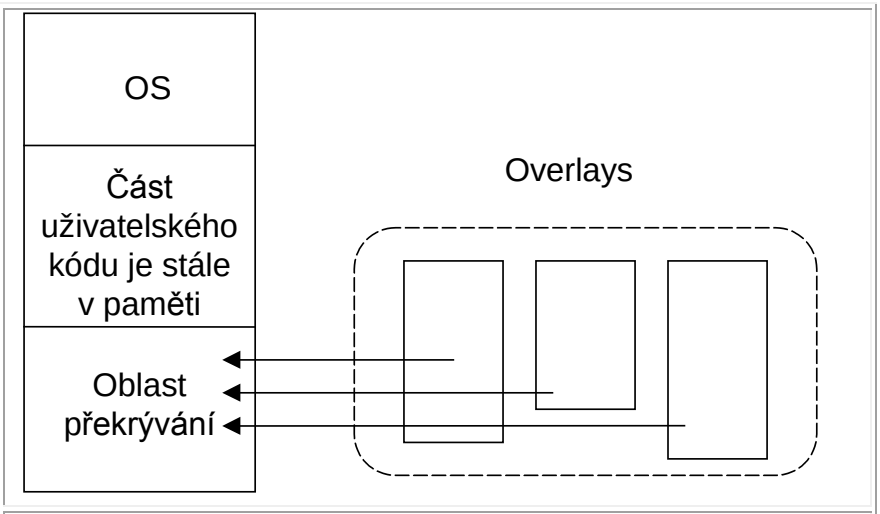
- Na začátku paměti je operační systém, pak proměnné a buffery.
- Následuje pak volné místo pro program.

Podle velikosti dostupné paměti, velikosti jádra operačního systému a počtu bufferů bylo nutné v CP/M přeadresovávat operační systém a v ISIS-2 přeadresovávat uživatelské programy. Přeadresování prováděl program zvaný locator. Locator podle tabulky adres v relativním programu změnil (dle umístění) příslušné adresy na absolutní.

Většina procesorů, na kterých se provozují systémy s touto strategií přidělování paměti, neumožňuje žádnou zvláštní ochranu paměti, pouze ochrana operačního systému před přepsáním uživatelským programem (pomocí mezního registru; změna pouze v privilegovaném stavu procesoru). V uživatelském stavu není možno zapisovat na adresy větší nebo rovné obsahu mezního registru, ani měnit obsah mezního registru.

V omezené míře je možné i při této strategii přidělování paměti používat multitasking. Při přepnutí kontextu je nutné nahrát celý obsah uživatelské oblasti paměti na disk a zavést z disku obsah adresního prostoru druhého procesu.

V některých případech je možno uživatelský program rozdělit na část trvale umístěnou v operační paměti a části, které jsou na sobě nezávislé, a proto mohou být překrývané. Tyto části jsou pak umístěny na disku a zavádí se do překrývané oblasti paměti v případě jejich provádění, jak je naznačeno na následujícím obrázku.

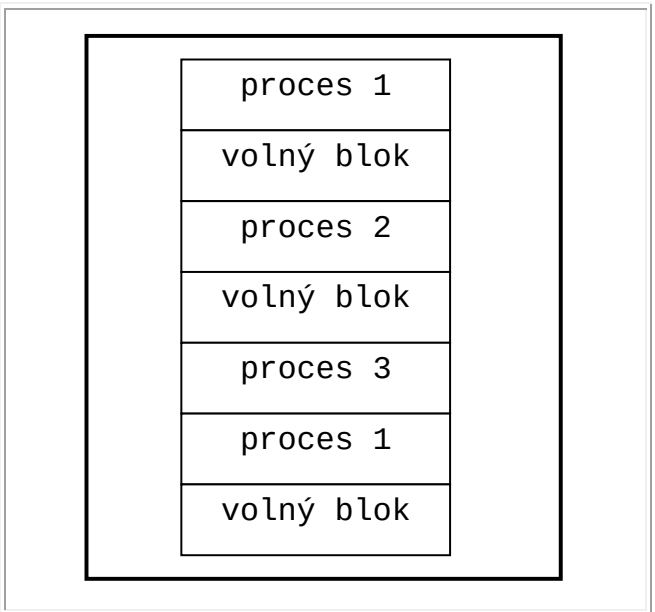


Přidělování paměti v jednouživatelských OS

5.1.2 Přidělování pevných bloků paměti

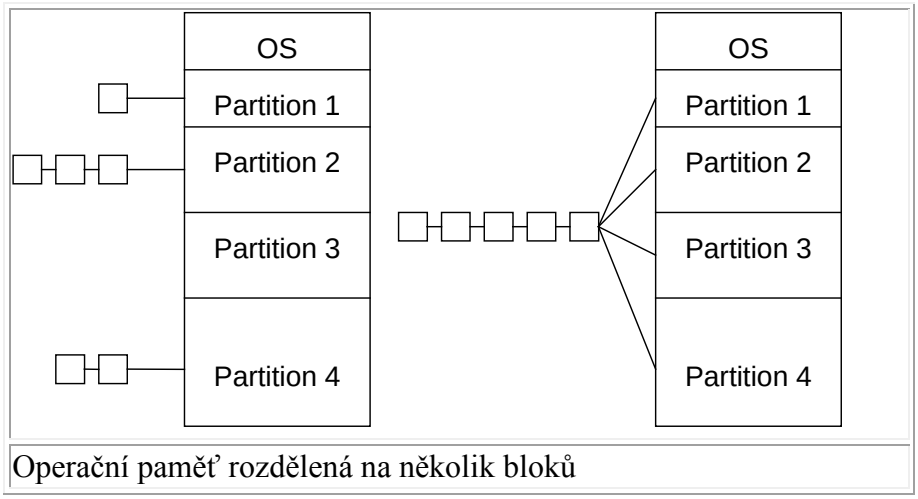
Většina jednoduchých operačních systémů využívala právě tuto strategii - jako příklady můžeme jmenovat MS DOS. Její základní princip je velmi jednoduchý: každý proces musí vědět, kolik operační paměti bude potřebovat a musí si tuto paměť od operačního systému explicitně vyžádat. Operační systém - respektive správce paměti - takový požadavek bud' splní přidělením bloku požadované velikosti, nebo zamítne, a v tom případě je úkolem procesu vzniklý problém nějak vyřešit (např. předčasným ukončením práce).

Na následujícím obrázku vidíme příklad operační paměti rozdělené na několik bloků. Operační paměť používají tři procesy; proces 1 má přiděleny dva bloky paměti, zbývající dva bloky mají jen po jednom bloku.



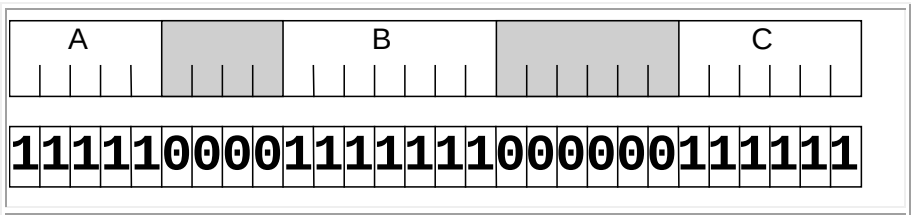
Operační paměť rozdělená na několik bloků

Paměť je rozdělena na bloky pevné velikosti a procesy jsou ve frontách podle požadované velikosti. Velkou nevýhodou je nutnost znát maximální potřebnou paměť před spuštěním a nevyužití částí oddílů paměti.



Operační paměť rozdělená na několik bloků

Procesy získávají paměť dle potřeby (pokud je dostupná). Nevýhodou je vytvoření nepoužitelných děr mezi použitými bloky paměti. Tuto nevýhodu lze odstranit řízením paměti pomocí bitové mapy. Paměť je rozdělena na bloky stejné délky jak je naznačeno na následujícím obrázku.



Řízení paměti pomocí bitové mapy

Dalším řešením je řízení paměti pomocí spojového seznamu. Operační systém má spojový seznam všech bloků (volných i použitých). Jinou variantou jsou dva spojové seznamy, jeden pro volné bloky, druhý pro bloky použité. Operační systém pak musí řešit dva problémy:

- rozhodnout, který z volných bloků přidělí při novém požadavku procesu na paměť (strategie přidělování),
- při uvolnění bloku sloučit nový volný blok s okolními volnými bloky.

5.1.2.1 Informace o blocích

Správce paměti musí o každém bloku vědět nejméně dvě věci:
- jeho délku,
- jeho vlastníka.

Je-li informace o vlastníkovi paměťového bloku prázdná, znamená to, že blok je volný a může být přidělen. Adresu dalšího bloku nemusíme explicitně uvádět, protože bloky na sebe samozřejmě v operační paměti navazují; adresu následujícího bloku tedy snadno spočítáme na základě délky a adresy aktuálního bloku. Správce paměti pak již potřebuje pouze znalost adresy prvního bloku, pak může snadno se spojovým seznamem bloků pracovat. Správce paměti může samozřejmě tyto údaje udržovat někde uvnitř vlastních datových struktur.

Toto řešení však má jednu podstatnou nevýhodu. Pro údaje, musíme vyhradit dostatek paměti. Velikost potřebné paměti je však úměrná počtu přidělených bloků paměti; jejich počet však pochopitelně není možné určit předem. S jistotou samozřejmě víme, že bloků paměti bude méně, než je v paměti k dispozici bytů; tolik prostoru pochopitelně není možné pro systémové tabulky vyhradit. Vyhradíme-li však méně místa, riskujeme, že ve výjimečném případě, kdy budou procesy požadovat velké množství malých bloků, vyhrazená paměť nepostačí.

Obvyklým a poměrně efektivním řešením tohoto problému je přidělit na žádost procesu vždy blok o několik bytů větší a do tohoto volného místa uložit informace o bloku samotném i o adrese bloku následujícího. V operační paměti tak vznikne spojový seznam volných i alokovaných bloků; správce paměti pak

při každém požadavku tento seznam prochází a zpracovává jeho položky - jednotlivé bloky.

Toto řešení má samozřejmě také svou nevýhodu. Systémové informace o blocích paměti jsou „zamíchány“ mezi paměťové bloky přidělené jednotlivým procesům. Jestliže technické vybavení počítače, na kterém operační systém provozujeme, neumožňuje zabezpečení ochrany paměti, je poměrně velké riziko, že některý chybný program systémové informace poškodí. V takovém případě prakticky není možné pokračovat v činnosti operačního systému, protože poškozením spojového seznamu bloků paměti je vyřazen vlastně celý správce paměti - uvědomme si, že bez informací z hlaviček bloků není možné ani bloky postupně projít; tím méně pak lze alokovat další paměť nebo uvolnit některý z přidělených bloků.

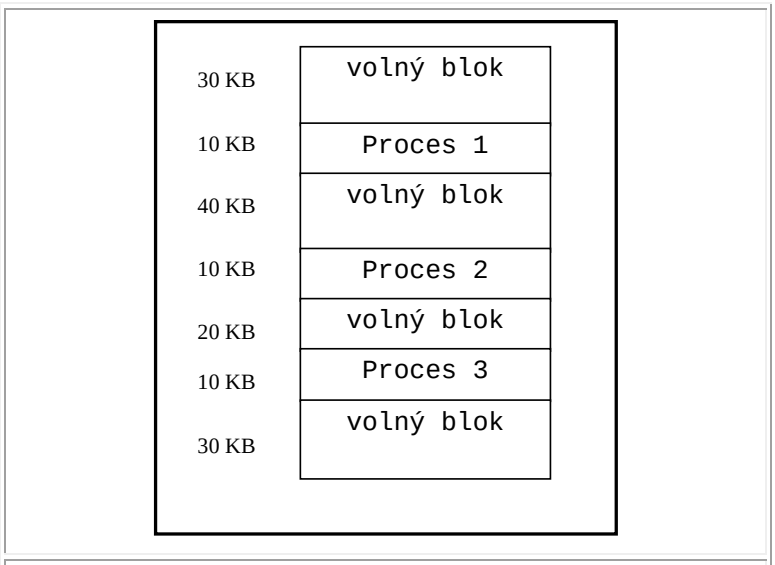
5.1.2.2 Fragmentace paměti

Systém přidělování paměti po blocích však má ještě jednu principiální nevýhodu, která se projeví i v případě, že žádný z provozovaných programů žádnou chybu neobsahuje. Dochází při něm totiž k tzv. **fragmentaci paměti.**

Již jsme se zmínili o tom, že při běhu operačního systému s přidělováním paměti po blocích se volný úsek paměti postupně rozpadá na více oddělených bloků. Správce paměti samozřejmě dokáže spojit dva volné bloky, které na sebe navazují, do jediného; to však nestačí - často dojde k situaci, kdy jsou dva volné bloky odděleny blokem přiděleným některému procesu.

Správce paměti pak pochopitelně není schopen splnit požadavek na přidělení většího množství paměti, než je velikost největšího z volných bloků, bez ohledu na to, že celková velikost volné paměti (tj. součet všech volných bloků) může být i několikanásobně větší.

Tento problém ilustruje následující obrázek, na kterém vidíme přidělené i volné bloky v operační paměti velikostí 150KB.



Fragmentace paměti

Na první pohled je zřejmé, že dokud některý ze tří procesů, které mají přidělenou paměť, svůj blok paměti neuvolní, nemůže správce paměti nikomu přidělit více než 40KB. Rozsah volné paměti je přitom daleko větší - celých 120KB.

Fragmentaci můžeme do jisté míry omezit volbou vhodné alokační strategie. Nemůžeme se jí však nikdy zbavit úplně, pokud zachováme jednoduchý systém přidělování bloků paměti.

5.1.2.3 Alokační strategie

Zamyslíme-li se nad problémem fragmentace paměti, napadne nás asi velmi jednoduchá strategie, která by dokázala fragmentaci úplně zamezit: stačilo by uvolňovat bloky v opačném pořadí, než ve kterém byly přidělovány procesům, tj. jako první vždy uvolnit ten blok, který byl alokován jako poslední. Na první pohled je samozřejmě vidět, že tato strategie není v praxi použitelná - představme si, že poslední alokovaný blok bude zapotřebí ještě dlouho, zatímco všechny předchozí by již dávno mohly být volné. Je však vhodné si uvědomit, že zásadní nedostatek této strategie spočívá v tom, že předepisovala jednotlivým procesům, jak se smí a nesmí chovat, navíc v závislosti jeden na druhém.

Takové závislosti jsou však mimořádně nešťastné. Procesy se totiž v operačním systému objevují zcela náhodně a jejich požadavky jsou také náhodné. Vneseme-li proto mezi různé procesy jakoukoli vzájemnou návaznost (a v dalších kapitolách uvidíme, že někdy se tomu bohužel nelze vyhnout), budou procesy nutně muset na sebe navzájem čekat a průchodnost operačního systému se drastickým způsobem sníží - a to si samozřejmě nepřejeme. Alokační strategie proto musí být vnitřní věcí správce paměti.

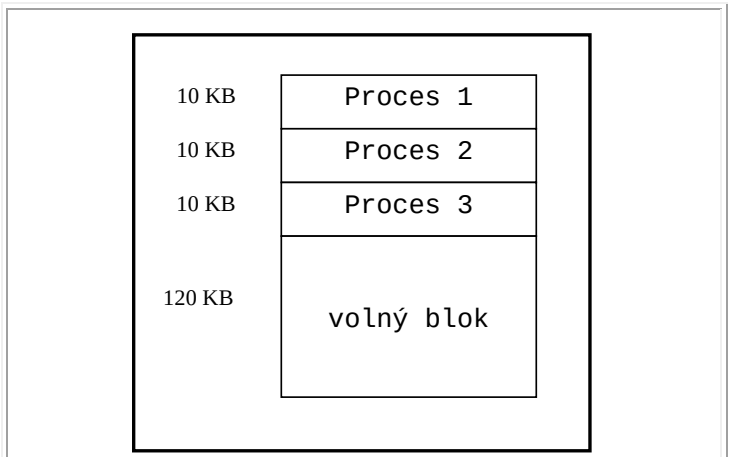
Je možné vypracovat nepřeberné množství alokačních strategií; během let se však jako životaschopné ukázaly pouze dvě: tzv. **first fit** (výběr prvního dostatečně velkého bloku), a tzv. **best fit** (výběr bloku, jehož velikost nejlépe odpovídá požadované velikostí). Strategie first fit je zcela přímočará: správce paměti prochází postupně volné bloky a porovnává jejich velikost s velikostí požadované paměti. Jakmile narazí na blok, který je dostatečně velký, požadavek uspokojí; zbytek bloku samozřejmě zůstane v seznamu jako menší volný blok. Výhodou této strategie je jednoduchost, rychlost a že nijak neomezuje fragmentaci.

Při strategii best fit naproti tomu správce paměti projde všechny volné bloky, a z těch, které jsou dostatečně velké, vyhledá nejmenší. Z něj pak již standardním způsobem uspokojí požadavek. Účelem strategie best fit je zachovat velké volné bloky co nejdéle 'nerozdrobené' a dělit především ty menší. Díky tomu se fragmentace skutečně trochu sníží. Pro úplnost dodejme, že v literatuře se často můžeme setkat i s pojmem 'strategie **last fit'.** Při této strategii se procházejí všechny volné paměťové bloky podobně jako při strategii best fit, ale požadavek se uspokojí z posledního bloku, který je dostatečně velký. Z hlediska fragmentace paměti je tato strategie dokonale rovnocenná strategii first fit, je však o něco málo složitější a o dost pomalejší. Používá se jen proto, že paměťové bloky, které jsou její pomocí alokovány,

jsou pokud možno na co nejvyšších adresách, což může být za určitých podmínek výhodné. Potřebujeme-li např. v operačním systému bez podpory multitaskingu blok pro zásobník (který obvykle roste směrem dolů), můžeme použít strategii **last fit.** Pak je velká pravděpodobnost, že mezi zásobníkem a ostatními daty zůstane nějaký úsek volné paměti a případné přetečení zásobníku ještě nemusí vést ke zhroucení programu.

5.1.2.4 Přesunování bloků

Podívejme se znovu na předchozí obrázek, který ukazuje fragmentovanou operační paměť. Na první pohled je zřejmé, že by nám pomohlo, kdybychom mohli obsazené bloky přemístit tak, že by ležely těsně vedle sebe. Volná paměť by se pak mohla spojit do jediného bloku zabírajícího celých 120KB. Obsah paměti po takovém přerovnání bloků - říkáme mu často **setřásání** - vidíme na následujícím obrázku. Správce paměti může bez problémů přidělit libovolně velký blok paměti, až do velikosti celé volné paměti, tj. do 120KB.



Setřesená paměť

Problémů s fragmentací jsme se tedy zbavili; bohužel však za to musíme zaplatit poměrně značnou cenu: správce paměti musí paměťové bloky přesunovat, což je samozřejmě časově náročné (zvláště jedná-li se o větší úseky paměti) a jednotlivé procesy musí počítat s tím, že přidělená paměť nemusí zůstat stále na stejné adrese. Jak uvidíme, jedná se o relativně složitý problém. Přesunování bloků samo o sobě samozřejmě zabere nějaký čas, to však není zase příliš tragické. Na řadě počítačů má správce paměti k dispozici specializovaný mikroprocesor (nejčastěji jej nazýváme **blitter**), který zajišťuje velmi rychlé přesuny dat v paměti, takže časová ztráta není tak velká. Ani na systémech, kde není blitter k dispozici, nehrozí vážnější problémy. Setřásání paměti je zapotřebí provádět pouze ve chvíli, kdy některý proces požaduje příliš velký blok a k tomu nedochází tak často.

Daleko horším problémem je automatické přesunování bloků z hlediska procesů. V ideálním případě by samozřejmě mělo být přesunování bloků pro procesy zcela transparentní. To však lze zajistit jen tehdy, když technické vybavení počítače umožňuje tzv. překlad adres. Máme-li však k dispozici překlad adres, můžeme implementovat virtuální paměť, která nejen zamezuje fragmentaci, ale přináší i řadu dalších výhod.

Procesy, které pracují pod operačním systémem, jehož správce paměti využívá automatického přesunování bloků, proto obvykle musí splňovat určité konvence, které zajistí, že přesun bloku paměti procesu „neuškodí“. Máme k dispozici v zásadě tři možnosti řešení tohoto problému:

1. Procesy musí pro přístup do paměti dodržet určité adresovací konvence, které zajistí přemístitelnost bloku (typicky se bude jednat o povinné bázování vhodným registrem, konkrétní řešení samozřejmě závisí na adresovacích módech použitého procesoru).
2. Procesy musí dodržovat určité konvence na vyšší úrovni - na úrovni vlastního algoritmu. Proces může být např. povinen před přístupem do paměti zjistit dotazem u správce systému momentální adresu bloku a pak po celou dobu práce s tímto blokem paměť 'zamknout' - tj. zakázat jeho přemístění.
3. Operační systém může procesu zaslat zprávu ve chvíli, kdy blok paměti přemisťuje. Proces pak na základě této zprávy přepočítá všechny své ukazatele, které do bloku míří, na správné hodnoty podle nové adresy bloku.

Každá z možností má své výhody i nevýhody.

První metoda je nepochybně nejlepší, závisí však do značné míry na technickém vybavení - u některých procesorů může velmi podstatným způsobem redukovat možnosti adresování v paměťových blocích. Musí jí být také přizpůsoben použitý překladač při tvorbě procesů ve vyšším jazyce.

Druhý způsob klade nejmenší nároky na správce paměti, o to více práce však mají programátoři, kteří vytvářejí procesy.

Třetí metoda není příliš vhodná pro běžné aplikace, dobře však poslouží jako doplněk první metody pro programy, které z nějakých důvodů musí nutně pracovat s absolutními adresami (typicky se jedná o ovladače periferních zařízení). V případě jejího využití je však nutné vhodným způsobem zvolit způsob, kterým správce paměti procesu oznámí přesunutí bloku paměti. Klasický aparát zpráv, se kterým se seznámíme později, není vhodný, protože zpráva by mohla přijít příliš pozdě (tedy po pokusu o práci s pamětí na původním místě).

Dodejme, že správce paměti může volit jednu ze dvou variant:

- bloky mohou být vyhrazovány libovolným způsobem tak, jak jsme předpokládali až dosud,
- lze alokovat bloky jednoho procesu vedle sebe, bez přerušení cizím blokem. V takovém případě budeme celou skupinu bloků jednoho procesu nazývat sekce.

Na závěr si proveďme shrnutí strategie ukládání do pevných bloků

- **First fit** - procházejí se bloky paměti a přidělí se první blok, délka je větší nebo rovna požadované paměti.
- **Last fit** - vybere se poslední vyhovující.
- **Worst fit** - vybere se největší vyhovující u přidělovaných bloků proměnné velikosti (používá se u strategie přidělování bloků paměti proměnné velikosti pro omezení fragmentace)

- **Best fit** - vybere se nejmenší vyhovující (pro přidělování bloků pevné velikosti se zpravidla používá tato strategie)

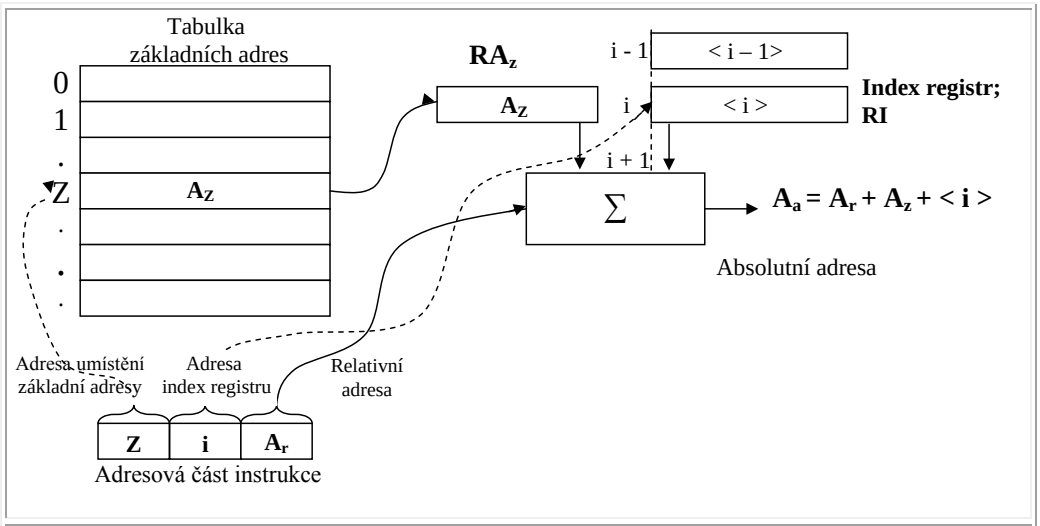
Jaké jsou výhody přidělování bloků pevné velikosti paměti:
- ve vnitřní paměti může být několik procesů současně,
- jednoduchost.

Nevýhody přidělování bloků pevné velikosti paměti:
- špatné využití paměti (fragmentace),
- je nutné předem znát paměťové nároky,
- proces jehož paměťové nároky jsou větší než velikost největšího bloku má smůlu (nelze ho odstartovat - jednomu programu není možno přidělit dva sousední bloky).

5.1.2.5 Adresování

Není možné předem stanovit na jaké adrese bude program uložen, tzn. že program musí být přemístitelný (relokatibilní). Máme tři možnosti:
- **Relokační tabulka v programu**, tj. tabulka s informacemi, kde jsou v programu adresní konstanty. Při zavádění programu se všechny adresní konstanty upraví podle skutečné adresy, na které je program.
- Program, ve kterém jsou **pouze relativní adresy**. U skoků to jsou autorelativní adresy (skočit o + nebo - N bytů). U proměnných se používá bázování, tzn. že určitý registr se naplní skutečnou adresou např. začátku programu. Pro odkazy na proměnné se používá místo pevné adresy hodnota bázového registru + posunutí (offset). Je velmi žádoucí, aby procesor podporoval bázování a musí mít relativní skoky.
- **Dynamické přidělování paměti** s využitím bázové adresy.

Na následujícím obrázku je naznačeno určení výsledné absolutní adresy na základě součtu relativní adresy, obsahu index registru a bázové adresy určené v tabulce základních adres.



Určení výsledné adresy

5.1.3 Přidělování bloků paměti proměnné velikosti

Volná paměť není pevně rozdělena, ale při startu programu se přidělí paměť podle nároků programu (resp. přidělí se celý volný blok a program vrátí, co nepotřebuje).

Výhodou při přidělování bloků proměnné velikosti oproti pevným blokům je lepší využití paměti.

Hlavní nevýhody lze charakterizovat následovně:

- Součet paměťových nároků, které jsou současně v paměti musí být menší nebo roven velikosti paměti.
- Fragmentace paměti, neboť procesy vyžadují souvislé úseky paměti. Může se stát, že není možné spustit další proces, protože má větší nároky na paměť, než je velikost největšího volného bloku i přesto, že součet velikostí volných bloků je větší než paměťové nároky procesu.

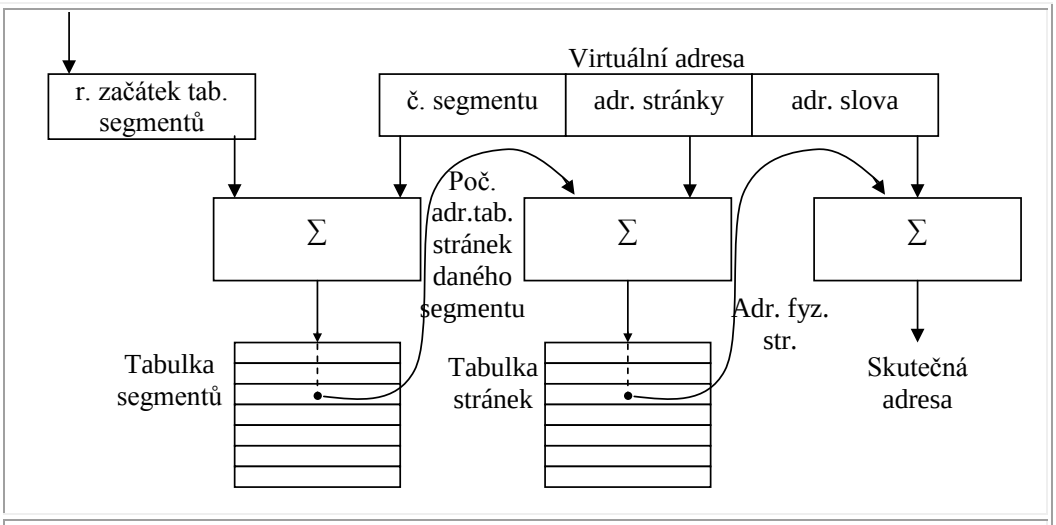
Pro odstranění fragmentace se proto používají další strategie:

- **Setřásání bloků** – zde však mohou nastat problémy s adresními konstantami, což lze odstranit použitím segmentace.
- **Stránkování** - kdy programu se jeví adresní prostor jako souvislý, přestože ve skutečnosti souvislý není.

Ochrana paměti se provádí obdobně jako u přidělování pevných bloků paměti tj. pomocí mezních registrů nebo mechanismem zámků a klíčů.

5.1.4 Segmentace paměti

Fyzická (skutečná) adresa v paměti se získává přičtením obsahu registru segmentu k logické adrese (= adresa použitá v programu). Obsah registru segmentu nastavuje operační systém a pro uživatelský program je nepřístupný. Díky tomu adresní prostor každého procesu začíná na adrese 0 a odpadají problémy s relokací programu. Většina systémů, které používají segmentaci paměti dovoluje procesům použít více segmentů.



Segmentace paměti

Rozdělení na segmenty zpravidla odpovídá struktuře paměťového prostoru procesu:

- kód programu → neměnný (obsah ani délka),
- konstanty → neměnné,
- inicializované statické proměnné → délka se nemění, obsah je nastaven při startu procesu (načte se ze souboru, který obsahuje program), při běhu procesu se jejich obsah může měnit, délka se nemění,
- neinicializované statické proměnné, délka se nemění, obsah ano,
- zásobník (návratové adresy z procedur a lokální proměnné a parametry), proměnné velikosti a obsahu, neobsahuje díry,
- halda, proměnné velikosti a obsahu, může obsahovat díry,
- paměť pro překryvné segmenty (overlaye), dynamické knihovny.

Toto rozdělení umožňuje, aby procesy, které jsou řízeny stejným programem sdílely kód programu a konstanty (úspora vnitřní paměti). U systémů se sdílenou pamětí je vhodné sdílená data uložit do zvláštního segmentu a ten zpřístupnit všem procesům, které je používají. Tím se zlepší ochrany paměti - procesy, které používají sdílenou paměť, nemají přístup k soukromým oblastem adresního prostoru jiných procesů.

Výhody segmentace paměti:

- je možné dynamické přemísťování segmentů za běhu procesu pro odstranění fragmentace (lze spojovat volné bloky paměti přesunutím překážejícího bloku),
- možnost dodatečně zvětšovat adresní prostor procesu,
- není nutné provádět relokaci programu,
- možnost sdílení segmentů.

Nevýhody segmentace paměti:

- součet nároků procesů v paměti musí být menší nebo roven velikost paměti (lze obejít odkládáním segmentů na disk, což může být časově náročné),
- nutná hardwarová podpora segmentace.

5.1.4.1 Odstranění fragmentace

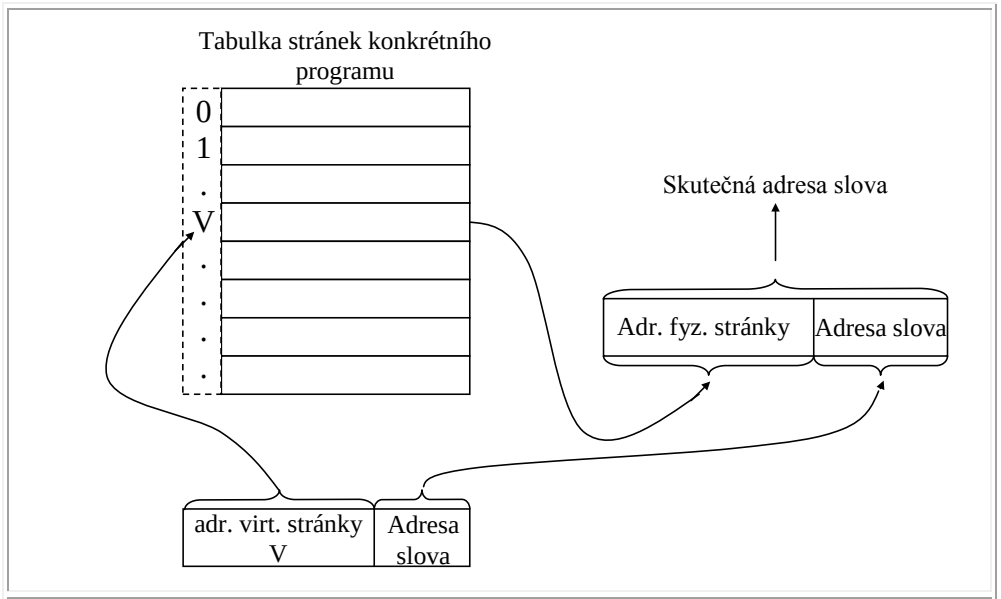
V případě, že procesy potřebují souvislý kus paměti je nutné provést setřásání segmentů. Provádí se zpravidla v okamžiku, kdy při startu nového procesu není žádný volný blok paměti dostatečné velikosti, ale součet volných bloků stačí (některé systémy v případě, že není dostatek volného místa, odkládají adresní prostory některých procesů na disk, provádějí tzv. swapping). Setřásání segmentů je realizováno překopírováním adresního prostoru některých procesů a změnou proměnné, která slouží pro naplnění registru segmentu. Existují různé algoritmy, které se snaží minimalizovat velikost kopírované paměti:

- některé procesory umožňují, aby každý proces mohl používat více segmentů,
- segmentace umožňuje, aby běžícímu procesu byla na žádost přidělena další paměť.

5.1.5 Stránkování paměti

Procesy pro svůj běh typicky požadují souvislý úsek paměti. Nutnost přidělovat souvislé úseky paměti a jejich uvolňování v libovolném pořadí podle toho, jak končí jednotlivé procesy, vede k fragmentaci paměti. Jednou z metod, jak se s fragmentací vyrovnat, je přemísťování segmentů, které však může být časově náročné. Stránkování paměti umožňuje přidělit procesu několik nesouvislých úseků paměti a vytvořit pro proces iluzi, že tato paměť souvislá je. Při stránkování paměti je fyzická paměť rozdělená na rámce - frames (někdy se nerozlišuje rámec a stránka).

Logická adresa (= adresa použitá v programu) může být rozdělena na dvě složky, číslo stránky a posunutí v rámci stránky (OFFSET). Velikost stránky bývá řádově jednotky kilobytů. Při velikosti stránky 4 KB je pro offset potřeba 12 bitů (2^{12} = 4K), čili spodních 12 bitů logické adresy je offset, zbylé bity jsou číslo stránky. Po rozkladu adresy (vše provádí procesor bez asistence programátora) na číslo stránky a offset, se číslo stránky použije jako index do tabulky stránek (každý proces má svoji vlastní). V tabulce stránek je uvedeno číslo rámce ve fyzické paměti. K číslu rámce se připojí offset a výsledkem je fyzická adresa v paměti.



Princip stránkování paměti

Výhody stránkování paměti:

- odstranění fragmentace,
- není nutné přemísťování bloků paměti.

Nevýhody stránkování paměti:

- poslední stránka procesu nebývá zcela využita a proto velikost stránek nesmí být příliš velká (tzv. vnitřní fragmentace),
- nutné HW podpora,
- součet paměťových nároků procesů v paměti nemůže překročit velikost fyzické paměti.

Ochrana paměti, tj. znemožnění použít adresy mimo adresní prostor procesu je realizována pomocí mezního registru, který obsahuje maximální číslo stránky pro příslušný proces. Procesu pak není dovoleno měnit obsah tabulky stránek.

5.1.6 Stránkování na žádost (demand paging)

Z pozorování principů přidělování paměti vyplývá, že většina procesů se chová tak, že po určitou dobu používá několik málo oblastí paměti a s průběhem procesu se tyto oblasti mění relativně pomalu (princip lokality paměťových odkazů). Díky tomu není nutné po celou dobu běhu procesu udržovat celý jeho adresní prostor v paměti. Adresování funguje podobným způsobem jako u obyčejného stránkování. V tabulce stránek je však pro každou stránku údaj, zda se stránka nachází v paměti nebo na disku. V případě, že je stránka na disku, je uvedeno i její umístění na disku. Pro stránkování se používá zpravidla zvláštní soubor, partition (oblast na disku) nebo dokonce disk.

V případě, že se proces odkazuje na stránku, která je přítomna ve fyzické paměti, vše probíhá jako u běžného stránkování. Pokud však stránka ve fyzické paměti není (je na disku), dojde k vyvolání vnitřního přerušení „výpadek stránky“. Obslužný program přerušení musí do vnitřní paměti zavést stránku z disku, opravit odkaz v tabulce stránek a zajistit zopakování instrukce, která výpadek stránky způsobila. Pokud je ve vnitřní paměti volné místo, použije se libovolný z volných rámců. Pokud jsou všechny rámce plné, je nutné vybrat některý z nich a přenést jej do stránkovacího souboru na disk. Existuje několik algoritmů nahrazování stránek nebo „výběru oběti“.
Výhody stránkování na žádost:

- odstranění fragmentace,
- není nutné přemísťování bloků paměti,
- součet paměťových nároků procesů v paměti může překročit velikost fyzické paměti.

Nevýhody stránkování na žádost:

- poslední stránka procesu nebývá zcela využita, proto velikost stránek nesmí být příliš velká (tzv. vnitřní fragmentace),
- nutná HW podpora.

Ochrana paměti je stejná jako u normálního stránkování, navíc zde musí být realizována ochrana stránkovacího souboru na disku.

5.1.6.1 Algoritmy nahrazování stránek

Zdánlivě nejjednodušší je **algoritmus FIFO**. Tento algoritmus vyhodí z paměti stránku, která je v ní nejdéle. Bohužel to může být stránka, která se používá trvale, což efektivitu algoritmu snižuje. Algoritmus FIFO také vykazuje tzv. FIFO anomálií. Pokud se provede tentýž výpočet dvakrát s různě velkou vnitřní pamětí, mělo by při výpočtu s větší pamětí dojít nejvýše ke stejnému počtu výpadků stránek jako při výpočtu s menší pamětí. Při použití strategie FIFO to nemusí vždy platit. Navíc není snadné implementovat nahrazování stránek pomocí strategie FIFO. V praxi se proto používají jiné algoritmy.

Optimální algoritmus nahrazování stránek by vyhodil z paměti tu stránku, která v budoucnosti nebude použita nejdelší dobu. Předpovídat budoucnost však není možné, proto se používají algoritmy, které pro odhad budoucího chování používají chování v minulosti. **Algoritmus LRU** (least recently used) vyhazuje z paměti tu stránku, která nebyla nejdelší dobu použita. Implementace tohoto algoritmu může používat buď registr udávající čas posledního odkazu na danou stránku nebo frontu, na jejíž začátek se zařazuje stránka, na kterou byl právě proveden odkaz. Má-li být z paměti některá stránka vyhozena, vybere se ta, která nebyla použita nejdéle (v případě použití fronty je to poslední ve frontě). Algoritmus LRU je sice kvalitní, ale jeho hardwarová implementace je obtížná. Proto se používá zjednodušení algoritmu LRU nazývané **NUR** (not used recently). V tomto případě je každému rámci přiřazen jednobitový příznak, zda byla příslušná stránka použita. Při hledání oběti se vybere ta stránka, která použita nebyla.
V algoritmech nahrazování stránek je vhodné brát v úvahu, zda byl obsah stránky změněn. V případě, že nebyl, stačí stránku pouze zahodit (její kopie je na disku). Pokud byla stránka změněna, je nutné ji nahrát na disk, což trvá přibližně stejně dlouho jako načtení nové stránky z disku. Pro tento účel bývá pro každý rámec k dispozici jednobitový příznak, který se vynuluje při zavedení stránky do operační paměti a nastaví při zápisu do rámce.

5.1.7 Segmentace se stránkováním na žádost

V případě strategie přidělování paměti segmentací se stránkováním na žádost se logická adresa určí z čísla segmentu, čísla stránky a offsetu na stránce. Každý proces má vlastní tabulku segmentů a každý segment má vlastní tabulku stránek (pro sdílený segment je tabulka jen jedna).

Výhody segmentace se stránkováním na žádost:
- odstranění fragmentace,
- je možné používat více paměti (virtuální paměť) než je velikost fyzické paměti,
- je možné sdílet segmenty.

Nevýhody segmentace se stránkováním na žádost:
- složitost,
- nutná podpora hardware.

5.2 Virtuální paměť

Za nejdokonalejší strategii správy paměti lze považovat tzv. virtuální paměť. V tomto odstavci se postupně seznámíme s nezbytnými předpoklady pro implementaci virtuální paměti i s jejím mechanismem.

5.2.1 Potřebné technické vybavení

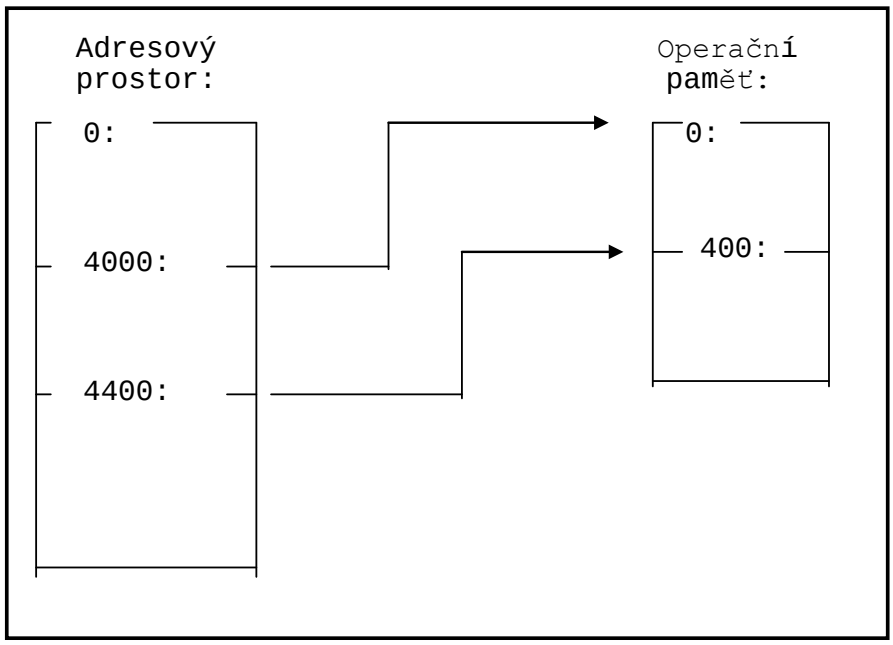
Pro implementaci virtuální paměti musíme mít k dispozici speciální technické vybavení, kterému se říká **jednotka řízení paměti** (MMU – memory management unit). Jedná se vlastně o specializovaný procesor, který stojí mezi procesorem a operační pamětí počítače a sám zpracovává požadavky procesoru na přístup k operační paměti. Navíc požadujeme určité schopnosti i po

samotném procesoru tj. musí být schopen zpracovávat přerušení a musí být schopen zopakovat přerušený přístup do paměti (nebo celou instrukci):

- **Přerušení** je signál nějakého vnějšího zařízení (v tomto případě jednotky řízení paměti), který je zapotřebí obsloužit speciálním programem ihned. Procesor tedy musí být schopen na přerušení reagovat tak, že dočasně přeruší program, který právě zpracovává a přejde na zpracování obslužné rutiny přerušení. Po jejím ukončení se vrátí k přerušenému programu.
- **Zopakování přístupu do paměti** je základem celé virtualizace. Pokusí-li se totiž mikroprocesor pracovat s pamětí, která neexistuje (je pouze virtuální, doslova zdánlivá), odpoví na to jednotka řízení paměti přerušením. Správce paměti ve spolupráci s jednotkou řízení paměti přerušení obslouží tím způsobem, že onu neexistující paměť doplní (zanedlouho uvidíme jak). Procesor pak musí být schopen zopakovat pokus o přístup do paměti - která již nyní existuje - aby mohl přerušený program pokračovat v práci.

Jednotka řízení paměti musí být schopna minimálně zajistit ochranu paměti, jinak není možné virtuální paměť vůbec implementovat. Velmi důležitou funkcí jednotky řízení paměti je i překlad adres - bez něj by byl systém virtuální paměti nesmírně těžkopádný a v praxi nepoužitelný. Pro rozumnou efektivitu virtuální paměti však obvykle požadujeme po jednotce řízení paměti ještě jednu službu - stránkování. Správce paměti předá jednotce řízení paměti vhodným způsobem informace ze svých tabulek - tj. který úsek paměti patří kterému procesu. Správce procesů zajistí, aby správce paměti věděl, který proces právě běží; správce paměti předá i tuto informaci jednotce řízení paměti. Při zpracování každého požadavku na přístup k operační paměti pak jednotka řízení paměti ověří, má-li aktivní proces právo s tímto úsekem paměti pracovat. Jestliže tomu tak není, jednotka řízení paměti procesoru přístup k paměti neumožní a namísto toho vyvolá výjimku. Správce paměti, který výjimku obsluhuje, pak může zajistit vše potřebné.

Překlad adres umožňuje (přibližně řečeno) přiřadit libovolnému úseku operační paměti libovolné adresy. Správce paměti může vytvořit tabulky, které určují nejen komu který blok paměti patří a jak je velký, ale i na které adrese v operační paměti má ležet. Překlad adres nám umožní používat celý adresový prostor procesoru s tím, že kterékoliv jeho části můžeme přiřadit skutečnou operační paměť. Můžeme mít tedy například blok velikostí 1KB, který pro program leží na adrese 4000h - program jej může naprosto běžným způsobem používat. Na adrese 4002h nalezne třetí byte bloku. Správce paměti však pomocí tabulek oznámil jednotce řízení paměti, že tento blok má v operační paměti ležet na adrese 0. Jednotka řízení paměti pak bude pracovat tak, že kdykoli zachytí požadavek na přístup k některé adrese v rozmezí 4000h až 43FFh, předá jej operační paměti, ale nejprve od adresy odečte hodnotu 4000h. Tuto situaci ilustruje obrázek.



Překlad adres

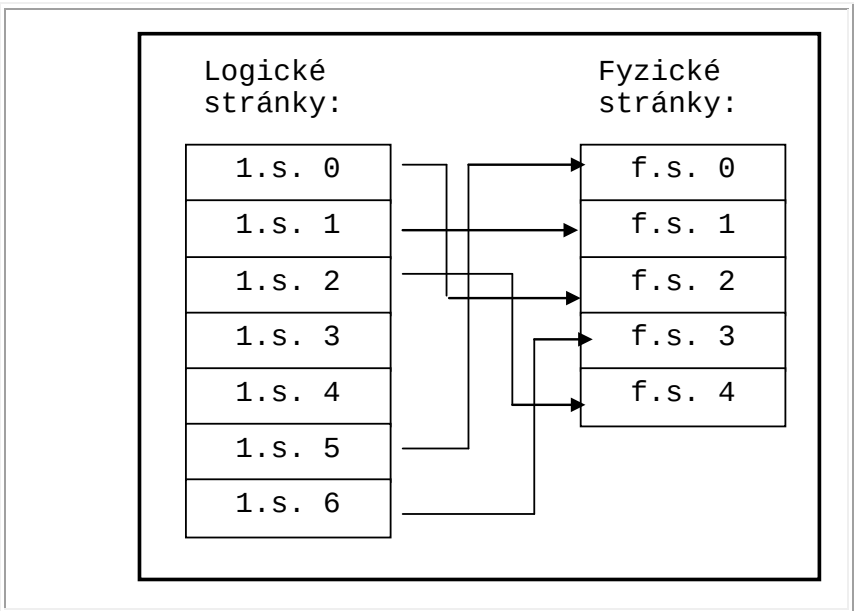
Abychom mohli dobře rozlišit obě adresy, které se účastní překladu adres, budeme adrese, kterou používá program a která leží v adresovém prostoru procesoru říkat **logická adresa.** Adrese v operační paměti, po překladu adres, budeme naproti tomu říkat **adresa fyzická.**

V minulém příkladu se tedy logická adresa 4000h přeložila na fyzickou adresu 0h, logická adresa 4002h na fyzickou adresu 2h a podobně.

Stránkování na první pohled vypadá jako omezení, protože požaduje, aby bloky, které se účastní překladu adres, byly složeny z tzv. stránek pevné velikosti. Adresový prostor procesoru se tak rozpadá na řadu stránek - první z nich začíná na (logické) adrese 0, druhá na adrese <velikost stránky>, třetí na adrese <2*velikost stránky> a tak dále. Těmto stránkám budeme říkat **logické stránky**, podobně jako adresy v tomto prostoru nazýváme logickými adresami. Obdobným způsobem rozdělíme na jednotlivé stránky operační paměť - první z nich bude začínat na (fyzické) adrese 0, druhá na adrese <velikost stránky>, třetí na adrese < 2*velikost stránky> a tak dále. Těmto stránkám budeme říkat **fyzické stránky**. Tabulky pro překlad adres pak mohou mít poměrně jednoduchou strukturu. Každé logické stránce bude odpovídat jedna položka v tabulce. V této položce bude uvedeno číslo fyzické stránky, která má logické stránce odpovídat (nebo informace o tom, že tato logická stránka žádnou fyzickou stránku přidělenou nemá). Při velikosti stránky např. 4KB tedy nemůžeme zajistit překlad logické adresy 10h na fyzickou adresu 4000h, protože nejnižších dvanáct bitů logické i odpovídající fyzické adresy musí být totožných. Systém překladu adres se díky tomu výrazně zjednoduší.

Příklad vzájemného přiřazení stránek vidíme na následujícím obrázku. První logické stránce je přiřazena fyzická stránka číslo 2, logická stránka číslo 3

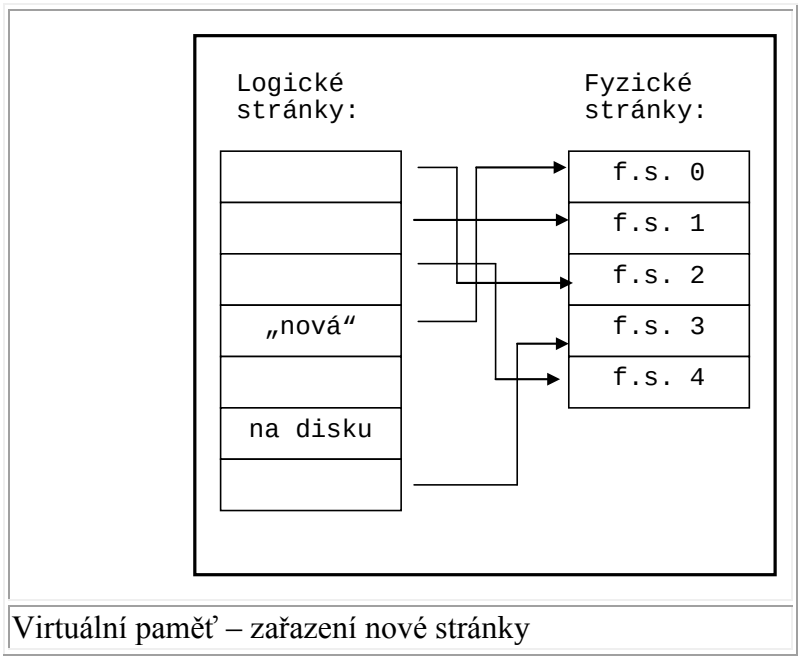
nemá vůbec žádnou fyzickou stránku přiřazenu. Připomeňme si ještě jednou, že fyzické stránky reprezentují paměť, zatímco logické pouze adresní prostor.



Virtuální paměť – příklad počátečního nastavení stránek

Základní princip virtuální paměti je poměrně jednoduchý. Správce paměti má k dispozici odkládací prostor na nějakém rychlém velkokapacitním paměťovém médiu, zpravidla na pevném disku. Dojde-li k situaci, že správce potřebuje někomu přidělit paměť a přitom nemá k dispozici žádný volný blok, odebere prostě paměť některému z procesů. Ve stránkových tabulkách, které odpovídají logickým stránkám tohoto procesu vyznačí, že nemají přiřazené žádné fyzické stránky. Obsah fyzických stránek, které procesu až dosud patřily, uloží do odkládacího prostoru a využije překlad adres k tomu, aby mohl stránky přidělit novému žadateli.

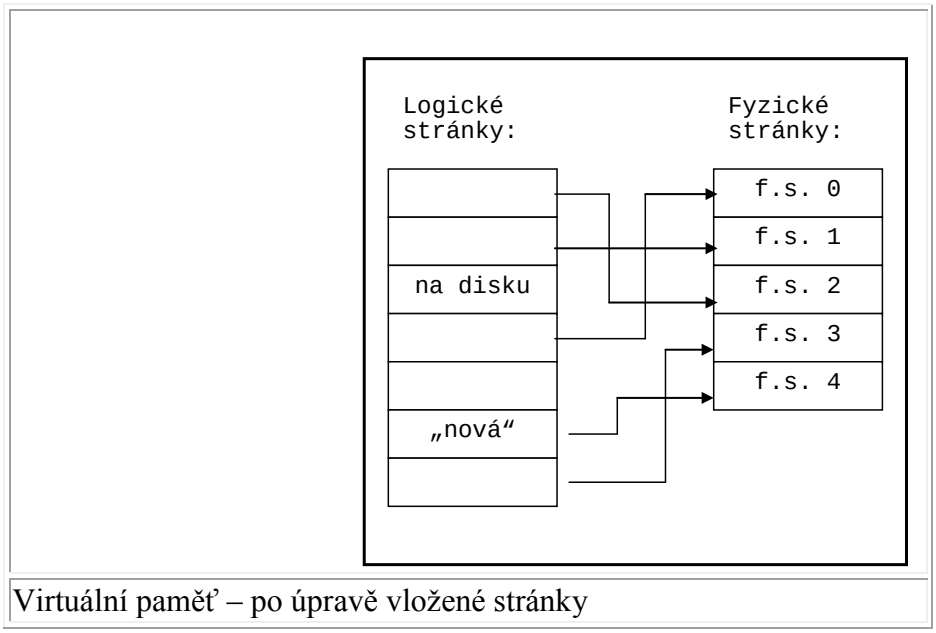
Ilustrujme si tento případ na předchozím obrázku**.** Předpokládejme, že logická stránka číslo 3 je dosud volná a nějaký proces požaduje přidělení paměti. Pro jednoduchost řekněme právě v rozsahu jedné stránky. Správce paměti přidělí logickou stránku číslo 3 a hledá odpovídající fyzickou stránku. Zjistí však, že jsou všechny obsazeny. Zapíše proto obsah fyzické stránky číslo 0 do odkládacího prostoru a pozmění potřebným způsobem stránkové tabulky. Výsledek vidíme na následujícím obrázku.



Virtuální paměť – zařazení nové stránky

Dojde-li později k tomu, že se původní vlastník pokusí pracovat se svou pamětí (kterou již vlastně nemá), detekuje jednotka řízení paměti, že požadovaná adresa neexistuje - této situaci budeme říkat výpadek stránky - a vyvolá výjimku, kterou obslouží správce paměti. Ten reaguje stejně, jako kdyby si proces vyžádal přidělení nové paměti. Nalezne pro něj nějaké volné fyzické stránky (nejsou-li volné, odebere je již známým způsobem někomu jinému), využije překlad adres k tomu, aby tyto stránky připojil k logickým stránkám procesu a zapíše do nich údaje z odkládacího prostoru. Proces pak může pokračovat stejným způsobem, jako by se s jeho pamětí nikdy nic nedělo.

Vraťme se opět k našemu příkladu. Původní vlastník fyzické stránky číslo 0 (pracoval s ní pochopitelně prostřednictvím logických adres z logické stránky číslo 5) se pokusí o přístup ke své paměti. Jednotka řízení paměti to samozřejmě detekuje, a protože žádná taková paměť není k dispozici (momentálně platí situace z předchozího obrázku, kde logická stránka 5 žádnou fyzickou stránku nemá připojenou), vyvolá výjimku. Výjimku obsluhuje správce paměti. Ten se pokusí vyhledat nějakou volnou fyzickou stránku. Nenalezne ji, a proto musí někomu jednu odebrat, zvolí např. fyzickou stránku číslo 4. Nejprve její obsah uloží do odkládací oblasti, potom do ní z odkládací oblasti přečte údaje, které byly ještě nedávno uloženy ve fyzické stránce číslo 0. Pak upraví stránkové tabulky tak, jak ukazuje následující obrázek.



Virtuální paměť – po úpravě vložené stránky

Poznamenejme, že této druhé části, kdy je stránka přidělována na základě pokusu o použití adresy uvnitř logické stránky, říkáme **stránkování na žádost**.

Na virtuální paměti tedy není v zásadě nic složitého. Musíme však ještě vyřešit řadu technických detailů. Správce paměti musí mít k dispozici nějaký klíč, podle kterého bude volit stránku, kterou má odebrat. Musí také rozhodnout, kam na odkládací prostor zapsat její obsah a musí toto místo rychle najít při obnovování stránky z disku.

V praxi je třeba zpochybnit i představu, kterou jsme v tomto odstavci rozvinuli - že totiž správce paměti reaguje na požadavky na přidělení bloku prací se stránkami. Kromě řady jiných nevýhod by to znamenalo, že velikost přidělovaných bloků paměti by musela být dělitelná velikostí stránky. To by ale bylo velmi nepraktické. V následujících odstavcích se proto na implementaci virtuální paměti podíváme podrobněji.

5.2.2 LRU a pseudo-LRU

Nejprve se podíváme na princip výběru stránky, kterou budeme uvolňovat v případě, že je zapotřebí vyhradit nějakou další paměť a všechny fyzické stránky již jsou přiřazeny. Zamyslíme-li se nad problémem z hlediska efektivity celého operačního systému, nalezneme brzy správnou odpověď na otázku „kterou stránku odebrat“. Přece tu, kterou už nebude nikdo potřebovat, a jestliže taková neexistuje, tak tu, kterou bude někdo potřebovat co nejpozději. Odpověď je to sice nepochybně správná, ale bohužel nepoužitelná, protože neexistuje způsob, jak tento údaj o stránkách zjistit.

Rozsáhlé statistiky používání stránek v řadě operačních systémů však ukázaly, že bychom se mohli přiblížit optimu, kdybychom vybírali vždy tu stránku, kterou po nejdelší dobu nikdo nepoužil. V průměru je totiž daleko pravděpodobnější, že stránka, se kterou až dosud každou chvíli někdo pracoval, bude stejně intenzívně využívána i nadále a naopak, než že by došlo k výrazné změně ve využití jednotlivých stránek. Algoritmus, který vybere právě tu

stránku, kterou co nejdelší dobu nikdo nepoužil, se jmenuje **LRU** (least recently used), čili nejdéle nepoužitá.

V praxi však bohužel nemůžeme využít ani algoritmus LRU - představme si, jak bychom jej asi realizovali. Zda některá stránka byla, nebo nebyla použita neví nikdo jiný než jednotka řízení paměti. Ta by musela při každém použití některé stránky - tedy vlastně při každém přístupu do paměti - generovat přerušení. Obslužná rutina přerušení by musela nejprve tento mechanismus vypnout (aby nebyla sama neustále přerušována) a pak by přemístila právě použitou stránku na konec fronty stránek, připravených k odebrání. Jinou alternativou je, že by obslužná rutina přerušení uložila někam do stránkových tabulek čas, kdy byla stránka naposledy použita a při odebírání stránky by se pak prohledaly všechny stránky a zvolila by se ta s nejstarším časem. Na první pohled je zřejmé, že počítač by neměl čas téměř na nic jiného než na stránkování. Volíme proto algoritmy, které jsou jednodušší, méně zatěžují počítač a jejich výsledky se přitom blíží výsledkům algoritmu LRU. Takovým algoritmům potom říkáme **pseudo-LRU** algoritmy.

Nejčastěji používaný pseudo-LRU algoritmus pracuje následujícím způsobem. Součástí stránkových tabulek je tzv. **used** bit - speciální bit, který jednotka řízení paměti nastaví na jedničku při každém přístupu k odpovídající logické stránce (a tedy také k její přiřazené stránce fyzické). To lze snadno zajistit bez jakékoli degradace výpočetního výkonu. Operační systém pravidelně v určitém intervalu prochází stránkové tabulky a nuluje všechny **used** bity. Nalezne-li již některý **used** bit nulový, znamená to, že tato stránka nebyla po celý interval použita (jinak by jednotka řízení paměti její **used** bit nastavila), a je tedy dobrým kandidátem na odebrání.

Tento algoritmus má poměrně velmi dobré výsledky. Poznamenejme bez detailního výkladu, že jej lze mírně vylepšit využitím tzv. strategie sdružování dvojic - při výběru stránky k odebrání se díváme na dvojici po sobě následujících stránek jako na jednu velkou stránku.

5.2.3 Virtualizace paměti

Všechny stránky mají stejnou velikost, jsou tedy jako stvořeny k ukládání na disk do rychle zpracovatelné struktury s pevnou velikostí záznamu a s přímým přístupem. Nejjednodušší implementace virtuální paměti pak může vyhradit na disku místo třeba pro celý adresový prostor procesoru a skutečnou operační paměť využívat vlastně jen jako cache.
Program potom může použít kteroukoli adresu z celého adresového prostoru. Jestliže adresuje logickou stránku, která nemá svůj ekvivalent v operační paměti, přidělí mu operační systém nějakou fyzickou stránku (buď volnou, nebo ji někomu odebere způsobem, se kterým jsme se již seznámili).

Komplexnější operační systémy však v praxi většinou postupují trochu lepším způsobem:

- Na začátku práce má program k dispozici celý adresový prostor, ale žádnou fyzickou paměť (všechny logické stránky tedy mají nastaven přepínač, určující přiřazení fyzické stránky, na „neexistenci“).

- Kdykoli se pokusí program použít nějakou adresu v logické stránce, která nemá a ani dosud neměla svůj ekvivalent v operační paměti (ze začátku tedy pokusí-li se program použít jakoukoli adresu), dojde k výpadku stránky a jednotka řízení paměti samozřejmě vyvolá přerušení. V něm se operační systém pokusí vyhledat nějakou dosud nevyužitou fyzickou stránku a přidělit ji programu. Pokud se mu to podaří, ukončí ihned přerušení a umožní tak programu pokračovat v přerušené práci - požadovaná adresa již ovšem má svůj ekvivalent v operační paměti.
- Ve chvíli, kdy operační systém nemá žádnou volnou fyzickou stránku, kterou by mohl programu přidělit, musí nějakou stránku uvolnit. K výběru fyzické stránky, která má být uvolněna, nejspíše využije strategii LRU. Původní obsah této fyzické stránky je zapotřebí zapsat do odkládací oblasti; tentokrát však logická adresa neurčuje místo na disku. Řešením je alokace vyrovnávací paměti na disku, s tím, že obsah fyzické stránky se uloží na disk (tam, kde je zrovna místo). Stránka se „staré“ logické stránce odebere a ke „staré” logické stránce se v tabulce stránek zaznamená, kde přesně na disku je obsah stránky uložen. Pak již nic nebrání tomu, aby byla uvolněná fyzická stránka přidělena „nové“ logické stránce. Jestliže potřebuje program použít data v logické stránce, která již dříve byla v operační paměti, ale fyzická stránka jí byla během času odebrána, proběhnou v podstatě opět předchozí dva body (nebo jen první z nich, je-li k dispozici nějaká volná fyzická stránka). Jedinou změnou je to, že obsah přidělené stránky se před návratem do přerušeného programu inicializuje daty z disku, jejichž adresa byla uložena v tabulce stránek, když se fyzická stránka logické stránce odebírala podle minulého bodu. Nakonec se ještě může uvolnit místo na disku, kde byl uložen obsah stránky.

Je vidět, že při této strategii není na disku obsazeno více místa, než je nezbytně třeba. Přitom je tato strategie prakticky stejně efektivní, jako výše navržená strategie udržování obrazu celého adresového prostoru na disku.

Systém stránkování tedy umožňuje, aby programy pracovaly s „dojmem“, že mají k dispozici daleko více operační paměti, než kolik jí je doopravdy v systému instalováno. Tato paměť je ovšem jen zdánlivá - **virtuální**. Proto tento způsob rozšíření paměti nazýváme **virtualizací paměti**.

Výhody, plynoucí z virtualizace paměti, jsou zřejmé. Nejsou žádné problémy s rozsahem dat, programátor může celkem pokojně pracovat s obrovskými poli dat. Odpadají také jakékoli problémy s překryvy (overlay) - program může být prakticky libovolně velký. Počítač přitom nemusí mít příliš velkou operační paměť. Je třeba si však uvědomit, že méně paměti sníží efektivitu systému (je nutné častěji „přehazovat“ stránky mezi diskem a operační pamětí).

5.2.4 Implementace správce paměti

Nejjednodušší a poměrně efektivní implementace virtuální paměti využívá správce paměti s přidělováním bloků, jak jsme jej poznali v předchozích odstavcích - tedy nejjednodušší algoritmus vůbec. Nepracuje však nad reálnou

operační pamětí, jako tomu bylo předtím, ale nad celým adresovým prostorem procesoru (nebo nad jeho částí, je-li rozsah virtuální paměti omezen).

O pokrytí adresového prostoru operační paměti se stará druhá úroveň správce paměti, která s první úrovní vůbec žádným způsobem nesouvisí. Správce paměti se v takovém případě vlastně rozpadá na dvě zcela samostatné části:

- správce virtuální paměti,
- vlastní správce paměti, který přiděluje procesům požadované bloky.

Na úrovni vlastního správce paměti dochází k fragmentaci. Fragmentován je však pouze adresový prostor, nikoli skutečná paměť, protože logickým stránkám v dírách fragmentovaného adresového prostoru samozřejmě nejsou přiděleny fyzické stránky. Vzhledem k rozsahu adresového prostoru (u běžných procesorů se jedná řádově o gigabyty) nám jeho fragmentace nevadí, protože i když fragmentace zabere hodně prostoru, zůstane vždy dostatečně velký blok ještě volného adresového prostoru. Na tomto principu je možné doplnit správce virtuální paměti i k operačnímu systému, který sám o sobě jednotku řízení paměti nevyužívá. Správce virtuální paměti, doplněný takovýmto způsobem k operačnímu systému zvenku, je samozřejmě lepší než žádný; toto řešení však má řadu nevýhod. Operační systém totiž může ve spolupráci se správcem virtuální paměti zajistit řadu velmi šikovných služeb - jmenujme některé z nich:

- Služeb správce virtuální paměti může využívat systém pro práci se soubory pro rychlý a efektivní přístup k datům na disku. Uvědomme si, že stačí vytvořit umělé stránkové tabulky obsahující odkazy na sektory disku, ve kterých je uložen soubor, a můžeme pohodlně pracovat s operační pamětí - ve skutečnosti však budeme pracovat se zvoleným souborem.
- Správce paměti může automaticky zvětšovat paměťové bloky, jestliže jejich uživatelé překročí hranice bloků. To je mimořádně výhodné pro bloky obsahující zásobník.
- Ovladače periferních zařízení mohou také využívat virtuální paměť s tím, že po dobu přenosu dat mezi pamětí a periferií odpovídající stránku zamknou (tj. upozorní správce virtuální paměti, že jim stránku nesmí odebrat).
- Správce paměti může být implementován odlišným (a daleko jednodušším) způsobem – viz následující odstavec.

5.2.5 Virtualizace adres

Jednotku řízení paměti můžeme využít ještě daleko lepším způsobem než pro samotnou virtualizaci paměti. Představme si, že systém stránkování, popsaný v minulých odstavcích, používáme ve víceúlohovém operačním systému. V tomto případě máme jedno nepříjemné omezení. Ačkoli nám stránkování umožnilo (virtuálně) rozšířit operační paměť na celý adresový prostor, musí se v něm pořád vedle sebe nějak srovnat všechny procesy.

Operační systém se přitom musí starat o spoustu věcí - relokace zaváděných programů, poměrně složité spojování se sdílenými knihovnami a podobně. Jednodušší by bylo, kdyby měl každý proces vlastní adresový prostor.

Řešení je přitom jednoduché. Stačí, když bude mít každý proces vlastní tabulku stránek. Spojíme-li tuto metodu se stránkováním na žádost, nemusí se vlastní správce paměti téměř o nic starat, všechnu práci zajistí správce virtuální paměti. Fyzické stránky se přidělují tomu, kdo je právě potřebuje a při odebírání se ukládají do bufferů na disku. Protože neexistuje pevné přiřazení mezi adresami na disku a logickými stránkami (vytváří se dynamicky až při práci systému), nemá správce virtuální paměti o nic více práce, než když zajišťoval stránkování na žádost pro jediný program (respektive pro celý operační systém jako pro jediný program). Po upozornění správce procesů, že se jiný proces stává aktivním, musí pouze zajistit výměnu tabulek popisujících stránky.

Tento jednoduchý trik zajistí, že každý proces může používat vlastní adresový prostor, který se nijak nekříží (nebo kříží, ale přesně tak, jak procesy požadují) s adresovým prostorem ostatních procesů. Proto pro něj používáme termín virtualizace adres.

Kontrolní otázky:

1. Vyjmenujte různé strategie přidělování paměti?
2. Jaký je rozdíl mezi segmentací a stránkováním paměti?
3. Jakým způsobem se překládají adresy ve virtuální paměti?
4. Jaký je rozdíl mezi ochranou paměti pomocí mezních registrů a mechanismem zámků a klíčů?

Úkoly k zamyšlení:

1. Zamyslete se nad výhodami a nevýhodami řešení potřeby extrémně velké operační paměti formou virtualizace na pomalém disku. Kdy je výhodné a kdy nevýhodné plně využívat operační paměť a kdy je výhodné co nejvíce programů ukládat na disk?

Korespondenční úkol:

1. Předpokládejte, že:
    a. rozsah operační paměti je 128 MB,
    b. průměrná délka úlohy je 512 KB
    c. průměrná doba průběhu úlohy je 1 s
    d. rychlost zhušťování (defragmentace) je 10 MB/s
    e. režie zhušťování je 1 ms
    f. při použití algoritmu přidělování sekcí zůstává třetina operační paměti nevyužita
    g. při přemísťování sekcí zůstává polovina operační paměti nevyužita

    Vyplatí se provádět zhušťování? Ukažte proč, resp. jaká část paměti zůstane nevyužita..

Shrnutí obsahu kapitoly

V této kapitole jste se seznámili s principy a metodami přidělování operační pamět. Důraz v této kapitole byl kladen na pochopení principů virtuálních pamětí a metod adresace

6 Správa vstupních a výstupních zařízení

V této kapitole se dozvíte:

- Jakými způsoby komunikují vnější zařízení s procesorem?

Po jejím prostudování byste měli být schopni:

- Charakterizovat způsoby komunikace V/V zařízení s procesorem.
- Znát vytváření ovladačů periferních zařízení.
- Porozumět rozdílům v komunikaci prostřednictvím přerušení a DMA.

Klíčová slova této kapitoly:

Přerušení, DMA, kanál, ovladač.

Doba potřebná ke studiu: 3 hodiny

Průvodce studiem

Studium této kapitoly vyžaduje znalosti základů architektury počítačů a detailně vám objasní funkce ovladače periferního zařízení. .
Na studium této části si vyhraďte 3 hodiny. Po celkovém prostudování a vyřešení všech příkladů doporučujeme vypracovat korespondenční úkol.

Základní jednotka počítače komunikuje s okolním světem prostřednictvím periferních zařízení. Těchto zařízení tedy samozřejmě musí využívat i jednotlivé procesy. V moderním víceúlohovém operačním systému není možné nechat procesy, aby si mezi periferiemi „dělaly, co je napadne“. Operační systém musí obsahovat poměrně komplikovanou správu zařízení. Z hlediska množství přenášených dat rozdělujeme periferní, vstupní a výstupní (dále označujeme I/O) zařízení na:

- **znaková**, zde patří klávesnice, znakové displeje a terminály, tiskárny, myši, plottery, tablety apod.,
- **bloková**, zde se řadí pevné disky, CD/DVD ROM, magnetické pásky apod.

Některá zařízení do tohoto dělení nezapadají. Jsou to zařízení tzv. paměťově mapované (grafické displeje, speciální časovače apod.).

Rozhraní vstupních a výstupních zařízení poskytované operačním systémem by mělo být jednotné pro všechna zařízení do takové míry, jak je to jenom možné.

6.1 Prostředky komunikace

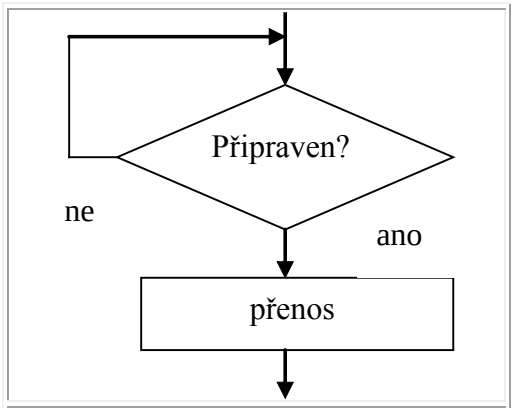
Vstupní a výstupní zařízení bez ohledu na způsob připojení používají čtyři základní techniky řízení přenosu:

- programové řízení vstupu a výstupu,
- řízení na základě přerušení,
- přímý přístup k operační paměti (DMA – Direct Memory Access),
- vstup a výstup pomocí specializovaného procesoru.

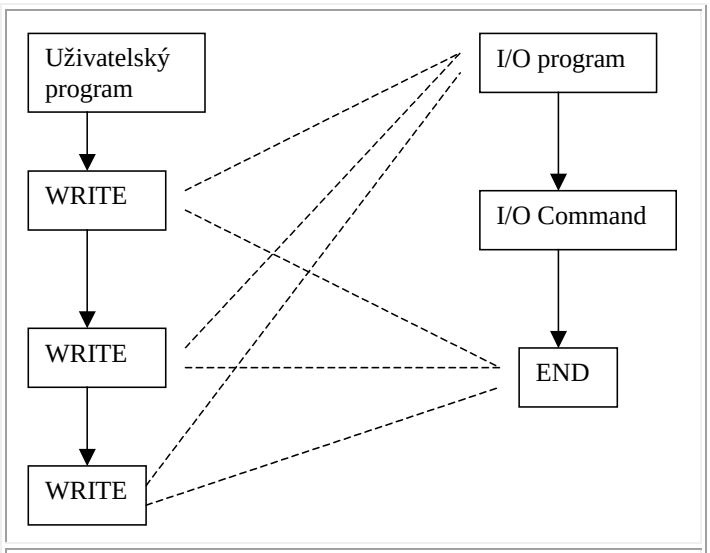
Nyní si uvedeme podrobnější analýzu jednotlivých druhů komunikací.

6.1.1 Programová obsluha

Při tomto způsobu řízení V/V operací určuje vždy procesor na základě programu okamžiky přenosu údajů **do** nebo **z** periferního zařízení. Synchronizace přenosu se pak děje jednoduchým testováním připravenosti zařízení na přenos dat, jak je uvedeno na následujících obrázcích.

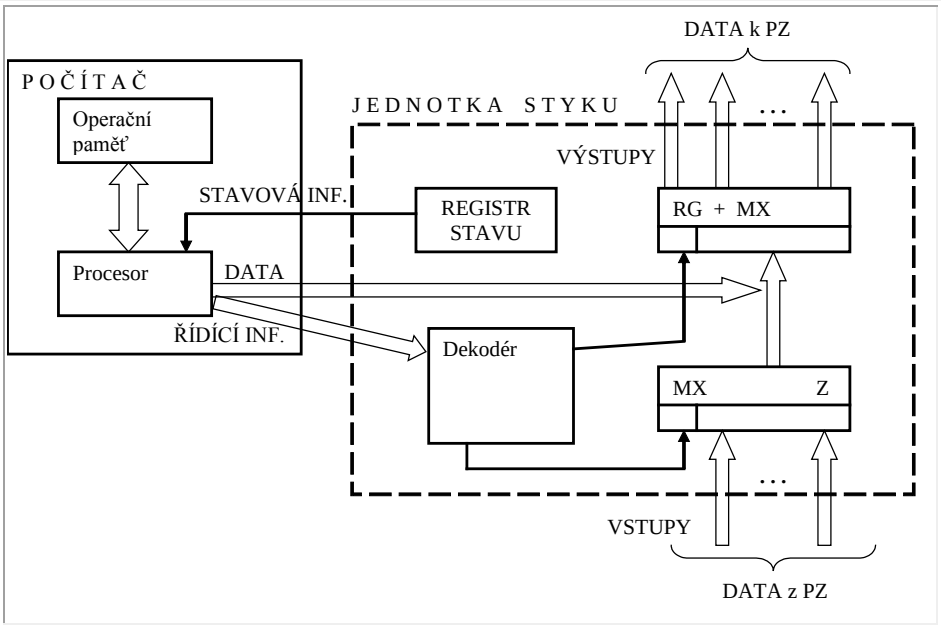


Princip podmíněného přenosu



I/O operace s činným čekáním

Tato technika je nejméně náročná na technické vybavení. Hlavní nevýhodou je to, že procesor je značně zatížen neproduktivní činností. Technické vybavení jednotky pro připojení periferního zařízení zahrnuje registr stavu, obsahující informaci o připravenost na vstup/výstup dat a multiplexor, směřující data k adresovanému perifernímu zařízení. Blokové schéma technických prostředků programového V/V je uvedeno na následujícím obrázku.



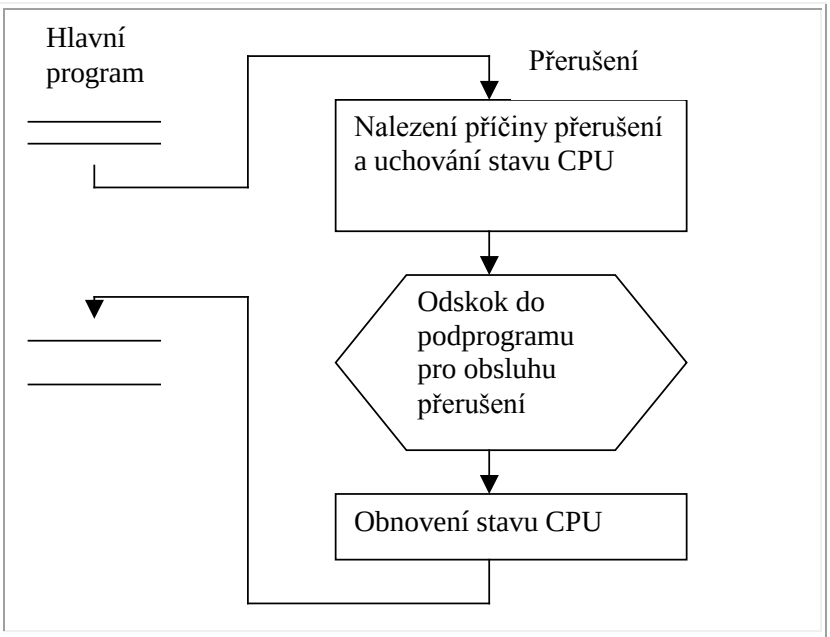
Programově řízená periferie

Vstupní a výstupní řadiče jsou realizovány jako:

- **paměťově mapované**: registry jsou adresovány jako paměť, přístupné pomocí běžných operací čtení a zápisu do paměti,
- **izolované**: registry jsou přístupné pomocí speciálních instrukcí (zpravidla nazývaných IN a OUT); díky tomu jsou adresní prostory paměti a vstupů/výstupů oddělené.

6.1.2 Přerušení

Cílem zavedení přerušení (interrupt) při řízení I/O přenosů dat je zlepšení účinnosti těchto operací. Přerušující událost způsobí, že se potlačí provádění běžícího procesu v CPU takovým způsobem, aby ho bylo možné později obnovit. V době řešení I/O operace se umožní, aby CPU prováděla jiné instrukce než periferní. Původně se tento mechanismus používal jen pro vyžádání pozornosti procesoru. Při vyvolání přerušení procesor začne provádět podprogram obsluhy přerušení (interrupt service routine - ISR) podobným způsobem, jako by byl vyvolán normální podprogram. Podprogram musí uchovat stav procesoru, pak provede vlastní obsluhu přerušení (například zašle znak nebo blok znaků na výstupní zařízení) a nakonec obnoví stav procesoru, aby přerušený program nic nepoznal (až na zpoždění). Podobá se vyvolání podprogramu, ale provádí se speciální instrukcí. Vývojový diagram obsluhy přerušení je uveden na následujícím obrázku.

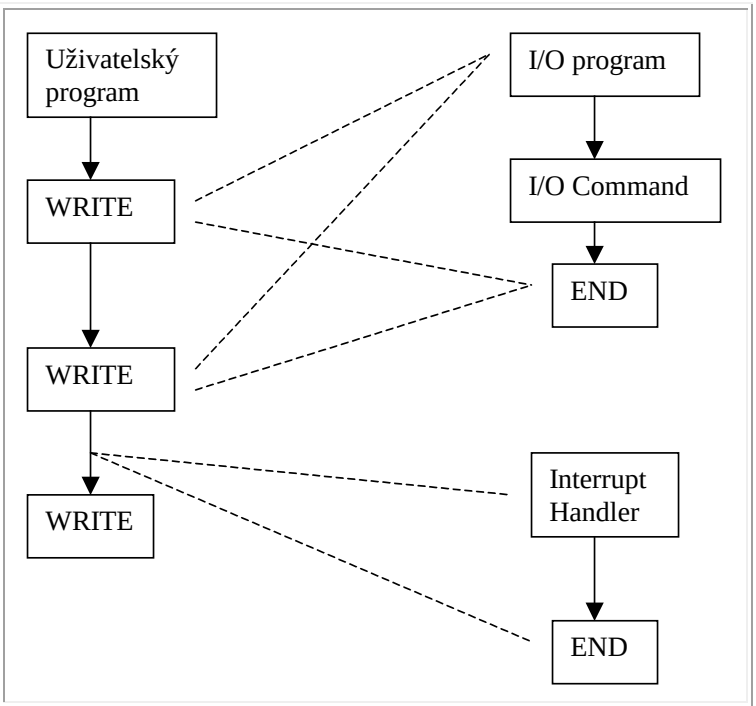


Přenos dat systémem přerušení

Posloupnost obsluhy přerušení je vyjádřena následujícími úkony:

- uchování stavu procesoru,
- vlastní obsluha přerušení,
- obnovení stavu procesoru.

Základní rozdíl mezi programově řízenou komunikací a komunikací prostřednictvím přerušovacího systému je v synchronizaci přenosu dat. Při programovém řízení se při provádění operace čtení/zápisu dat musí čekat na potvrzení připravenosti periferního zařízení a teprve pak se provede přenos, přičemž procesor v době čekání neprovádí žádné následné instrukce. Při obsluze zařízení pomocí přerušení procesor pokračuje v provádění operací a je obeznámen o připraveném periferním zařízení pomocí přerušení. Tento princip komunikace je zřejmý z následujícího obrázku.

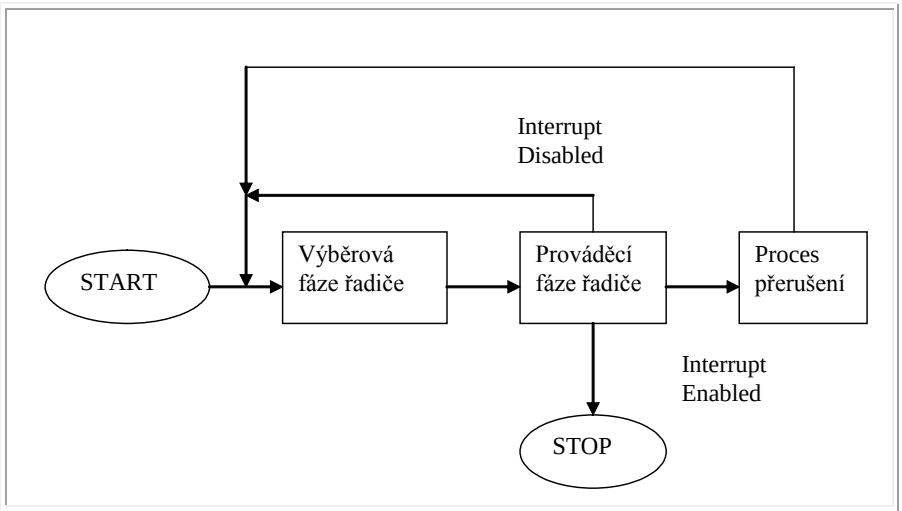


I/O operace s přerušením

Je zřejmé, že principy přerušení se tak nemusí používat jen pro operace vstupu/výstupu dat, ale i pro jiné typy synchronizace procesů. Proto přerušení dělíme na:

- **Vnější** - zdrojem jsou řadiče (zejména I/O zařízení) umístěné „vně procesoru“. K přerušení dochází bez ohledu na právě prováděné místo v programu a ISR je vyvolán po dokončení instrukce. Reakci na přerušení lze dočasně zakázat (maskovat), v tom případě pak k obsluze přerušení dojde po povolení přerušení. Po návratu z ISR přerušený program pokračuje další instrukcí.
- **Vnitřní** - přerušení je vyvoláno chybou při provádění strojové instrukce (dělení nulou, přetečení, porušení ochrany paměti, výpadek stránky). ISR může vypsat chybové hlášení a ukončit program, dosadit náhradní výsledek v případě aritmetické chyby, zavést stránku do vnitřní paměti z disku apod. Při některých chybách je možné zopakovat instrukci, která chybu způsobila.
- **Programové** - přerušení je vyvoláno instrukcí volání přerušení umístěnou přímo v programu. Používá se pro volání služeb operačního systému. Výhoda oproti volání podprogramů: není možné vyvolávat podprogramy na libovolných adresách (pouze adresy uvedené v tabulce přerušení).

Vzhledem k tomu, že synchronizace činnosti procesoru je dána návazností výběrové a prováděcí fáze cyklu řadiče je nutné asynchronní požadavky přerušení začlenit do tohoto synchronního procesu a to dle následujícího obrázku. Zákaz nebo povolení přerušení vyvolává procesor.



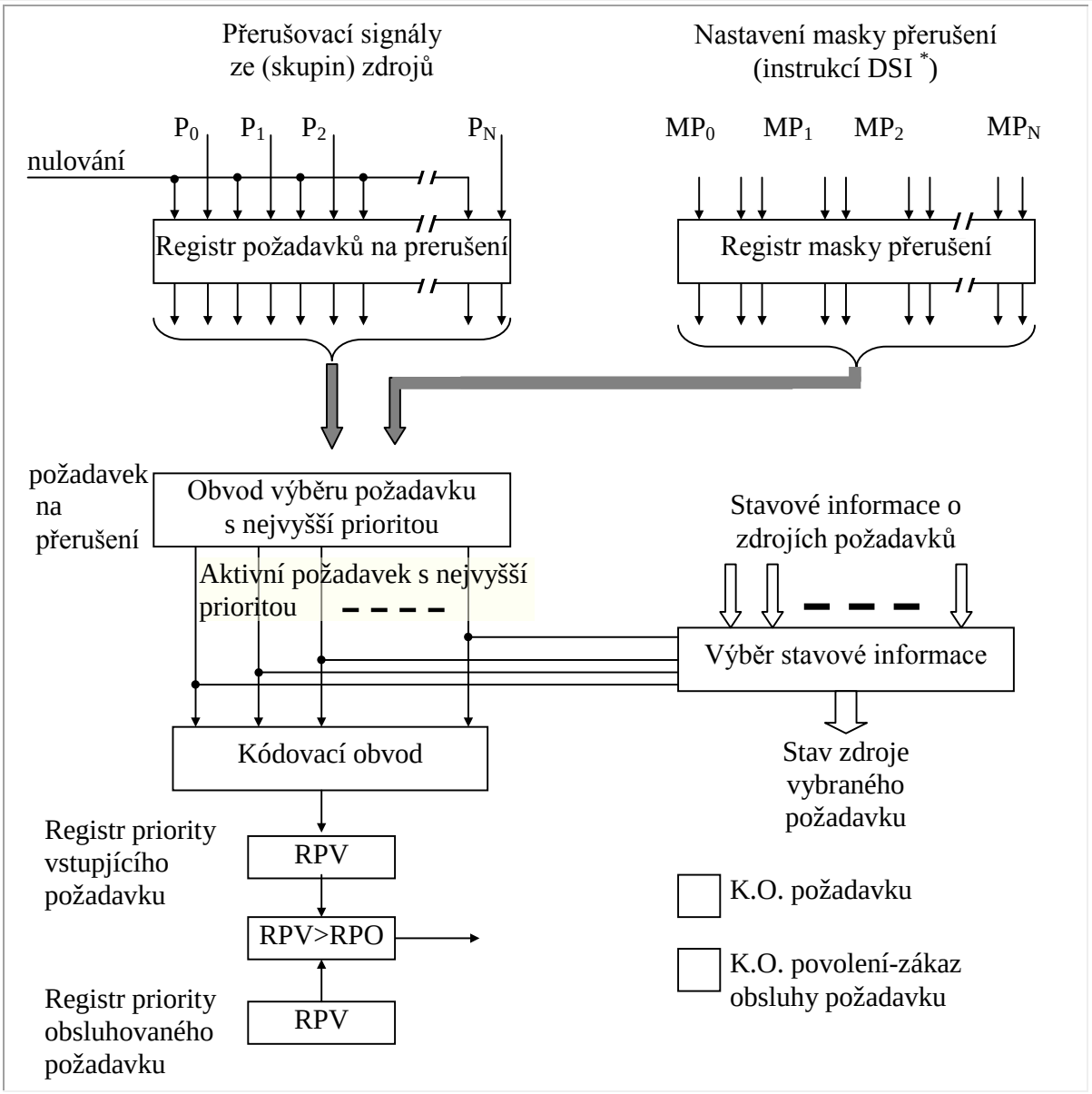
Instrukční cyklus s přerušením

Z hlediska řízení činnosti je přerušení (interrupt) událost, která změní sekvenci, ve které CPU vykonává instrukce. Jak jsme si již uvedli, je generováno HW nebo SW. Když se vyskytne přerušení, tak:

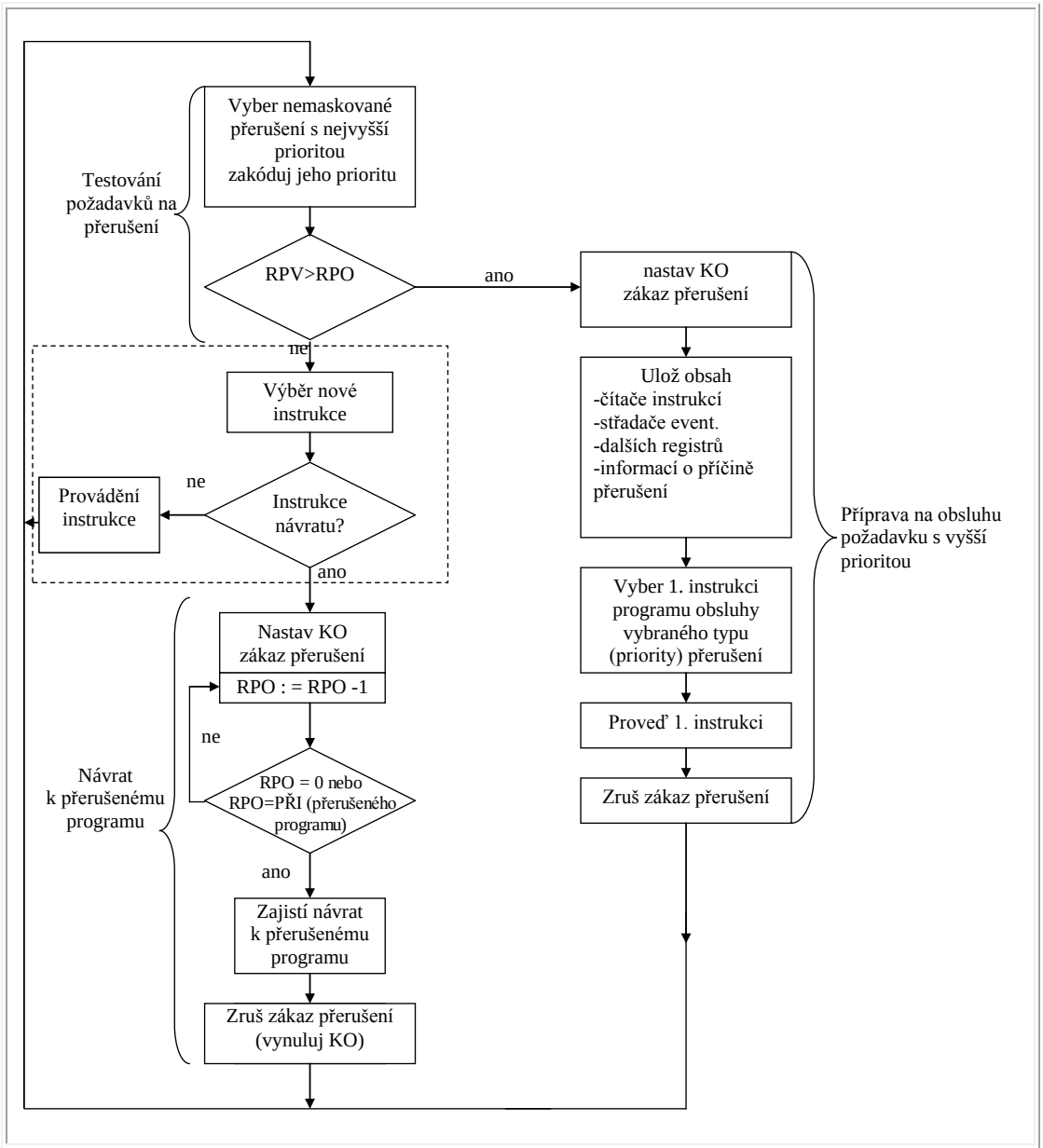
- operační systém se ujme řízení (tj. HW předá řízení do operačního systému),
- operační systém uloží stav přerušeného procesu,
- operační systém analyzuje přerušení a předá řízení do obslužné rutiny přerušení (dnes většinou zajišťuje již HW),
- obslužná rutina přerušení pak obslouží přerušení,
- je obnoven stav přerušeného procesu,
- přerušený proces pokračuje.

Přerušení může být generováno **synchronně** běžícím procesem (trap instrukce) nebo **asynchronně**, které vznikne nějakou vnější událostí. Dalším možným typem jsou **výjimky (exceptions)** vzniklé nesprávným chováním programu (dělení 0, neznámá instrukce, ...). Hlavní výhodou konceptu přerušení je odstranění potřeby **osahávání (polling)** zařízení. Zdroj vydá požadavek přerušení, řadič přerušení určí konkrétní příčinu přerušení a provede potřebné akce. Při řízení přerušení musí být zachována možnost vrátit řízení přerušenému programu do místa, kde byl přerušen. Na místo přerušitelnosti se nekladou omezení, musí se tedy uchovat stav běhu procesu (obsah čítače instrukcí, registru instrukcí, registrů operační jednotky).

Na následujících obrázcích jsou uvedeny nutné technické prostředky přerušovacího systému a detailní algoritmus obsluhy přerušení realizovaný těmito technickými prostředky.



Technické prostředky obsluhy přerušení



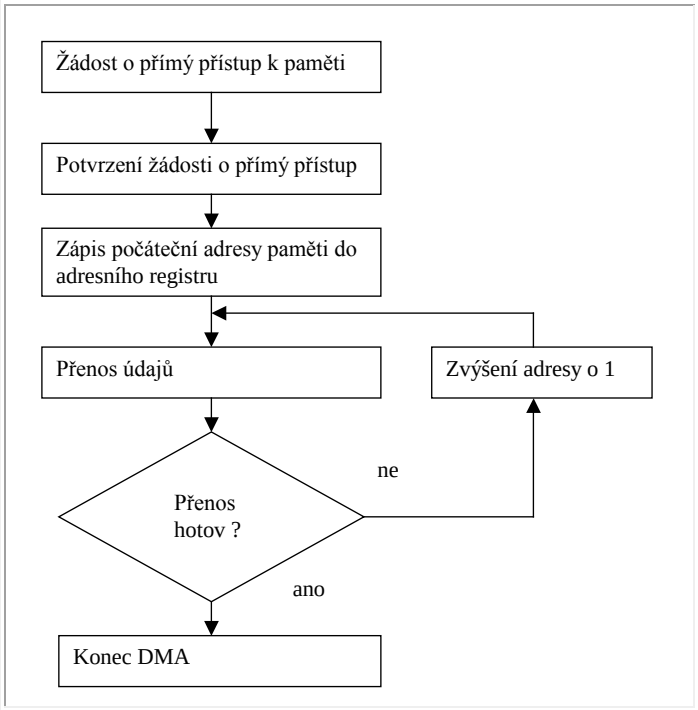
Algoritmus obsluhy přerušení

6.1.3 Vstup a výstup dat přímým přístupem do paměti

Pro odlehčení procesoru bývají součástí počítače obvody schopné realizovat větší množství I/O operací (jedná se o odchylku od Von Neumannova schématu počítače):

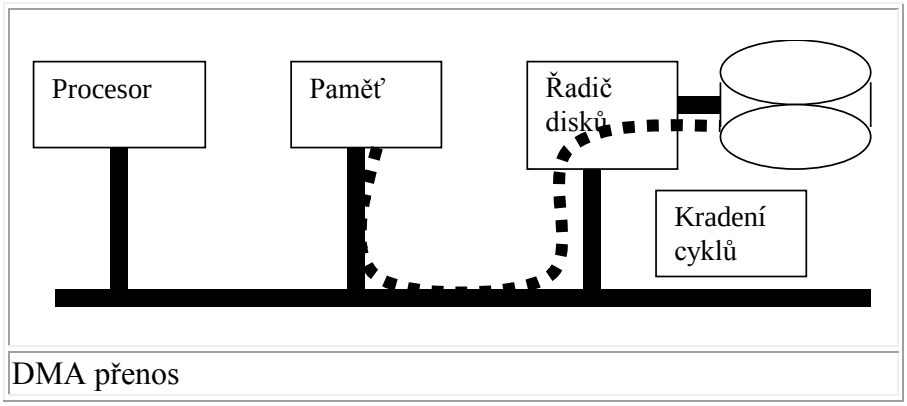
- **Kanály přímého přístupu do paměti (DMA)**: pro kopírování bloků dat mezi pamětí a I/O zařízením. Je třeba je naprogramovat zápisem do hardwarových registrů.
- **Specializované I/O procesory** (někdy nazývané kanály): jsou řízeny posloupností vlastních instrukcí (tzv. kanálovým programem):
  - selektorové - obsluhuje 1 rychlé zařízení (magnetiský disk, CD mechanika),
  - multiplexní - mohou obsluhovat několik pomalých zařízení (tiskárny, některé terminály, apod.).

DMA kanály jsou specializované řadiče řídící kopírování bloků dat mezi pamětí a I/O zařízením. Činnost řadiče je řízena algoritmem dle následujícího vývojového diagramu.



Vývojový diagram přenosu s přímým přístupem k paměti (DMA)

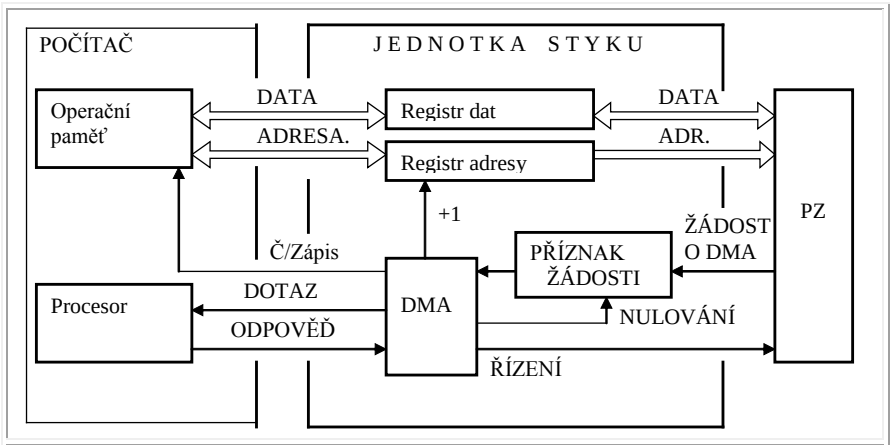
Neprogramově řízený přenos využívá zvláštní obvody tzv. kanál přímého přístupu k paměti (DMA). Při jednoduchém řešení kanálu DMA bývá během přenosu tímto kanálem CPU vypojen z činnosti. Při jiném složitějším řešení se přenos uskutečňuje tzv. kradením cyklů probíhajícího programu, tj. CPU pokračuje zpomalenou rychlostí v běžné činnosti.



DMA přenos

Princip přenosu řízeného technickými prostředky při přímém přístupu k paměti je uveden na následujícím obrázku. Když periferní zařízení (PZ) vyšle žádost o přímý přístup do paměti, procesor přejde do stavu čekání (WAIT) nebo odpojení od sběrnice (HOLD) a informuje o tom obvody DMA. Když skončí přenos dat mezi PZ a pamětí, příznak žádosti o DMA se vynuluje, zruší se

dotazovací signál z obvodů DMA, procesor se připojí na sběrnici a tím k paměti.



DMA řízená periferie

6.1.4 Vstupní a výstupní řadiče

Vstupní a výstupní řadiče slouží k připojování I/O zařízení. Z hlediska programátora řadič vypadá jako sada hardwarových registrů, přičemž registry mohou být:

- jen pro čtení,
- jen pro zápis,
- pro čtení i zápis.

Při inicializaci počítače je potřeba zjistit, které řadiče jsou v počítači zapojeny a inicializovat je. Při vstupu nebo výstupu dat je zpravidla nutné čtením z řadiče zjistit, zda je zařízení připraveno zapsat do řadiče příkaz a zapsat nebo přečíst data. Pokud není možné ihned pokračovat, zařízení zpravidla signalizuje svoji připravenost vyvoláním přerušení.

Podle architektury počítače se vstupy/výstupy dělí na:

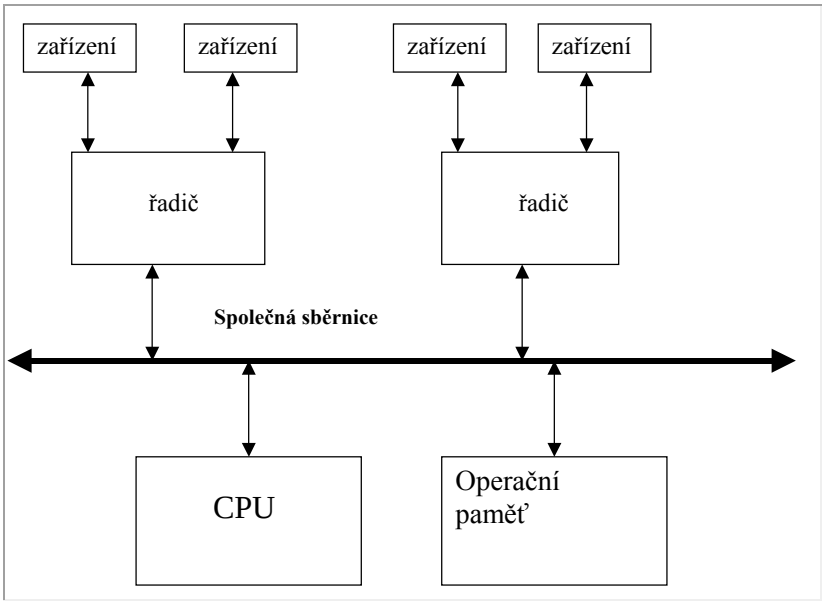
- **paměťově mapované**: registry jsou adresovány jako paměť, přístupné pomocí běžných operací čtení a zápisu do paměti,
- **izolované:** registry jsou přístupné pomocí speciálních instrukcí (zpravidla nazývaných IN a OUT); díky tomu jsou adresní prostory paměti a vstupů/výstupů oddělené.

S rozvojem počítačů postupovala na mnoha úrovních i standardizace. Standardizace připojování periferních zařízení je závazná jak pro výrobce periferních zařízení tak pro výrobce počítačů. Jeden způsob standardizace je ve vzniku propojení částí počítačů prostřednictvím sběrnic, což vedlo k normalizaci vnitřních a vnějších sběrnic a jejich komunikačních protokolů. V moderních počítačových systémech se pak setkáváme s různými úrovněmi sběrnic:

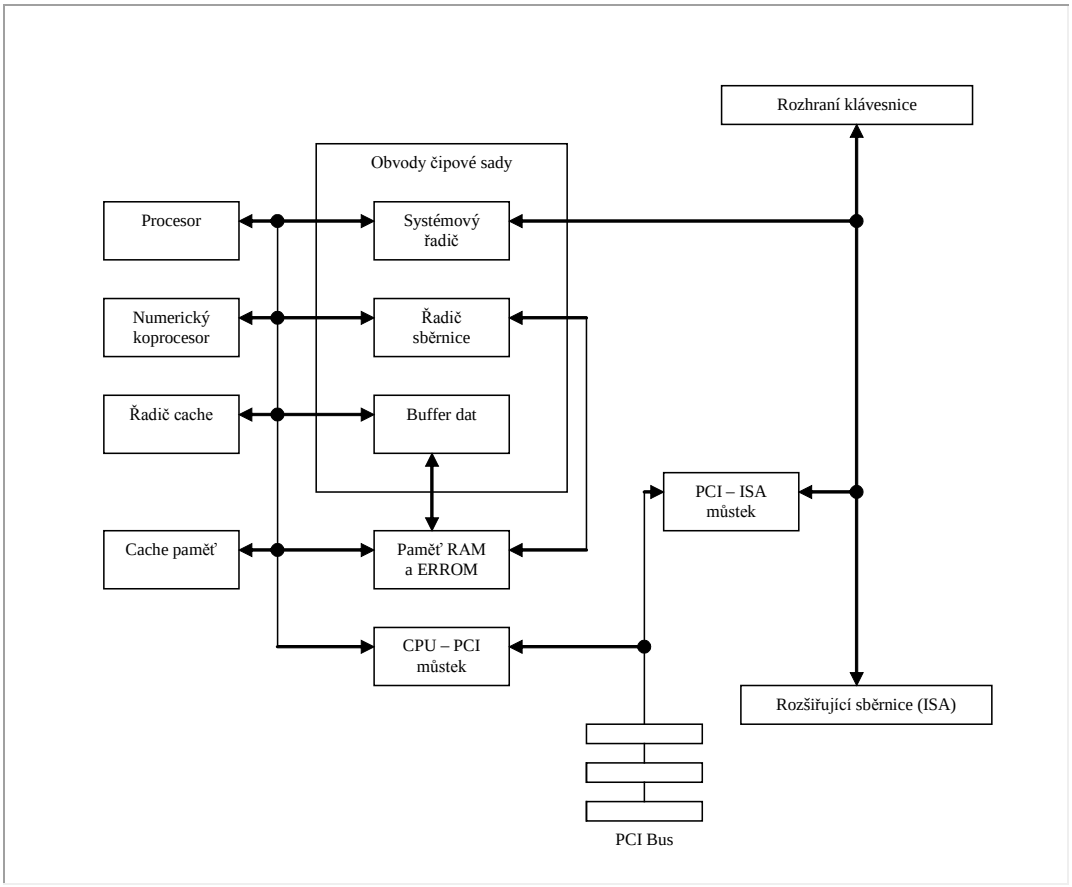
- **vnitřní** (obvodová) sběrnice propojující funkční jednotky uvnitř integrovaného obvodu nebo tištěného spoje,

- **vnější** (systémová, společná) sběrnice propojující zásuvné jednotky nebo funkční celky,
- **I/O** (paralelní, seriové) sběrnice pro jednotné připojování různých periferních zařízení.

Na následujících obrázcích jsou naznačeny sběrnice PC.



Vstupní/ výstupní řadiče



Připojení V/V zařízení v PC pomocí sběrnice

6.2 Ovladače periferií

Hlavním úkolem operačního systému ostatně není ani tak zařízení přímo ovládat, jako zajistit jejich korektní přidělování jednotlivým procesům - toto je samozřejmě pravda pouze do jisté míry - rozumný operační systém pochopitelně nabízí již hotové služby pro ovládání zařízení na daleko vyšší úrovni, než jakou nabízí zařízení samo o sobě. Tyto služby však nemusí být nutně součástí vlastního operačního systému. Namísto toho mohou být soustředěny např. na sdílených knihovnách. Srovnání si můžeme uvést např. se správcem paměti, který operační paměť jednotlivým procesům pouze paměť přiděluje, aniž by se jakkoli staral o to, co do přidělené paměti budou procesy zapisovat.

Ačkoli je pravda, že hlavním úkolem správce zařízení je starat se o jeho přidělování jednotlivým procesům a ne zařízení přímo ovládat, existuje několik velmi dobrých důvodů, proč v praxi nelze ani ovládání zařízení nechat na jednotlivých procesech. Ten, kdo přímo ovládá periferní zařízení, totiž musí mít přístup k potenciálně nebezpečným službám operačního systému i technického vybavení (jako je např. mechanismus přerušení). Řada periferních zařízení navíc může při špatném řízení zapříčinit zhroucení operačního systému. Existují dokonce zařízení, u kterých lze špatným řízením docílit zničení technického vybavení. Není proto možné nechat procesy, aby samy

ovládaly jednotlivá zařízení v žádném případě. Proto ovladače zařízení musí být součástí operačního systému.

Ovladač samozřejmě musí nabízet služby dostatečně silné na to, aby bylo možné využít plně všech možností zařízení. Bude-li s pomocí ovladačů zařízení pracovat i se soubory, je nutné mít k dispozici nějaký způsob detekce konce souboru nebo zjištění jeho velikostí. Zároveň však by měly být všechny ovladače v maximální možné míře navzájem podobné, alespoň z hlediska rozhraní mezi ovladačem a procesem, který jej využívá. Je tedy zapotřebí vytvořit takovou skupinu služeb, která bude dobře sloužit pro přístup ke kterémukoli zařízení. Mezi nimi přitom musí být alespoň jedna speciální služba, která zpřístupní neobvyklé a výjimečné vlastnosti konkrétního zařízení, samozřejmě za tu cenu, že programátor musí typ zařízení sám detekovat.

Řízení periferních zařízení se realizuje pomocí následujících služeb:
- Služba **Init** zajistí inicializaci zařízení na začátku práce celého systému nebo po jeho restartování. Bývá obvykle bez parametrů a může vracet stav zařízení (tj. podařilo-li se jej inicializovat nebo ne).

- Služba **Open** slouží k vytvoření kanálu nebo přípravě zařízení pro přenos dat. Typicky se jedná o zařízení pro komunikaci. Před zahájením vlastní komunikace bývá nutné navázat spojení, vytvořit komunikační kanál, kterým bude komunikace probíhat. Zařízení, která tuto službu nepotřebují, ji mohou snadno implementovat jako prázdnou. Služba open navíc může snadno sloužit k virtualizaci zařízení. Její volání vytvoří nové virtuální zařízení, které bude samo nadále sloužit programu. Nejtypičtějším případem zde bývá ovladač disku. Jeho služba open obvykle vytvoří logické zařízení odpovídající souboru. Parametrem služby open bývá obvykle jméno požadovaného virtuálního zařízení nebo kanálu. Služba vrací nové číslo zařízení, které bude program nadále používat.
  - Služba **Close** slouží k uzavření vytvořeného kanálu nebo zrušení virtuálního zařízení. Jejím jediným parametrem bývá číslo zařízení.
  - Služby **Read, Write, Getc** a **Putc** slouží pro vlastní výměnu dat mezi zařízením a programem. První dvě předávají celé bloky dat, druhé dvě postupují po jediném bytu.
  - Služba Seek je dokladem toho, že u velkého množství zařízení se můžeme, vrátit a znovu si přečíst data, která jsme již jednou četli nebo můžeme nějakou skupinu údajů přeskočit a číst až za nimi. Služba sděluje ovladači relativní pozici dat, která chce program číst v rámci celého datového bloku.
  - Konečně pro speciální případy, které není možné pokrýt dosud popsanými službami, je k dispozici služba **Cntl.** Její funkce není definována, a záleží prostě na typu konkrétního zařízení.

Základní princip komunikace s typickým zařízením vypadá takto:
- Ovladač na základě požadavku některého procesu zapíše data na vhodné místo v paměti a pak zařízení aktivuje, tj. předá mu příkaz odešli data z konkrétní adresy.
- Zařízení po nějakém čase data odešle. Pak dá ovladači na vědomí, že je možné připravit další data. Jestliže ovladač již odeslal vše, co bylo

zapotřebí, ukončí proces; jinak ovladač připraví další dávku dat a zařízení opět aktivuje.

- Tento postup se opakuje, dokud není požadavek procesu zcela splněn (nebo dokud nedojde k nějaké chybě, která jeho splnění znemožní).

Je zřejmé, že se jedná o komunikaci typu producent/konzument, kterou jsme poznali již v odstavci o synchronizaci procesů. Víme, že má-li taková komunikace probíhat efektivně a bez časových ztrát, i když jsou oba procesy (zde příprava dat programem na straně jedné a jejich odesílání zařízením na straně druhé) různě rychlé a pracují nestejnoměrně (tak tomu v daném případě skutečně bývá), je zapotřebí použít semaforů. To ale znamená, že ovladač musí být složen ze dvou samostatných programů, které spolu popsaným způsobem komunikují. První z nich slouží jako producent (dosud jsme mu říkali ovladač), a druhý je konzumentem (obvykle je umístěn v obslužné rutině přerušení, kterým zařízení oznamuje odeslání dat).

Je tedy zřejmé, že ovladač běžného zařízení má dvě části, tvořící rozhraní. První z nich - tzv. horní polovina - je volána procesy (stává se tedy vlastně součástí volajícího procesu) a zajišťuje předávání údajů do sdílené paměti pro výstup a odebírání údajů ze sdílené paměti pro vstup. Horní polovina se zařízením přímo nekomunikuje až na jedinou výjimku a tou je aktivace zařízení na samém začátku výstupu. Druhá část ovladače - tzv. dolní polovina - pak zajišťuje synchronizaci mezi zařízením a horní polovinou ovladače. Služby horní poloviny komunikují s dolní polovinou ovladače na základě vztahu producent-konzument nad sdílenou pamětí. Při výstupu dat z počítače je horní polovina ovladače v postavení producenta a dolní polovina pracuje jako konzument. Při čtení dat je tomu právě naopak.

Představme si, že by uživatelské programy volaly přímo funkce, které tvoří horní polovinu ovladačů jednotlivých zařízení. Takový přístup by nebyl právě praktický - program by pracoval neměně s jediným pevně zvoleným zařízením, a pro použití jiného zařízení by bylo zapotřebí jej znovu přeložit. Operační systémy proto musí obsahovat tabulku zařízení, která mapuje jméno nebo identifikační číslo zařízení na jednotlivé obslužné rutiny.

Nevýhodou popsaného mechanismu ovladače je to, že proces musí (v rámci některé ze služeb horní poloviny) čekat, než bude jeho požadavek splněn. Mezitím může samozřejmě pracovat jiný proces. Proto je výhodnější následující mechanismus.

Proces by si prostřednictvím služeb horní poloviny ovladače vyžádal vstup nebo výstup a pak by okamžitě pokračoval v práci. Nesměl by ovšem měnit údaje, které byly určeny pro výstup nebo snažit se číst údaje ze vstupu. Teprve ve chvíli, kdy proces opravdu načtená data potřebuje (nebo kdy je nutné změnit data zapisovaná), by dal ovladači na vědomí, že nyní již musí počkat. Ovladač by zjistil, zda dosud data nejsou načtena (zapsána) a pouze v takovém případě by proces pozastavil.

Tohoto mechanismu lze dosáhnout dvěma způsoby. Buďto můžeme přeprogramovat horní polovinu ovladače tak, že dosud popsané služby

nebudou čekat na ukončení přenosu a namísto toho přibude nová služba **Iowait,** která čekání zajistí (tj. k návratu z ní nedojde dříve, než bude přenos ukončen a než dolní polovina ovladače nastaví patřičný semafor). Druhou možností je vytvořit samostatný proces, který bude sám využívat ovladač zařízení a ostatní procesy mu budou své požadavky předávat prostřednictvím zpráv. Takový proces pak může čekat na ukončení přenosu, zatímco procesy, které požadavky vyvolaly mohou bez problémů běžet dále. Proces vyhrazený pro obsluhu některého zařízení nazýváme **server.**

Některé ovladače jsou samozřejmě pevnou součástí operačního systému. Ve většině případů však tomu musí být jinak - operační systém musí být schopen pracovat s řadou nejrůznějších zařízení. Není dost dobře možné, aby obsahoval všechny potenciálně potřebné ovladače. Musí být také možné zapojit do operačního systému i zařízení nové, vytvořené později než operační systém sám.

Snad všechny operační systémy proto nabízejí prostředky, umožňující při startu systému zavést potřebné ovladače. Jednotlivá zařízení v takovém případě bývají obvykle identifikována jménem, které je součástí každého ovladače. Přidávání nových druhů zařízení je realizováno několika způsoby:

- zásahem do jádra operačního systému,
- nainstalováním příslušného ovladače zařízení,
- kombinací obou možností.

Účelem správy je tedy zabezpečit přístup k zařízení (pro operační systém) standardním způsobem. Zpravidla se požaduje transparentnost přístupu k zařízením (tj. stejný přístup jako k souborům, kdy až při běhu programu lze určit, kam výstup půjde). Dalším úkolem je zajistit přidělování a sdílení zařízení, ochrana zařízení (přístupová práva různá pro různé uživatele).

6.3 Časovač

Časovač umožňuje plánovat události při kterých proces může vyvolat službu operačního systému, která jej zablokuje ("uspí") do uplynutí požadovaného času. Časovač se taktéž používá pro některé systémové akce (přidělování časových kvant, zastavování motorů disketových mechanik apod.).

Není možné předpokládat, že počítač má dostatek hardwarových časovačů pro všechny účely. Zpravidla je k dispozici jen jeden časovač a programová časová fronta. Hardwarový časovač se pak používá pro odměření času do nejbližší události ve frontě. Do fronty jsou události řazeny tím způsobem, že se zjistí, po které události má k nově přidávané události dojít a spočítá se a zaznamená čas (který má uplynout mezi těmito dvěma událostmi). Při vynulování časovače se vybere první událost z fronty, provede se příslušná akce a hardwarový časovač se přeprogramuje na čas následující události.

Kontrolní otázky:

1. Vysvětlete způsob řízení V/V zařízení na základě přerušení?
2. Vysvětlete způsob řízení V/V zařízení pomocí DMA?

Úkoly k zamyšlení:

1. Lze jedno zařízení připojit jak pomocí přerušení tak i prostřednictvím DMA?

Korespondenční úkol:

1. Prostudujte si u Vašeho počítače připojení veškerých periferních zařízení a zjistěte, zda nejsou nějaké konflikty v připojení těchto zařízení. Tento soupis zašlete.

Shrnutí obsahu kapitoly

V této kapitole jste se seznámili se základními principy komunikace procesoru s periferními zařízeními. Důraz v této kapitole byl kladen na pochopení funkce ovladače periferního zařízení.

7 Správa sekundární paměti

V této kapitole se dozvíte:

- Jak pracují diskové paměti a jak se vyhledává informace na disku?

Po jejím prostudování byste měli být schopni:

- Popsat způsoby vyhledávání informací na disku.
- Znát způsob výpočtu různých parametrů disků.

Klíčová slova této kapitoly:

Konstrukce disků, přístup na disk.

Doba potřebná ke studiu: 2 hodiny

Průvodce studiem

Studium této kapitoly je ve srovnání s předešlými kapitolami poměrně jednoduché. V případě že máte prostudovány základy architektury počítačů pak studium této kapitoly je pouze doplněním znalostí o vnějších diskových pamětech..
Na studium této části si vyhraďte 2 hodiny. Po celkovém prostudování a vyřešení všech příkladů doporučujeme vypracovat korespondenční úkol.

Sekundární paměť je v převážné míře realizována pomocí pevných disků, které fungují jako vstupní i výstupní zařízení. Na disky se zpravidla ukládají data v podobě souborů, které jsou odlišeny jménem.

7.1 Konstrukce disku

Magnetické disky jsou tvořeny několika kruhovými deskami umístěnými na společné ose. U pevných disků jsou desky kovové, diskety jsou tvořeny jedním kotoučem z ohebné plastické hmoty. Jednotlivé desky jsou z jedné nebo obou stran pokryty magnetickou vrstvou. Pro přístup (čtení nebo zápis) ke každé vrstvě (povrchu) se používá jedna magnetická hlava, která při otáčení disku (pevné disky rotují rychlostí až 10 000 otáček za minutu, novější i rychleji) opisuje na povrchu kružnici, jejíž průměr závisí na vystavení hlavy. Na každém povrchu se tím vytváří soustava soustředných kružnic nazývaných stopy (diskety mívají několik desítek stop, pevné disky stovky až tisíce). Hlavy pro jednotlivé povrchy jsou pevně spojeny, takže v jednom okamžiku (při určitém vystavení hlav) jsou zároveň přístupné stejné stopy na všech deskách (tzv. cylindr, válec). Všechny hlavy na stejné stopě tvoří cylindr.

Stopy jsou rozděleny na sektory. V současné době se používají zpravidla sektory pevné délky o velikosti 128 až 4096 bytů. Stopy blíže středu disku jsou kratší, přesto se zpravidla na všechny stopy zaznamenává stejný počet sektorů (záznam u středu disku je hustší). Proto se u některých disků na stopách blíže

středu snižuje záznamový proud (tzv. prekompenzace záznamu). Sektor je nejmenší část disku, kterou je možné číst nebo zapisovat. Pokud má být změněn jediný byte, musí OS zajistit načtení celého sektoru, změnu příslušného bytu a zpětný zápis sektoru na původní místo.

Při přístupu k určitému sektoru musí disková mechanika vystavit hlavy na požadovanou stopu a počkat až požadovaný sektor projde pod hlavou. Předpokládejme, že disk má *S* sektorů, *H* hlav a *T* stop. Převod trojrozměrné adresy (s, h, t), kde *s* je číslo sektoru, *h* je číslo hlavy a *t* je číslo stopy (cylindru) na jednorozměrný prostor sektorů je pomocí vzorce

$$s + S \times (h + H \times t).$$

Příklad č.1:

Jako příklad si nyní uveďme výpočet kapacity disků a přístupových dob při čtení dat. Mějme disk s následujícími parametry:

- 20 povrchů,
- 25 sektorů/stopu,
- 800 stop/povrch,
- 512 bytů/blok,
- 3600 ot/min,
- rotační zpoždění = 16,7 ms,
- průměrná doba vystavení = 28 ms (mezi válci 7 ms).

Výpočet:
Slabik/stopu = slabik/blok x bloků/stopu = 512 x 25 = 12 800 B = 12,8 kB.
Slabik/povrch = slabik/stopu x stop/povrch = 12 800 x 800 = 10 240 000 B = 10,24 MB.
Bytů/svazek = bytů/povrch x povrchů/svazek = 10,24 x 20 = 204,8 MB.

Pro výpočet doby přečtení celého disku (válec po válci, sekvenčně) je třeba vyjít z doby přečtení stopy = 16,7 ms (doba otočky).
Pak:
Doba přečtení válce = doba přečtení stopy x stop/válec = 16,7 x 20 = 334 ms.
Doba přečtení svazku = čtení 800 válců + 799 vystavení = 800 x 334 + 799 x 7 = 267200 + 5593= 272793 ms = 272,793 s.
Doba odpovídající čtení slabik = 273s/204,8MB = 1,33 ms/KB.

Příklad č.2:

V dalším příkladu mějme následující zadání. Disk s parametry:

- 6000 záznamů po 200B,
- 1K sektor,
- 16 sektorů/stopu,
- clustery po 4 sektorech,
- blokování více záznamů do sektorů,
- průměrná doba vystavení = 25 ms,
- 3600 ot/min,
- rotační zpoždění = 16,7 ms.

Kolik se obsadí clusterů?
Jak dlouho se čte cluster/soubor?

Řešení:
Volí se blokovací faktor 5 (24B fragmentace při K=1024).
Pak je 20 záznamů/cluster (4 sektory x 5) a je třeba 300 clusterů (6000 záznamů/20).
Průměrná doba čekání na natočení na cluster = rotační zpoždění/2 = 8ms.
Průměrná doba přenesení bloku = 1/4stopy = rotační zpoždění/4 = 4 ms.
Průměrná doba čtení clusteru = průměrná doba vystavení 25ms + 8ms + 4 ms = 37 ms.
Průměrná doba čtení souboru = 300 clusterů x 37 ms = 11,1 s.

Při srovnávacích rychlostních testech pevných disků (benchmark) se hodnotí především „Average Seek Time“ - průměrná doba vystavení hlaviček, udávaná v ms (doba, za kterou jsou hlavičky disku na potřebném místě), „Track to Track Seek Time“ - doba přechodu ze stopy na stopu, rovněž udávaná v ms a také parametr určovaný do značné míry řadičem – „Transfer Speed“ - přenosová rychlost v MB/s.

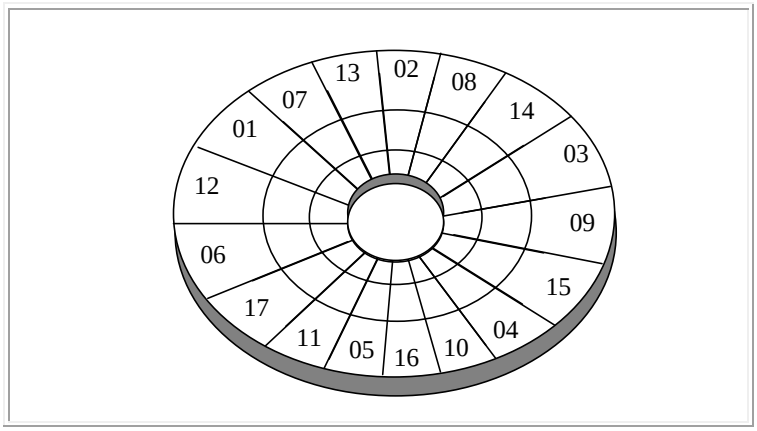
7.2 Prokládání sektorů (interleave)

Přenos dat mezi řadičem disku a operační pamětí trvá určitou nenulovou dobu. Pokud se má zpracovat několik následujících sektorů, může dojít k tomu, že během zpracování prvního z nich se disk pootočí a další sektor mezitím „ujede“. Počítač pak musí čekat celou otáčku, aby mohl tento sektor přečíst nebo zapsat. Totéž se může opakovat u dalších sektorů. V krajním případě může být pro načtení jedné stopy potřeba tolik otáček disku, kolik sektorů je na stopě.

Tento problém je možné odstranit pomocí prokládání sektorů (interleave). Sektory nejsou uloženy na stopě za sebou, ale „na přeskáčku“. Například je-li interleave faktor 1:3, bude pořadí sektorů na stopě následující:

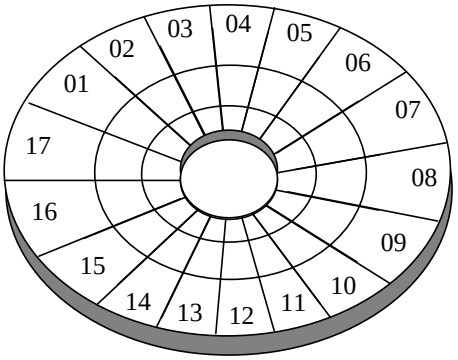
```
1 7 13 2 8 14 3 9 15 4 10 16 5 11 17 6 12
```

Poměr 1:3 značí, že po sobě následující sektory (například číslo jedna a dva) jsou "ob tři" sektory.



Prokládání 1:3

Interleave faktor 1:1 naopak znamená, že prokládání není použito (pořadí 1, 2, 3, 4, 5, ...).



Prokládání 1:1

Velký interleave faktor se používal zejména u pomalých počítačů. V dnešní době je rychlost počítače vzhledem k rychlosti otáčení disků dostatečná a interleave faktor proto ztrácí smysl (nebo je před uživatelem skryt).

Moderní pevné disky disponují značnou vlastní „inteligencí“ a jsou schopny detekovat problematická místa svého povrchu a přemapovat je na volné stopy, které jsou k tomu určeny a jsou jinak uživatelsky nedostupné.

7.3 Obsluha požadavků na přístup k disku

Čas přístupu na disk je dělen na tři části:

- SEEK - přesun hlavy na požadovanou stopu (cylindr),
- LATENCY - otočení disku na začátek požadovaného sektoru,
- TRANSFER - přesun dat z disku/na disk.

Při velkých zátěžích v OS s více procesy nejvíce zpomaluje přístup na disk čas SEEK.

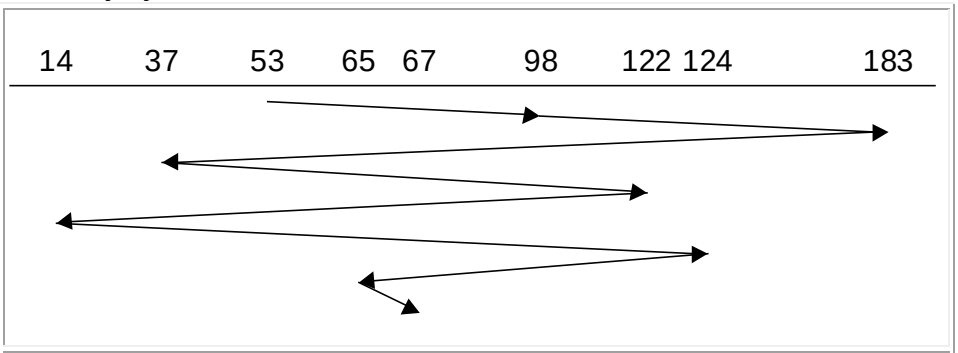
Existuje několik metod plánování disku a je možná i jejich adaptivní kombinace v závislosti na zátěži. Při vyřizování požadavku OS na disk je potřeba nejprve vystavit hlavy na příslušnou stopu a pak počkat, až bude požadovaný sektor pod hlavami. U víceúlohových systémů mohou přicházet požadavky na disk rychleji, než je možné je vyřizovat.

7.3.1 FCFS (First Come, First Served)

Vyřizování požadavků v pořadí, jak přicházejí (tzv. FIFO nebo FCFS - First In First Out, First Come First Serve) není optimální a je vhodný pro lehké zátěže.

Příklad:
Žádosti přicházejí v pořadí: 98, 183, 37, 122, 14, 124, 65, 67.

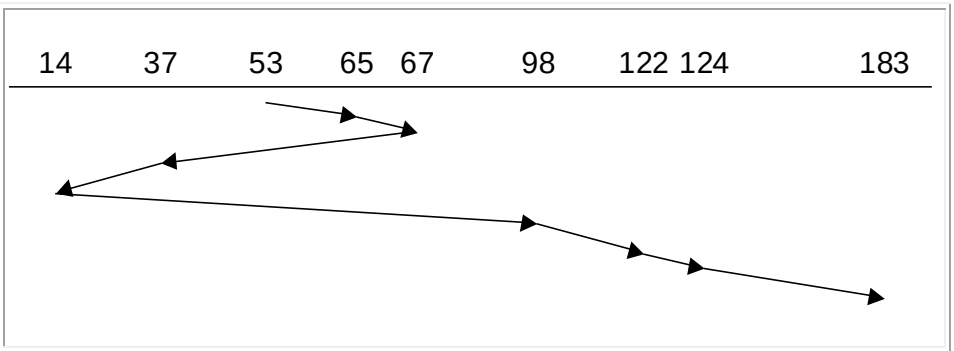
Hlavy jsou na pozici 53.
Na obrázku je naznačeno vyřizování požadavků na vystavování hlav na požadovaný cylindr.



Princip FCFS (First Come, First Served)

7.3.2 SSTF (Shortest Seek Time First)

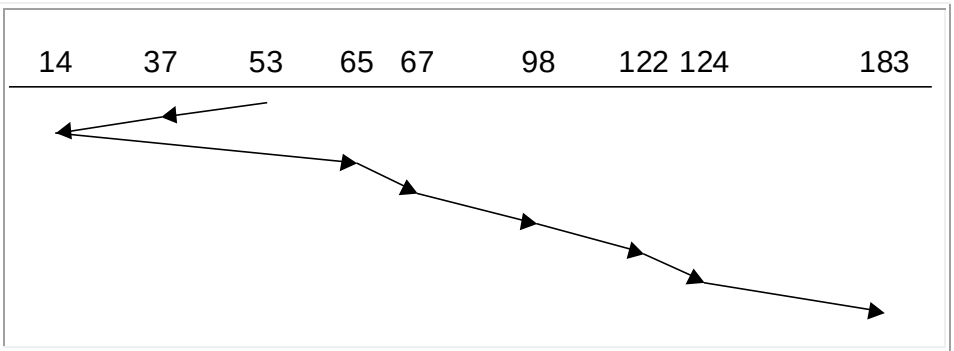
Naplánován je požadavek s nejmenším relativním pohybem hlavy. Existuje zde možnost, že nastane starvation.



Princip SSTF (Shortest Seek Time First)

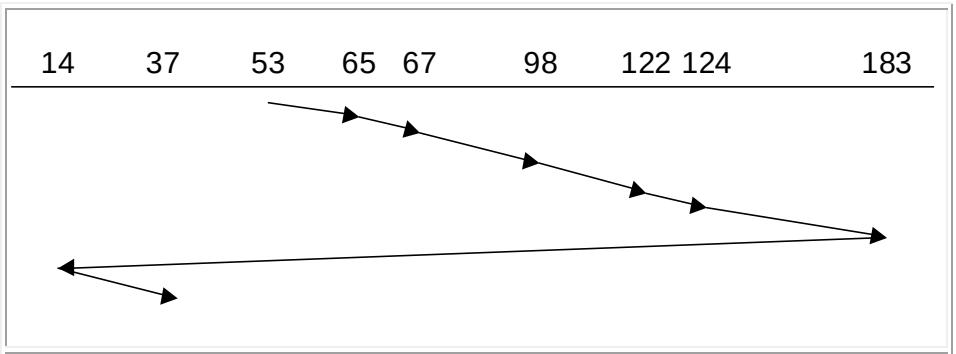
7.3.3 SCAN

Je určen směr pohybu hlav. Z fronty jsou zpracovávány pouze požadavky postupně v určeném směru. Po zpracování nejkrajnějšího požadavku se směr pohybu hlav obrátí.



Princip SCAN

Podvariantou je C-SCAN (Circular SCAN), který posouvá hlavy pouze v jednom směru a po zpracování nejkrajnějšího požadavku přesune hlavy opět na začátek.



Princip C-SCAN (Circular SCAN)

Obě varianty jsou vhodné i pro velmi těžké zátěže.

Poznamenejme, že moderní řadiče disků umí optimalizovat i požadavky pro přístup k disku v závislosti na poloze hlav a natočení disku. Výsledkem je změna pořadí požadavků a zrychlení vykonání požadavků v celkovém součtu.

Kontrolní otázky:

1. Které jsou nejzákladnější parametry disku?
2. Srovnejte různé metody přístupu na disk?

Úkoly k zamyšlení:

1. Lze u více diskových mechanik připojených k řadiči měnit algoritmy plánování pohybu hlav?

Korespondenční úkol:

1. Prostudujte si u Vašeho počítače připojení diskové paměti a zjistěte, jakým způsobem plánuje řadič disků čtení dat. Tento soupis zašlete.

Shrnutí obsahu kapitoly

V této kapitole jste se seznámili s principy plánování pohybu hlavy na disku. Důraz v této kapitole byl kladen na pochopení algoritmů.

8 Správa souborů

V této kapitole se dozvíte:

- Proč je systém ukládání souborů na velkokapacitních záznamových zařízeních důležitou součástí dobře fungujícího operačního systému.

Po jejím prostudování byste měli být schopni:

- Charakterizovat základní způsoby ukládání souborů na disk.
- Znát různé organizace struktur adresářů.
- Porozumět typům operací, které musí obsahovat operační systém při práci se soubory.

Klíčová slova této kapitoly:

Systém souborů, struktura adresářů, přístup k souborům..

Doba potřebná ke studiu: 2 hodiny

Průvodce studiem

Studium této kapitoly je jednoduché a popisným způsobem zde nastudujete způsob ukládání souborů na disk a organizaci adresářů.
Na studium této části si vyhraďte 2 hodiny.

Pod systémem souborů chápeme datové objekty, které se uchovávají na vnějších pamětech počítače. V této kapitole se budeme zabývat jen základními principy.

8.1 Základní pojmy

Systémy souborů:

- Soubory uchovávají data a programy. Operační systém (OS) implementuje abstraktní koncepci souborů správou velkokapacitních záznamových zařízení.

Organizace systému souborů:

- OS odstiňuje uživatele od fyzických vlastností záznamových zařízení a vytváří logickou jednotku uložení - **soubor.** Soubory jsou mapovány OS na fyzické zařízení.

Systém souborů se skládá ze dvou částí:

- sady souborů obsahující vlastní uložené informace,
- struktury adresářů obsahující informace o souborech.

Koncepce souborů:

- Soubor je sekvence bitů, bytů, řádků nebo záznamů, jejichž význam je definován zakladatelem a uživatelem souboru.
- Soubor je obvykle pojmenován a má nějaké atributy (např. typ, jméno zakladatele, délka, čas poslední změny, ...).

Podpora typů souborů OS:

- Výhodou je možnost kontroly správného použití souboru uživatelem, další přídavné funkce (automatický make při spuštění programu, ...).
- Nevýhodou je velikost OS (pro každý typ souboru nějaký kód) a možnost neobsažení všech typů a následného špatného použití.
- Opačným extrémem je nepodporovat OS žádné typy souborů (např. UNIX) a chápat soubor jako sekvenci osmibitových bytů.

Uložení souborů na disku:

- Disky mají typicky definovanou velikost bloku určenou velikostí sektoru disku. Všechny I/O operace jsou prováděny po blocích a všechny bloky jsou shodné velikosti. Logické bloky souboru jsou pak zabaleny do fyzických bloků disku.
- Obvykle dochází ke ztrátě určité části fyzických bloků díky délce souboru či rozdílu mezi délkou fyzického a logického bloku.

Struktura adresářů:

- Adresář zařízení je na každém fyzickém zařízení a popisuje všechny soubory uložené na tomto zařízení. Obsahuje fyzické atributy souboru (délka, jak je umístěn na disku, ...).
- Struktura adresáře souborů je logickou organizací souborů na všech zařízeních. Obsahuje logické atributy souboru (jméno, typ, vlastník, přístupová práva, ...).
- Systém adresářů mapuje jména souborů na položky adresářů.

Operace se soubory: :

- CREATE - vytvoří soubor. Nalezení prostoru pro soubor a záznam nové položky do adresářové struktury.
- OPEN - otevře existující soubor. Prohledání adresářové struktury a získání atributů souboru. Vrací handle.
- CLOSE - zavře otevřený soubor.
- WRITE - zapíše do otevřeného souboru na pozici kurzoru souboru.
- READ - čte z otevřeného souboru z pozice kurzoru souboru.
- SEEK - nastavení pozice kurzoru v souboru.
- DELETE - smaže soubor.

Přístupové metody:

- **Sekvenční přístup** - čtení a zápis na pozici kurzoru souboru s automatickým posunem kurzoru.
- **Přímý přístup** - soubor je chápán jako očíslovaná sekvence záznamů a pomocí operace SEEK lze nastavit kurzor souboru na libovolný záznam souboru.
- **Paměťové mapování** - soubor je pomocí stránkování namapován (je obrazem paměti na disku) do virtuálního adresového prostoru a s jednotlivými záznamy v souboru lze manipulovat jako s pamětí.

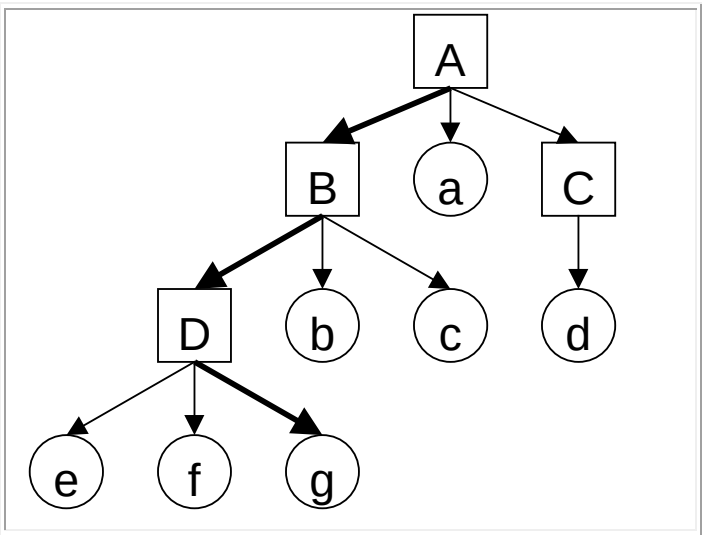
8.2 Organizace struktury adresářů

Souborům se přiřazují jména. Systém ovládání souborů hledá na základě udání jeho jména přístupovou cestu k souboru. Implementačně to obvykle znamená nalézt řídící blok souboru. Datová struktura používaná k zobrazení z množiny jmen souborů do množiny řídících bloků souborů je typicky tabulka, kterou nazýváme adresář nebo katalog. Adresář se může vytvořit jako nesetříděná nebo setříděná tabulka. Adresáře se pak mohou uložit jako soubor a uspořádat je do struktur. Jedna položka adresáře může nést údaj o umístění jiného adresáře. Jednoduchou formou takové struktury jsou hierarchické adresáře. Prakticky nejvýznamnější jsou jednoúrovňové, dvouúrovňové a stromové adresáře. Rovněž se často používají adresáře s acyklickou strukturou.

Jednoúrovňový adresář se obvykle implementuje „rozptýleně“ tak, že se na jednotlivých svazcích vytvoří tabulky obsahující informace o souborech na těchto svazcích.

U dvouúrovňového adresáře jsou v hlavním adresáři uloženy informace o uživatelských adresářích. Uživatel nemá přístup k hlavnímu adresáři a tudíž nemá uživatel **a** přístup k adresáři uživatele **b.**

Přirozeným rozšířením dvouúrovňových adresářů jsou adresáře se stromovou strukturou. V kořenu stromu je hlavní adresář. Uživatelské adresáře mohou mít vytvořenu vlastní hierarchii adresářů a v nich logicky seskupovat své soubory. Organizaci takových to adresářů si můžeme představit v nějaké podobě grafu s kořenem, viz obrázek.

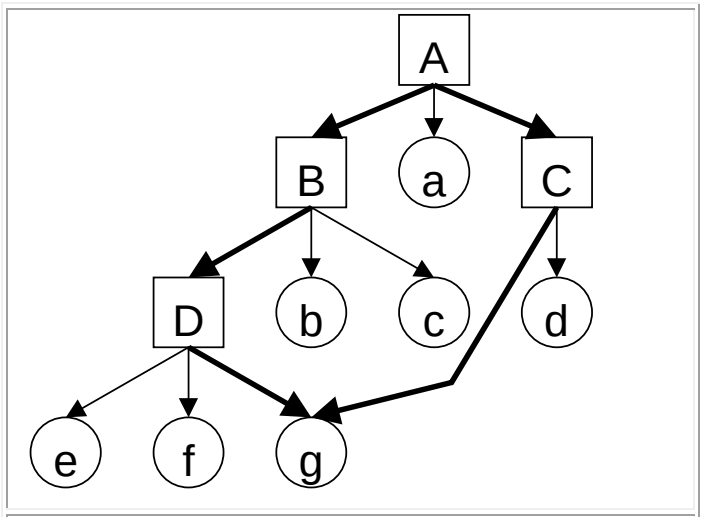


Stromová struktura adresářů /A/B/D/g

Pozice uživatele v adresářové struktuře se nazývá **aktuální adresář.** Identifikace uživatele, resp. jméno jeho adresáře a jméno souboru tvoří úplnou přístupovou cestu k souboru. Říkáme také absolutní přístupovou cestu. **Absolutní cesta** je dána jménem souboru spojené se jmény všech podadresářů ležících na cestě z kořenového adresáře k danému souboru. Obdobně existuje relativní cesta, která je dána jménem spojeným se jmény všech podadresářů

ležících na cestě z aktuálního adresáře k danému souboru. Obvykle vznikne přidáním speciálních jmen adresářů pro aktuální adresář a předchůdce adresáře.

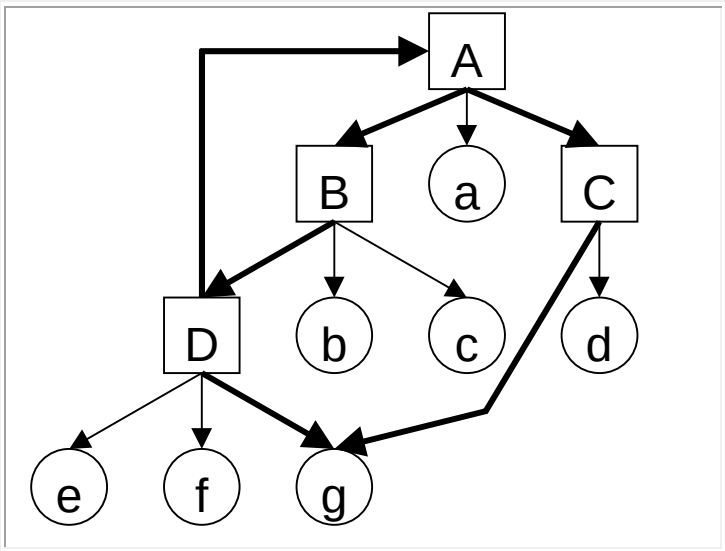
Zobecníme-li stromovou strukturu adresářů na acyklickou, dosáhneme toho, že k jednomu souboru může vést několik přístupových cest.



Acyklický graf adresářové struktury

Například soubor určený přístupovou cestou **/A/B/D/g** je rovněž určený přístupovou cestou **/C/g**.

Může-li být struktura adresářů obecným grafem, mohou se v přístupových cestách opakovat tytéž posloupnosti.



Obecný graf adresářové struktury

Například přístupové cesty **/A/B/D/g** a **/A/B/D/A/C/g** ukazují na stejný soubor.

Existují různé přístupy k více diskům:

- Oddělené systémy souborů na jednotlivých discích (A:, B:, C: - MS-DOS, Windows, podobně MacIntosh, VMS).

- Mountování filesystémů (UNIX). Na jednom disku je kořenový adresář. V hiearchické soustavě adresářů se na některý adresář namapuje celý disk (přepneme-li se do tohoto adresáře, ocitneme se na jiném fyzickém disku).

Ve starších operačních systémech byla značná omezení na jména souborů: krátká jména (CP/M maximálně 6+3 znaky, MS-DOS 8+3, Unix 14), nerozlišovala se malá a velká písmena, množina znaků, které mohou tvořit jméno souboru je značně omezená. Novější OS často rozlišují malá a velká písmena, povolují speciální znaky (mezera, +, !, :) a národní znaky (ěščřžýáié...). V pojmenovávání souborů vývoj směřuje k dlouhým jménům (desítky znaků).

8.3 Generační soubory

Ve většině operačních systémů se udržují i starší verze souborů. Pro odkaz na soubor lze používat buď pouze jméno (odpovídá nejnovější verzi souboru) nebo jméno a generaci. V některých operačních systémech jsou generační soubory součástí systému a v některých se řeší programově.

Operační systém zpravidla o souboru udržuje další informace:

- Atributy:
  - R - read-only, jen pro čtení,
  - H - hidden, skrytý,
  - S - system, systémový,
  - A - archive, nebyl archivován.
- Datumy a časy:
  - vytvoření,
  - poslední změny,
  - posledního přístupu.

U řady operačních systémů jsou ještě další informace o souborech ve kterých se např. určuje, jakou aplikací se má soubor zpracovávat.

V některých systémech jsou soubory členěny na datovou část a resource část (texty, obrázky, ikony, dialogy). Tyto operační systémy zpravidla umožňují změnu resource části programu bez nutnosti nového překladu programu.

Při zápisu do souboru (databáze) se celý soubor nebo určitá část souboru uzamkne a ostatní procesy nebo uživatelé nemohou po dobu uzamčení obsah příslušné oblasti měnit (nebo ani číst). Používá se například pro přístup k databázím (příklad: databáze místenek přístupná z několika předprodejů v různých městech).

8.4 Alokační metody

Alokační metody jsou dány možností přímého přístupu na disk. Existují tři základní metody alokace:

- souvislá,
- spojovaná,
- indexová.

8.4.1 Souvislá alokace

Každý soubor zabírá souvislou množinu bloků na disku. Adresa souboru je pak udána pouze adresou prvního bloku. Ostatní bloky na sebe navazují a vytvářejí na disku souvislý paměťový prostor.

Výhodou souvislé alokace je malá pravděpodobnost pohybu hlavou disku a hlavní nevýhodou je nutnost souvislého volného místa pro soubor a obtížná implementace dynamických změn velikosti souboru (zvětšování).

Souvislá alokace je jednoduchou metodou, při které se udává počáteční báze (číslo bloku) a délka tj. počet bloků. Lze zde realizovat sekvenční i přímý přístup k datům.

Zmíněná prostorová neúspornost je v případě dynamického přidělování místa na disku závislá na efektivitě hledání volného prostoru (FIRST FIT x BEST FIT, NEXT FIT, WORST FIT, . . .). V případě, že není k dispozici souvislý prostor pro uložení dat na disku je nutná reorganizace bloků dat formou defragmentace, což je časově dlouhá operace.

Z těchto důvodů nemohou soubory růst a musí se již při zavádění souboru na disk určit jeho počáteční rozměr. Při překročení vyčleněné kapacity se provede ABORT procesu a musí se provést nové vyhledání volného prostoru tzn. že se vytvoří kopie souboru do jiného prostoru což způsobuje skrytá zpomalení procesu ukládání dat na disk.

8.4.2 Spojovaná alokace

Spojovaná alokace odstraňuje nevýhody souvislé alokace pospojováním bloků použitých pro soubor. Soubor je pak vázaným seznamem diskových bloků, přičemž bloky mohou být rozptýleny po disku libovolně. Nevýhodou tohoto přístupu je ztráta "hezké" velikosti a relativní pomalost při náhodném přístupu do souboru. Řešením je FAT tabulka (File Allocation Table), kde spojový seznam je tvořen polem na vyhrazeném místě disku.

Počet alokačních bloků přidělovaných souboru bývá dán alokační politikou operačního systému. Typicky se nepřidělují prázdné alokační bloky, nevytváří se žádná rezerva. Většina alokačních strategií má snahu o alokaci sousedních alokačních bloků. Délka alokačního bloku je dána jednak parametry operačního systému (16 bitový, 32 bitový) a také především kapacitou disku. Platí, čím kratší alokační blok, tím více existuje alokačních bloků. Krátký alokační blok vytváří fragmentaci souboru a častou alokaci nesousedních alokačních bloků. Dlouhý alokační blok snižuje fragmentace souboru, hrozí však tzv. vnitřní fragmentace, tj. nevyužívání celé kapacity přiděleného alokačního bloku.

8.4.3 Indexová alokace

Indexová alokace upravuje špatný přímý přístup spojované metody umístěním indexů bloků jednotlivých souborů pohromadě. Ukazatelé bloků přidělených

souboru jsou seskupeny ve společném (*indexovém*) bloku, v tabulce indexů, kdy indexové bloky lze uspořádávat hierarchicky.

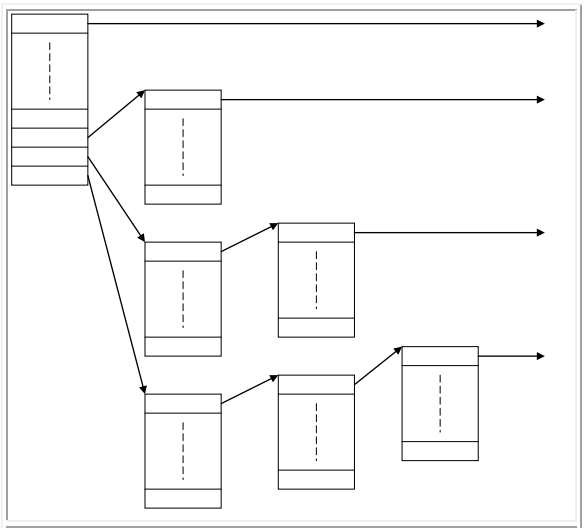
Vytváří se potřebná tabulka, tzv. indexový blok. Tato metoda je vhodná pro přímý a dynamický přístup bez externí fragmentace. Je však nutná jistá prostorová režie pro indexový blok. Index se při otevření souboru nahrává do operační paměti a tím je umožněna hierarchie indexů.

Pro efektivní správu volné paměti se používá bitová mapa, kdy pozice jednoho bitu odpovídá jednomu bloku (fyzické stránce) na disku. Jedná se o lineární transformaci pozice bitu na pozici bloku. Bitová mapa vyžaduje implementaci v samostatném diskovém prostoru. Například při délce bloku 4 KB a kapacitě disku 1 GB je délka bitové mapy

$$n = 1\text{GB} / 4\text{KB} = 32\ \text{KB},$$

což je zanedbatelné procento proti celkové kapacitě disku. Je nutno si uvědomit, že mapa musí být trvale uložena na disku, ale upravuje se v operační paměti a tedy kopie nemůže být kontinuálně konzistentní s originálem. Nelze tedy připustit, aby pro blok [*i*] platilo bit[*i*] = 1 v operační paměti a bit[*i*] = 0 na disku. Řešení je v postupném nastavování tj. bit[*i*] = 1 na disku, pak přiděl blok[*i*] a nastav bit[*i*] = 1 v operační paměti.

U indexové metody lze správa volné paměti realizovat i řetězením volných fyzických stránek a to z důvodu obtížného hledání souvislých prostorů na disku.



Indexová alokace

Kontrolní otázky:

1. Které informace operační systém udržuje o souborech?
2. Jaké existují organizace struktur adresářů?
3. Jaké existují přístupové metody k souborům?

Úkoly k zamyšlení:

1. Zamyslete se úlohou vyrovnávací paměti při ukládání a čtení souborů na disk.

Shrnutí obsahu kapitoly

V této kapitole jste se seznámili s principy souborového systému operačních systémů. Důraz v této kapitole byl kladen na pochopení způsobu ukládání souborů na disk.

9 Networking, správa sítí

V této kapitole se dozvíte:

- Proč se v modelech počítačových sítí používá rozdělení komunikujících systémů do vrstev?
- Jak vznikl referenční síťový model ISO/OSI a jaké má vlastnosti?
- Jaké jsou nejdůležitější síťové protokoly, resp. protokolové sady používané v počítačových sítích?
- Jaké jsou hlavní odlišnosti mezi prakticky používanými protokolovými sadami a referenčním modelem OSI/OSI?

Po jejím prostudování byste měli být schopni:

- Rozumět struktuře vrstevnatých modelů počítačových sítí a způsobu komunikace, která v nich probíhá.
- Charakterizovat jednotlivé vrstvy referenčního síťového modelu ISO/OSI a jim vymezené funkce.
- Charakterizovat základní vlastnosti protokolových sad NetBIOS/NetBEUI, SPX/IPX, TCP/IP.
- Specifikovat základní rozdíly mezi referenčním modelem OSI/OSI a síťovou architekturou TCP/IP.

Klíčová slova této kapitoly:

Síťový model, síťová architektura, model vrstevnatý, model referenční ISO/OSI, vrstva fyzická, vrstva linková, vrstva síťová, vrstva transportní, vrstva relační, vrstva prezentační, vrstva aplikační. Protokolová sada, NetBIOS/NetBEUI, SPX/IPX, TCP/IP

Doba potřebná ke studiu: 2 hodiny

Průvodce studiem

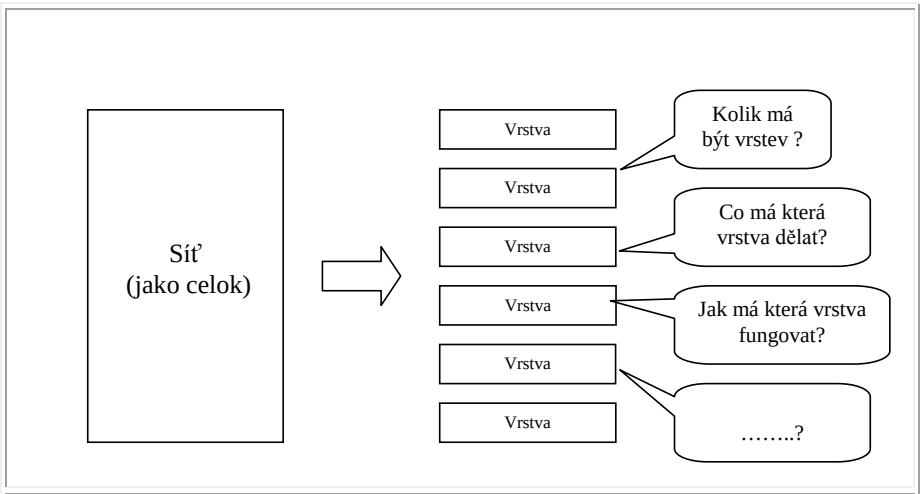
Cílem této kapitoly je seznámit studenty se zásadním pojmem z oblasti počítačových sítí, totiž pojmem síťový model, a s ním souvisejících pojmů. Za nejpodstatnější v této kapitole považujeme pochopení rozdělení funkcí obecné počítačové sítě mezi jednotlivé vrstvy referenčního modelu ISO/OSI.

Studium této kapitoly je poměrně náročné především z důvodu zavedení mnoha nových pojmů. Pro studium této kapitoly si v zájmu toho, abyste mohli důkladně promyslet a pochopit rozdělení funkcí sítí do jednotlivých vrstev modelu ISO/OSI, vyhraďte dostatek času.

Správa sítí zabezpečuje komunikaci procesorů, které nesdílejí ani fyzickou paměť ani hodiny synchronizující činnost procesoru. Každý procesor má svoji lokální paměť a lokální hodiny. Procesory jsou propojeny komunikační sítí a vlastní komunikace je řízena protokoly.

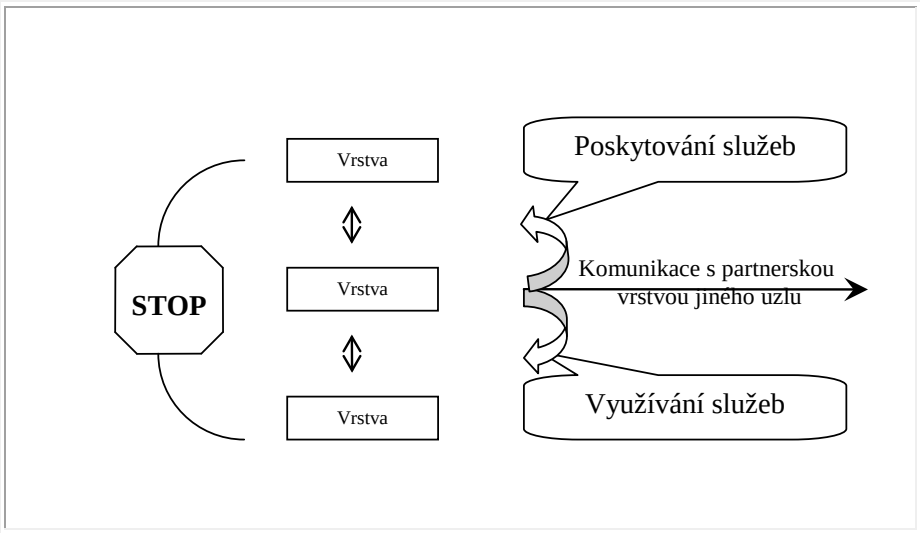
9.1 Síťový model a síťová architektura

Implementovat funkční síť je hodně složité a náročné, jedná se o obdobnou situaci jako při řešení velkých SW celků. Jde o jeden velký problém, který se vyplatí dekomponovat, tj. rozdělit na menší části, které je možné řešit samostatně. Dekompozice se provede po hierarchicky uspořádaných vrstvách, což dobře odpovídá povaze řešeného problému. Přináší to i další výhody jako je možnost alternativních řešení na úrovni nižších vrstev a větší modulárnost. Představa dekompozice je zřejmá z následujícího obrázku.

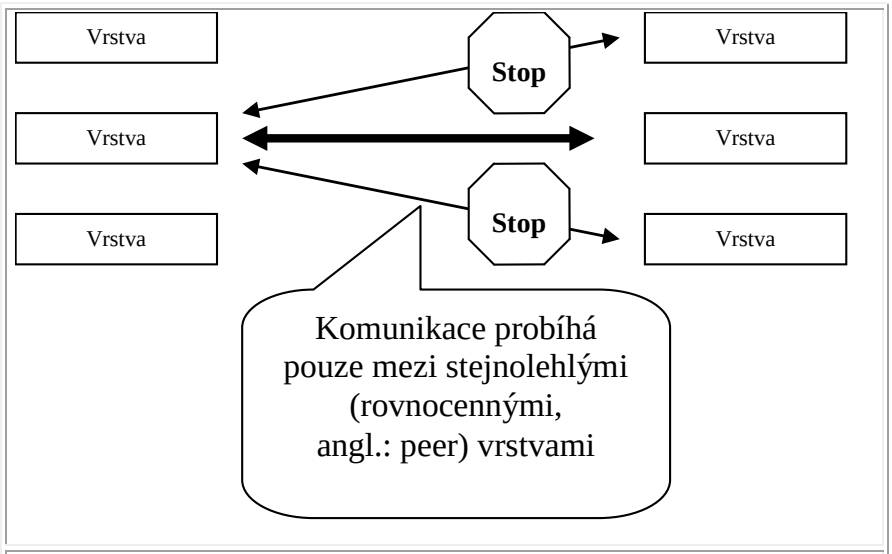


Dekompozice sítě na vrstvy

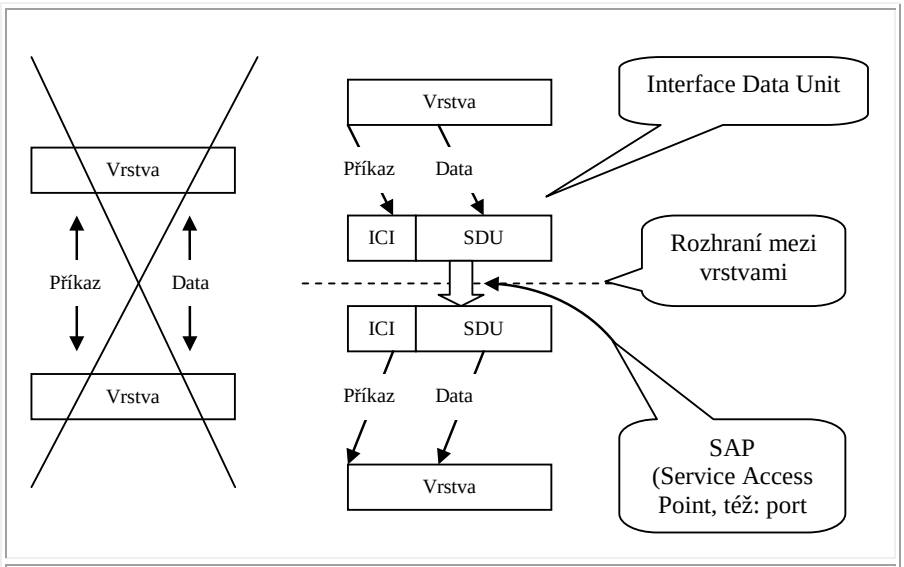
Způsob komunikace mezi vrstvami je zřejmá z následujících obrázků.



Komunikace mezi sousedními vrstvami

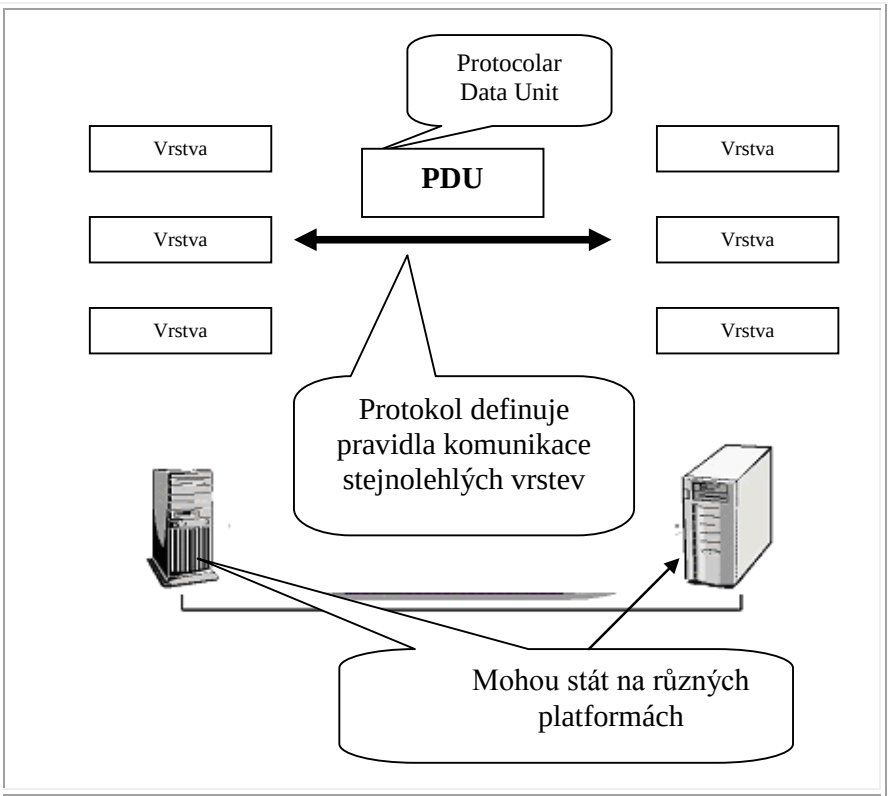


Komunikace mezi stejnolehlými vrstvami

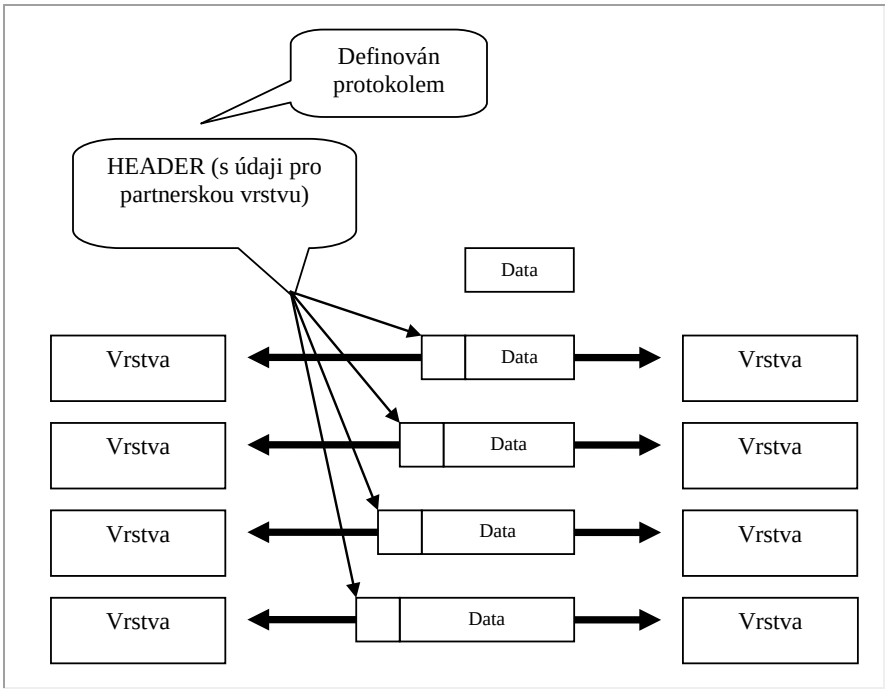


Definice protokolu

V této souvislosti je nutné vysvětlit pojem protokol. Protokol definuje pravidla komunikace na stejnolehlých vrstvách, přičemž komunikující subjekty mohou pracovat na různých platformách.



Definice protokolu



Protokoly na jednotlivých vrstvách

Z uvedených obrázků vyplývá, že vrstvy nejsou „jednolité“, v každé vrstvě může existovat a fungovat několik relativně samostatných entit. Pod pojmem entita můžeme chápat např. proces, démon, úlohu apod. Entity ve stejné vrstvě mohou plnit rozdílné funkce (nekonkurovat si) nebo plnit obdobné funkce (ale jiných způsobem, tj. konkurovat si). Protokol definuje pravidla komunikace

mezi entitami stejnolehlých vrstev. Každý protokol vždy „patří“ do určité konkrétní vrstvy a určuje způsob, jakým je realizována určitá služba. Pro každou vrstvu může existovat několik alternativních protokolů přičemž současné použití různých protokolů (v rámci téže vrstvy) se nemusí vylučovat.

Síťový model je ucelená představa o tom, jak mají být sítě řešeny. Zahrnuje představu o počtu vrstev a o tom, co má mít která vrstva na starosti. Nezahrnuje konkrétní představu o tom, jak má která vrstva své úkoly plnit. Síťová architektura obsahuje navíc také konkrétní představu o způsobu fungování jednotlivých vrstev, tj. obsahuje i konkrétní protokoly. Příkladem síťového modelu je referenční model ISO/OSI a příkladem síťové architektury je TCP/IP.

V několika uplynulých letech byla vyvinuta řada síťových norem. Některé rozhodující organizace v tomto oboru vytvořily protokoly nebo pravidla, které zajišťují kompatibitu pro síťový hardware a software různých výrobců.

Dosud jsme si všímali hlavních komponentů sítí LAN. Kdyby byly počítače, aplikační programy, síťový software a kabeláž vyrobeny stejným dodavatelem, nebyl by žádný problém zajistit, aby vše hladce spolupracovalo. Dnešní realita je však taková, že obvykle síťový software jednoho výrobce LAN nebude pracovat v síti jeho konkurenta, takže aplikační programy a dokonce i kabeláž musí být vybrány pro určitou síť LAN. Aby bylo dosaženo alespoň nějaké úrovně sjednocení mezi různými dodavateli sítí, vytvořila organizace ISO normy nazvané OSI (Open Systems Interconnection). Různé počítače vzájemně propojené do sítě musí vědět, v jaké formě budou přijímat informace. Kde je začátek určitého slova, kde je jeho konec a kde začne slovo následující? Má počítač možnost zjistit, zda byla zpráva při přenosu zkreslena? Model OSI odpovídá na tyto a další otázky pomocí řady norem, které by měly v budoucnu umožnit veřejnosti, aby si kupovala síťové prvky různých dodavatelů s určitou jistotou, že budou spolupracovat.

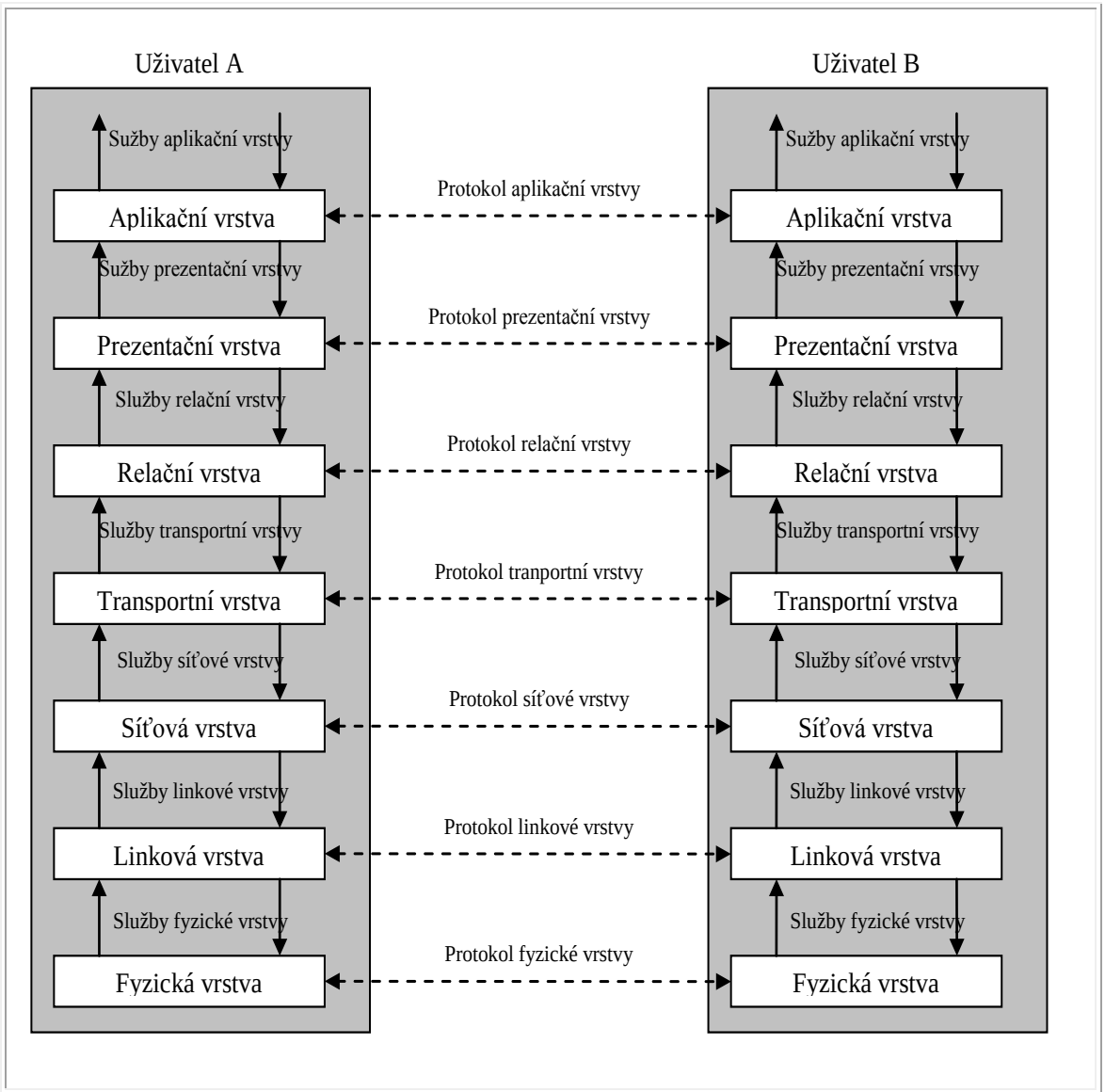
Referenční model ISO/OSI byl pokusem vytvořit univerzální síťovou architekturu. Skončil jako síťový model bez protokolů, které se dodělávaly postupně. Pochází „ze světa spojů“ od organizace ISO (International Standards Organization, správně: International Organization for Standardization) - členy ISO jsou národní normalizační instituce. Byl „oficiálním řešením“, dnes je prakticky odepsaný, neboť prohrál v souboji s TCP/IP.

Referenční model ISO/OSI reagoval na vznik proprietárních a uzavřených sítí IBM SNA. Prvotním záměrem bylo definovat, jak mají vypadat otevřené systémy. Odsud je název Open Systems Architecture. Chování „uvnitř“ a nejen „mezi sebou“ se ukázalo jako příliš náročné, došlo k redukci ambic. Proto nastalo druhé přiblížení, které se týkalo jen vzájemného propojení otevřených systémů a nastala změna názvu na Open Systems Interconnection Architecture. Opět se ukázalo jako příliš náročné. Proto nastalo třetí přiblížení, které neobsahovalo konkrétní protokoly, ale jen představu o počtu vrstev a o tom, co má která vrstva dělat. Poslední iterace názvu je tedy Open Systems Interconnection. Byly „odstraněny“ protokoly, zbyl jen síťový model (tedy obecný „rámec“), do kterého jsou konkrétní řešení „zasazována“ a konkrétní

protokoly pro referenční model ISO/OSI jsou vyvíjeny samostatně a teprve dodatečně jsou zařazovány do rámce ISO/OSI. Filosofie referenčního modelu ISO/OSI vznikala „od zeleného stolu“ a pak byla „nadiktován“ uživatelům. Vznikala maximalistickým způsobem a autoři se snažili zahrnout „vše, co by se někdy někomu mohlo hodit“. Výsledek byl dosti odtažitý od reálné praxe. Celá řešení se často ukázala jako nerealizovatelná a hledala se implementovatelná podmnožina, avšak vzájemně kompatibilní. Mnohé výchozí předpoklady se ukázaly jako chybné. Autoři ISO/OSI se dosti dlouho přeli o počtu vrstev. Nakonec zvítězil návrh na 7 vrstev, což se dnes zdá být zbytečně mnoho. Kritéria pro volbu vrstev byla stanovena následovně:

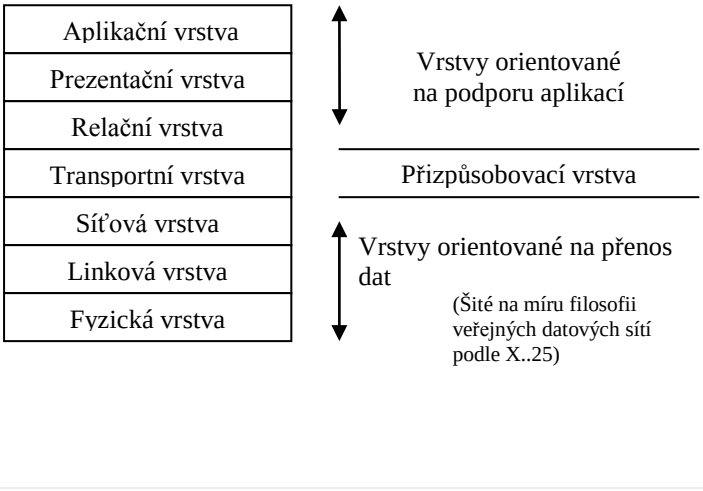
- činnosti na stejném stupni abstrakce mají patřit do stejné vrstvy,
- odlišné funkce by měly patřit do odlišných vrstev,
- aby bylo možné převzít již existující standardy,
- aby datové toky mezi vrstvami byly co nejmenší,
- aby vrstvy byly rovnoměrně vytíženy.

Standard OSI (Open Systems Interconnection) představuje model se sedmi vrstvami, který zajišťuje účinnou komunikaci v rámci sítě LAN a také mezi různými sítěmi. Model OSI se skládá ze sedmi vrstev specifikací, které popisují manipulaci s daty při jednotlivých fázích přenosu. Každá vrstva poskytuje určité služby vrstvě, jež je bezprostředně nad ní. Sedm vrstev ISO/OSI je na následujícím obrázku.



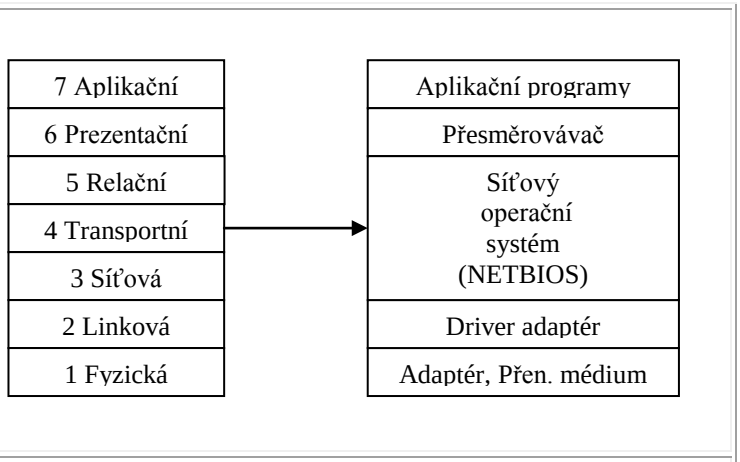
Vrstvy ISO/OSI

Orientace vrstev je zřejmá z následujícího obrázku.



Orientace vrstev ISO/OSI

Každá vrstva vykonává vlastní služby přičemž zajišťuje návaznost na sousední vrstvy.



Vrstvy modelu ISO/OSI

Model OSI přiřazuje složitým procedurám, nezbytným pro komunikaci v síti, sedm různých vrstev. Model je navržen tak, aby bylo možno snadno dosáhnout výchozí dohody na nižších vrstvách a konečně na všech sedmi vrstvách.

9.1.1 Fyzická vrstva

Standardy fyzické vrstvy se týkají technických norem zajišťujících kompatibilitu sítí. Zahrnují použité napěťové úrovně, časování přenosu dat a mechanismy vzájemného potvrzování.

První vrstva (fyzická) představuje řadu pravidel týkajících se technického vybavení užitého pro přenos dat. Zabývá se takovými věcmi, jako jsou napěťové úrovně, časování přenosu dat a pravidla pro komunikaci, sloužící k

vytvoření spojení. Fyzická vrstva určuje, zda mají být bity odesílány v poloduplexním režimu (to je podobný způsob, jakým se data předávají pomocí radiostanice) nebo v plně duplexním režimu, což vyžaduje současné vysílání i příjem dat. Další technická specifikace, kterou pokrývá fyzická vrstva, zahrnuje konektory a rozhraní na přenosová média. Na této úrovni se model OSI zabývá elektrickými veličinami a bity (nulami a jedničkami). Bity nemají na této úrovni žádný skutečný význam. Přiřazení významu provádí až další vrstva OSI. Fyzická vrstva se zabývá výhradně přenosem bitů (bez ohledu na jejich význam), otázkami typu kódování, modulace, časování, synchronizace, elektrickými parametry signálů, konektory, řídícími signály rozhraní. Nabízí služby typu přijmi bit, odešli bit a musí zajistit, že v případě vyslání jedničkového bitu jej druhá strana přijme jako jedničkový a ne jako nulový. Nijak neinterpretuje to, co přenáší a jednotlivých bitům nepřisuzuje žádný specifický význam. Na úrovni fyzické vrstvy rozlišujeme paralelní a sériový přenos, synchronní, asynchronní a arytmický přenos, přenos v základním a přeloženém pásmu. Standardem fyzické vrstvy je RS-232-C, V.24, X.21.

9.1.2 Linková (spojová) vrstva

Spojová vrstva se zabývá shlukováním dat do rámců (frame), v nichž jsou přenášena.

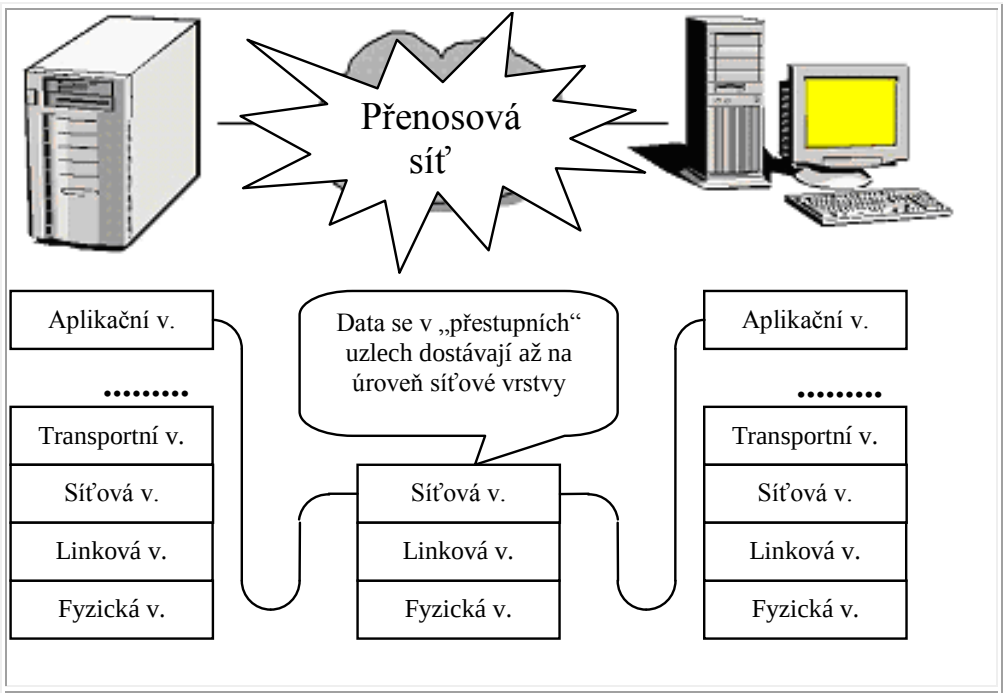
Model OSI je navržen tak, že každá vrstva poskytuje vyšší vrstvě určitý klíčový prvek. Fyzická vrstva předává spojové vrstvě bity. A nyní přichází okamžik, kdy je těmto bitům nutno dát nějaký význam. V tomto bodě se už nezabýváme bity, nýbrž datovými rámci (pakety) obsahujícími jak data, tak řídící informace. Spojová vrstva přidává na začátek a na konec zprávy značky (flag). Normy této vrstvy provádějí dvě důležité funkce. Zaručují, že data nejsou zaměněna za značky a prověřují výskyt chyb uvnitř datového rámce. Toto prověřování chyb může probíhat tak, že se na stranu přijímače odešle informace o určitém datovém rámci a pak se čeká na potvrzení, že vše bylo správně přijato. Linková vrstva (spojová vrstva) přenáší celé bloky dat tzv. rámce (frames) a zajišťuje přenos pouze v dosahu přímého spojení, tj. bez „přestupních stanic“. Může fungovat spolehlivě či nespolehlivě, spojovaně či nespojovaně a může využívat různé přenosové technologie, tj. linkové i bezdrátové. Úkolem je synchronizace na úrovni rámců (správné rozpoznání začátku a konce rámce, i všech jeho částí), zajištění spolehlivosti (detekce chyb a náprava), řízení toku (aby vysílající nezahltil příjemce) a přístup ke sdílenému médiu (řeší konflikty při vícenásobném přístupu ke sdílenému médiu). Linková vrstva řídí přístup uživatele na síť a tvoří obálku paketů. Úkolem je změnit prosté komunikační vybavení na spoj, který se vůči síťové vrstvě chová jako bezchybný.

9.1.3 Síťová vrstva

Síťová vrstva se zabývá přepojováním paketů. Vytváří virtuální spojení pro datovou komunikaci mezi počítači nebo terminály.

Třetí vrstva modelu OSI, síťová vrstva, se zabývá přepojováním paketů. Vytváří virtuální spojení (trasu mezi dvěma počítači nebo terminály) pro

datovou komunikaci. U odesílatele zformuje síťová vrstva zprávu z transportní vrstvy do datových paketů, které pak mohou dvě nižší vrstvy přenášet. Na straně přijímače zrestauruje síťová vrstva původní zprávu. Abychom pochopili použití datových paketů, bude nezbytné podívat se na průmyslový standard, zahrnující spodní tři vrstvy modelu OSI. Je to standard X.25. Síťová vrstva přenáší bloky dat označované jako pakety (packets) a zajišťuje doručení paketů až ke konečnému adresátovi. V prostředí, kde není přímé spojení, hledá vhodnou cestu až k cíli tj. zajišťuje tzv. směrování (routing). Musí si uvědomovat skutečnou topologii celé sítě (obecně), přičemž používá různé algoritmy směrování jako adaptivní či neadaptivní, izolované či distribuované. Je poslední vrstvou, kterou musí mít přenosová infrastruktura.



Přenosová infrastruktura

Hlavním úkolem síťové vrstvy je určení směřování dat (paketů) v síti ze zdroje do cíle a musí řešit problematiku zahlcení sítě, ke které může dojít při velkém množství paketů. Bývá v ní zabudována také účtovací funkce ACOUNT ke zjištění, kolik který uživatel vyslal bitů.

9.1.4 Transportní vrstva

Prvotní úlohou transportní vrstvy je rozpoznání chyb a zotavení po chybě, zabývá se však také multiplexem zpráv a regulací toku informací.

Transportní vrstva modelu OSI má mnoho funkcí včetně několika úrovní rozpoznávání chyb a zotavení po chybě. Na nejvyšší úrovni může transportní vrstva rozpoznávat (či dokonce opravovat) chyby, odhalovat pakety, které byly odeslány v nesprávném pořadí, a přerovnávat je do pořadí správného. Tato vrstva také multiplexuje několik zpráv do jednoho spoje a vytváří hlavičky určující kterému spoji která zpráva patří. Transportní vrstva také reguluje tok informace tím, že řídí pohyb zpráv. Transportní vrstva zajišťuje komunikaci mezi koncovými účastníky (end-to-end komunikaci) přičemž může měnit

nespolehlivý charakter přenosu na spolehlivý, méně spolehlivý přenos na více spolehlivý, nespojovaný přenos na spojovaný. Přitom vychází ze skutečnosti, že nelze „hýbat“ s vlastnostmi a funkcemi nižších vrstev. Vzhledem k tomu, že vyšší vrstvy mohou chtít něco jiného, než co nabízí nižší vrstvy, je úkolem transportní vrstvy zajistit potřebné přizpůsobení. Transportní vrstva rozkládá data na menší části tzv. pakety, řeší problematiku komunikace koncových uživatelů (end-to-end), přičemž využívá ke komunikaci záhlaví a řídící zprávu (transportní záhlaví).

9.1.5 Relační vrstva

Relační vrstva se věnuje řízení sítě. Ovládá rozpoznávání hesel, procedury při hlášení a odhlášení a rovněž dohlížení nad sítí a vytváření přehledových zpráv.

Dosud jsme viděli, že model OSI se zabývá jenom bity a datovými zprávami a nerozlišuje jednotlivé uživatele v síti. Relační vrstvu si můžeme představit jako vrstvu, jejíž úlohou je řízení sítě. Je schopna přerušit určitou relaci a řídí řádné ukončení relací. Uživatel komunikuje přímo s touto vrstvou. Relační vrstva může prověřovat heslo zadané uživatelem a umožnit uživateli, aby přepnul z poloduplexního přenosu na plně duplexní. Dokáže určit, kdo hovoří, jak často a jak dlouho. Řídí přenosy dat a zodpovídá rovněž za zotavení systému po jeho výpadku. Konečně relační vrstva může sledovat využití systému a účtovat uživatelům spotřebovaný čas. Relační vrstva zajišťuje vedení relací. Relace může zajišťovat synchronizaci, šifrování a podporu transakcí. Relační vrstva kontroluje přístup uživatele a jeho programů na síť, dovoluje spojení uživatelů na různých typech zařízení, kteří mají mezi sebou zavedeny tzv. relace (session) a dovoluje, aby se uživatel mohl přihlásit ve vzdáleném víceuživatelském systému a přenesl soubor mezi dvěma počítači. Řeší řízení dialogu mezi jednotlivými počítači.

9.1.6 Prezentační vrstva

Bezpečnost sítě, přenosy souborů a formátovací funkce jsou úkoly prezentační vrstvy.

Prezentační vrstva modelu OSI se věnuje bezpečnosti sítě, přenosu souborů a formátovacím funkcím. Na bitové úrovni je prezentační vrstva schopna kódovat data v řadě různých formátů včetně ASCII a EBCDIC. Americký standardní kód pro výměnu informací (ASCll) je kód pro přenos dat používající 7 bitů pro kódování znaku a osmý bit jako paritní. Je to kód používaný nejobecněji. Mnoho větších počítačů IBM používá rozšířený dvojkově kódovaný desítkový kód (EBCDIC - Extended Binary Coded Decimal Intechange Code). Prezentační vrstva musí být schopna použít pro přenos dat libovolný z těchto standardů. Prezentační vrstva má na starosti potřebné konverze z různých kódování znaků (ASCII, EBCDIC,...), formátu čísel, formátu struktur, polí a ukazatelů (pointerů). Nižší vrstvy se snaží doručit každý bit přesně tak, jak byl odeslán a přitom stejná posloupnost bitů může mít pro příjemce jiný význam než pro odesilatele. Prezentační vrstva určuje tvar dat v jakém jsou dostupné uživateli. Přesměrovává požadavky uživatele na síť, provádí kódování jednotlivých informací v paketech (kryptografie z hlediska

utajení před nepovolanými uživateli - ASCII, EBDIC), způsob komprese a zhuštění informací pro zmenšení počtu přenášených bitů.

Pro dosažení komunikace musí prezentační vrstva v obou vzájemně komunikujících počítačích obsahovat tytéž protokoly neboli pravidla pro manipulaci s daty. Tato vrstva provádí rovněž konverzi protokolů mezi různými počítači používajícími různé formáty. Většina funkcí textových procesorů, které spojujeme s formátováním textu (stránkování, počet linek na obrazovce, pohyby kurzoru po obrazovce) je rovněž náplní prezentační vrstvy.

Práce s terminály, které mají nekompatibilní kódy, je rovněž zajišťována touto vrstvou. Terminálový protokol řeší odlišnosti tak, že umožní každému datovému terminálu aby fungoval jako stejný virtuální terminál. Výsledkem tohoto postupu je, že existuje řada překladových tabulek fungujících mezi místním a vzdáleným terminálem. Lokální terminál vysílá datovou strukturu, která definuje okamžitý obsah jeho obrazovky v takových termínech, jako je zobrazený počet znaků na řádce. (Tento počet se může podstatně lišit. Mnoho terminálů zobrazuje 132 znaků na řádek, ale existují i jiné formáty.) Tato datová struktura přichází do řídícího prvku vzdáleného terminálu a ten převede tento počet na takový kód, kterému terminál rozumí a může jej použít. Další kódy se používají pro tučné písmo, podtržení, grafiku atd.

9.1.7 Aplikační vrstva

Síťové programy nacházející se v aplikační vrstvě zahrnují elektronickou poštu, řízení databází, software pro file servery a print servery. Aplikační vrstva zpracovává hlášení, vzdálená přihlašování a zodpovídá za statistiku řízení sítě. Na této úrovni najdete programy pro řízení databází, elektronickou poštu, programy pro file servery, print servery a příkazy operačního systému.

Ve většině případů jsou funkce vykonávané touto vrstvou závislé na uživateli. Vzhledem k tomu, že různé uživatelské programy mají různé požadavky, je obtížné zevšeobecňovat protokoly, které se zde nalézají. Některá odvětví, např. bankovnictví, vyvinula na této úrovni řadu vlastních standardů. Aplikační vrstva původně měla obsahovat aplikace. Problémem je ale velké množství aplikací a ty by musely být všechny standardizovány, což by nemělo ani smysl. Později bylo definováno pouze „jádro“ aplikací, které má smysl standardizovat. Například přenosové mechanismy elektronické pošty a ostatní části aplikací (typicky uživatelská rozhraní) byly vysunuty nad aplikační vrstvu. Aplikační vrstva představuje vlastně úroveň aplikačních programů např. elektronická pošta, síťové databázové systémy, vytváří všeobecně platné protokoly pro přenos údajů mezi nekompatibilními terminály, definuje pohyby kurzoru po obrazovce s možností vytvoření virtuálního terminálu.

9.2 Síťové protokoly

Bloky dat předávané mezi koncovými účastníky obvykle označujeme jako **pakety**. V jejich formátech najdeme síťové adresy obou koncových účastníků a informace potřebné pro potvrzování a případně i řízení toku. Pakety mohou být předávány jako zcela nezávislé datagramy, nebo jako součást souvislejší

komunikace po virtuálním kanále. Mezi nejdůležitější a nejpoužívanější síťové protokoly v lokálních sítích patří:

- NetBIOS, NetBEUI,
- IPX/SPX,
- TCP/IP.

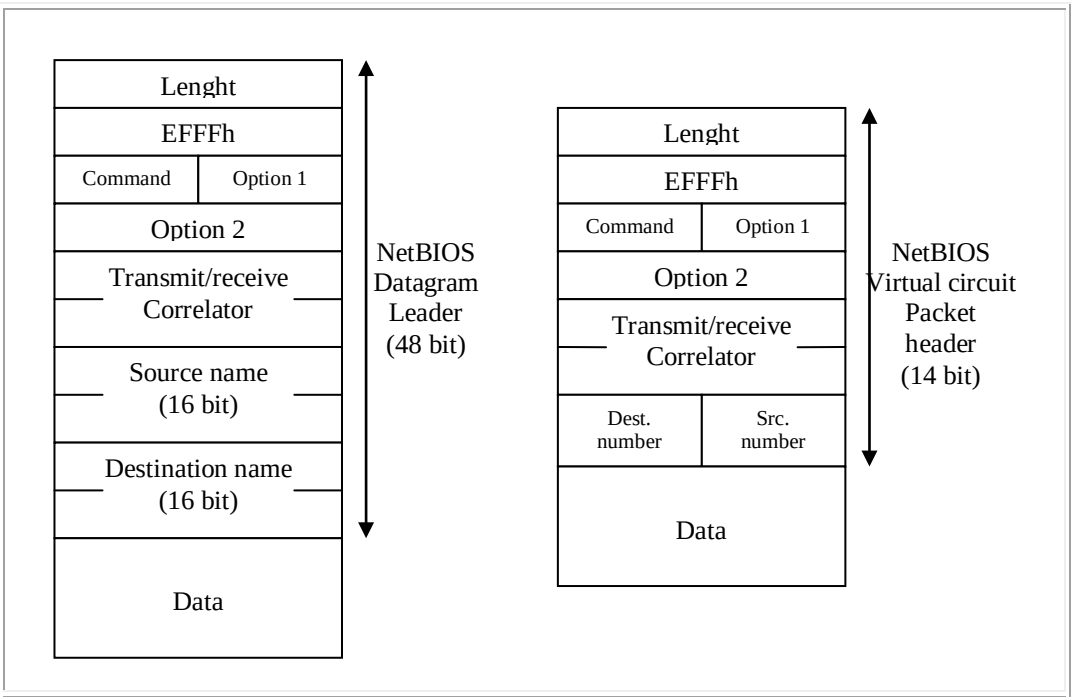
9.2.1 Protokoly NetBIOS a NetBEUI

Nejstarším síťovým protokolem určeným specificky pro prostředí lokální sítě (kde existuje možnost, aby rámec odeslaný jednou ze stanic sítě byl přijat všemi ostatními stanicemi) je **NetBIOS** navržený firmou IBM. Aplikace se identifikuje jménem a protokol pro správu NetBIOSu se stará o jedinečnost tohoto jména v síti. NetBIOS byl přímo vázán na ovladač komunikačního řadiče, stejným způsobem je implementován v LAN Manageru, kde je rozšířen, doplněn uživatelsky lepším rozhraním a pojmenován **NetBEUI** (NetBIOS Extended User Interface).

Aplikace musí pro vyžádání funkce NetBIOSu připravit požadavkový blok **NCB** (Network Control Block), ve kterém zadává parametry volání - jména, číslo logického kanálu, adresu a délku předávaných dat, časové limity pro vyslání a příjem. Požadavek předá aplikace NetBIOSu voláním systému (přerušení $5C_H$). Po předání požadavku může aplikace pokračovat ve své činnosti. Ukončení požadavku může aplikace aktivně testovat nebo lze ukončením požadavku aktivovat dokončovací rutinu.

Komunikující aplikace (nebo její komunikační kanály) jsou identifikovány jmény, která mají délku 16 znaků. Jméno může být buď individuální (a jedinečné v určité síti) nebo skupinové.

Základní služba, kterou NetBIOS podporuje, je datagramová služba (u protokolu NetBEUI jí odpovídá služba *MailSlot*) dovolující předání zprávy o délce do 512 Bytů jednomu adresátovi nebo libovolné stanici na síti, která takovou zprávu očekává (Broadcast). Virtuální kanály (u NetBIOSU relace, u NetBEUI služba Named Pipes) dovolují přenášet zprávy o délce 131071 znaků, které jsou při přenosu děleny do paketů. Na dalším obrázku je struktura paketů protokolu NetBIOS.

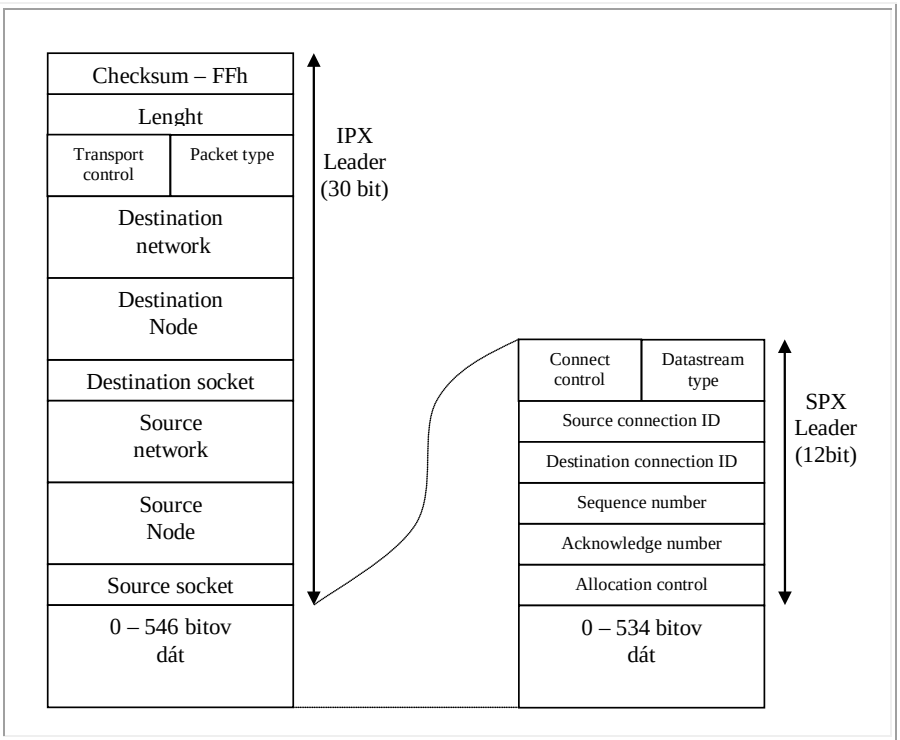


Pakety protokolu NetBIOS

9.2.2 Protokol IPX/SPX

S protokoly **IPX/SPX** (Internet Packet eXchange/Sequential Packet eXchange) komunikuje síťový operační systém Novell Netware. Protokoly vycházejí ze systému **XNS** (Xerox Network System), který byl alternativou firmy Xerox k protokolům TCP/IP. Protokol *IPX* zajišťuje přenos paketů bez potvrzování mezi aplikacemi připojenými na zvolená připojovací místa (Socket). Protokol *SPX* je nadstavbou protokolu IPX, zajišťuje potvrzování přenesených paketů a umožňuje práci více aplikačních procesů na jednom portu.

Komunikační funkce IPX/SPX vyžadují, aby aplikace uložila potřebné parametry do požadavkového bloku **ECB** (Event Control Block), obsluha požadavků může být asynchronní k dalšímu běhu aplikace. Aplikace může na ukončení požadované funkce aktivně čekat nebo může být přerušena dokončovací rutinou. Následující obrázek ukazuje strukturu paketů protokolů IPX a SPX.



Pakety protokolů IPX a SPX

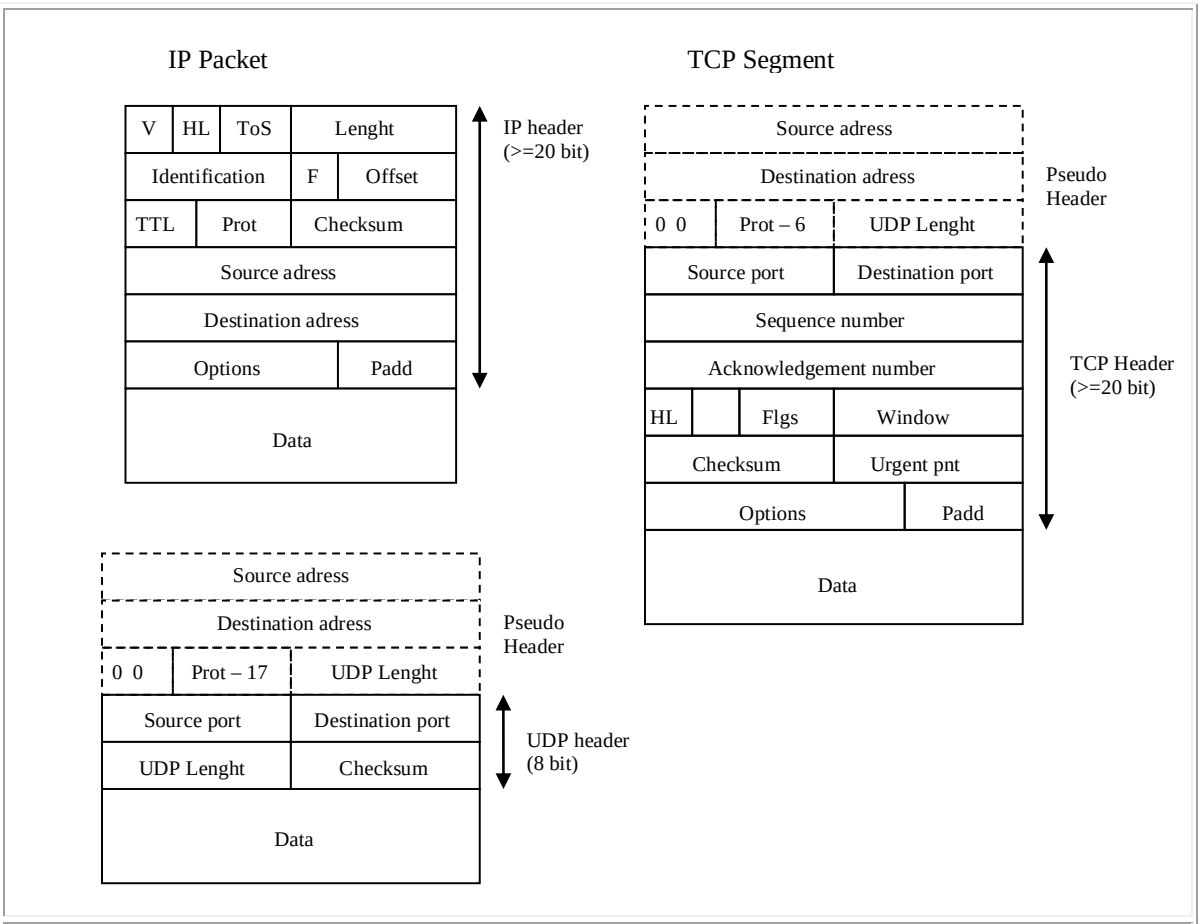
Výhodou protokolů IPX/SPX je adresace, která vychází z adresace stanic v lokální síti. Adresa je v IPX definována jako dvojice (32bitová adresa sítě, 48bitová adresa stanice), to zjednodušuje práci směrovačů ale i stanic v síti. Podstatnou nevýhodou IPX/SPX je skutečnost, že adresu sítě definuje správce konkrétní sítě. Chybějící kooperace v přidělování adres v principu znemožňuje vzájemné propojení sítí protokoly IPX/SPX mezi sebou.

9.2.3 Protokol TCP/IP

Protokoly **TCP/IP** jsou v současnosti akceptovány jako standard pro komunikaci v rozsáhlých počítačových sítích. Architektura TCP/IP zahrnuje vlastní přenos paketů *IP* (Internet Protocol), jednoduché datagramové rozhraní **UDP** (User Datagram Protocol) a protokol logického kanálu **TCP** (Transmission Control Protocol). Protokol TCP zajišťuje potvrzování v prostředí propojených sítí, ve kterých mohou být pakety dodávány v nezaručeném pořadí, mohou být štěpeny na *fragmenty* a mohou se ztrácet. Je vybaven důmyslným řízením toku a ochranou proti chybám vyvolaným opakovaným navazováním spojení. Aplikacím viditelné protokoly IP, UDP a TCP jsou podporovány služebními protokoly, které zajišťují transformace adres TCP/IP na adresy lokální sítě (ARP, RARP), řízení sítě (ICMP) a podporu směrování (RIP, OSPF).

Aplikační rozhraní protokolů TCP, IP a UDP jsou poměrně přesně definována v systémech UNIX jako *BSD sokety* (BSD Sockets) nebo jako rozhraní *TLI* (Transport Layer Interface). Rozhraní v systémech Windows je obdobou BSD soketů doplněné o podporu asynchronního provádění funkcí.

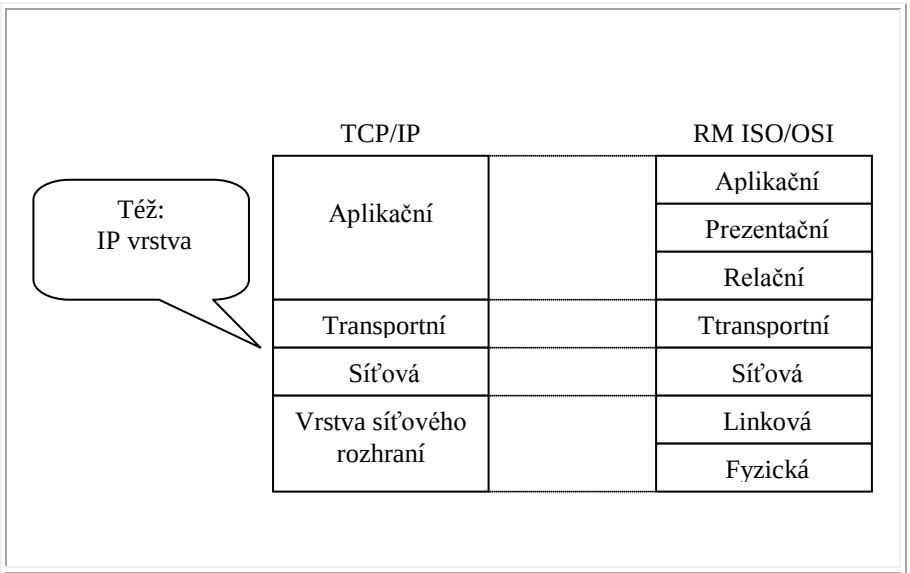
Funkce rozhraní zahrnují vytváření (Socket) a rušení (Close) datových struktur řídících komunikaci na daném přípojném místě (portu) nebo po virtuálním kanále, jejich vazbu na logický kanál, vazbu na adresační informaci (Bind) a limit počtu neobsloužených požadavků na vstupu (Listen). Součástí rozhraní TCP jsou funkce pro pasivní a aktivní otevření kanálu (Accept a Connect) a pro jeho uzavření (Close). Přenos paketů a zpráv zajišťující volání funkcí Write a Read, spolu s několika formami funkcí Send a Receive. Formát IP paketů, UDP datagramů a TCP segmentů je na následujícím obrázku.



Formáty paketů protokolů IP, TCP a UDP

TCP/IP je ve skutečnosti síťová architektura, která obsahuje ucelenou představu o počtu, úloze vrstev a obsahuje i konkrétní protokoly. Historie vzniku TCP/IP souvisí s Internetem (ARPANETem) postaveným na „prozatímní“ paketové technologii NCP (Network Control Protocol). Cílem bylo ověřit životaschopnost paketové technologie. Protože protokol NCP nebyl vhodný pro rutinní používání, byl TCP/IP vyvíjen jako „definitivní“ řešení pro vznikající Internet. Filosofie, uplatněná při vzniku TCP/IP vycházela ještě z původních požadavků na ARPANET (který byl navržen pro vojáky), že nesmí mít žádnou centrální část (tu by nepřítel zničil jako první). Charakterem musí být převážně decentralizovaný, musí být velmi robustní (tak aby to aspoň nějak fungovalo, když nepřítel část sítě odstřelí) a komunikace bude mít spíše nespojovaný charakter.

Filosofie TCP/IP je řešena tak, aby šlo snadno připojovat dříve samostatné sítě a různé síťové technologie. Požadavkem bylo, aby přenosová část hlavně přenášela data a ne se starala o další věci. Spolehlivost si zajistily až koncové uzly a nikoli přenosová část sítě. Proto přenosová část (síťová vrstva) funguje pouze nespolehlivě a mechanismy zajišťující spolehlivost jsou implementovány až v transportní vrstvě (ale jako volitelná možnost - tj. není povinnost je využívat). Otázkou je, proč je výhodnější, aby si spolehlivost zajišťovaly až koncové uzly? Některé aplikace nemusí spolehlivost potřebovat a dají přednost rychlému a pravidelnému přenosu. Spolehlivost je vždy relativní (nikoli 100%), někomu by nemusela postačovat míra „zabudované“ spolehlivosti a musel by si ji zajišťovat sám a znovu. To by bylo neefektivní, protože režie by se sčítala. Protože k zajištění spolehlivosti je třeba výpočetní kapacita (ta je lacinější v koncových uzlech než „uvnitř“ sítě), autoři TCP/IP se nespolehlivých služeb nebáli. Filosofie uplatněná při vzniku TCP/IP, tj. maximální robustnost vedla k nespojovanému charakteru (aby při výpadku nebylo nutné složitě rušit stávající a navazovat nová spojení). Komunikace by měla mít spíše bezestavový charakter (aby se nemusela uchovávat žádná stavová informace o dosavadním průběhu komunikace), tzn. aby nebylo nutné se složitě zotavovat z případného výpadku. Další filosofií, uplatněnou při vzniku TCP/IP, je sdílení mechanismů. TCP/IP to řeší tak, aby režii nenesli ti, kdo je nechtějí používat tj. ne-sdíleně. Zabudovávají se přímo a pouze do těch aplikací, které je skutečně potřebují. Režii nenese ten, kdo mechanismy nepotřebuje. Koncepce vrstev TCP/IP je zřejmá z obrázku.



Koncepce TCP/IP

Aplikační vrstva TCP/IP má koncepci obdobnou aplikační vrstvě ISO/OSI. Jsou zde základní části aplikací, ostatní (UI) jsou nad aplikační vrstvou. Služby relačního a prezentačního charakteru jsou přímou součástí aplikací. Původní služby aplikační vrstvy jsou elektronická pošta, přenos souborů a vzdálené přihlašování. Později vznikají další, jako sdílení souborů, správa sítě, zpřístupnění informací, WWW.

Transportní vrstva TCP/IP řeší komunikaci koncových účastníků (end-to-end communication). Sama využívá nespojovaný a nespolehlivý přenos na úrovni síťové vrstvy. Alternativně nabízí spojovaný a spolehlivý přenos. Aplikace si mohou vybrat dle vlastního uvážení. Protokol TCP (Transmission Control Protocol) zajišťuje spolehlivý a spojovaný přenos, „tváří se“ jako proud (stream), který přenáší jednotlivé byty. Protokol UDP (User Datagram Protocol) zajišťuje nespojovaný a nespolehlivý přenos. Je jen „lehkou nadstavbou“ nad síťovou vrstvou, nemění povahu přenosových služeb síťové vrstvy.

Síťová vrstva TCP/IP zajišťuje pouze nespojovaný a nespolehlivý přenos tj. „holé minimum“, ale snaží se být co nejrychlejší. Zajišťuje jednotné přenosové služby nad všemi možnými přenosovými technologiemi nižších vrstev, vytváří „jednotnou pokličku“ .

Protokol IP (Internet Protocol) je hlavní (a jediný) přenosový protokol, snaží se zakrývat specifika přenosových technologií nižších vrstev a fungovat nad nimi optimálně. Jsou varianty jako SLIP (Seriál Line IP), PPP (Point-to-point protokol).

Vrstva síťového rozhraní (Network Interface Layer) zahrnuje „vše pod síťovou vrstvou“. TCP/IP tuto vrstvu samo nijak nenaplňuje, tj. nespecifikuje svoje vlastní přenosové technologie na nejnižších vrstvách. Předpokládá se, že zde se použije to, co vznikne někde jinde (mimo rámec TCP/IP), například Ethernet, Token Ring, ATM. TCP/IP se zabývá pouze tím, jak tyto technologie co nejlépe využít, jak nad nimi provozovat IP.

Snaha překrýt odlišné přenosové technologie jednotnou „pokličkou“ naráží na problémy s adresami. Např. Ethernet používá 48-bitové adresy, ARCnet 8-bitové atd. (tj. linkové adresy jsou hodně odlišné). Protokol IP používá 32-bitové adresy (tzv. IP adresy). Protokol IP sám data fyzicky nepřenáší (ale využívá těch přenosových technologií, které jsou „pod ním“). Služby „pod“ protokolem IP mohou být dosti různorodé, zejména z hlediska:

- velikosti přenášených bloků (rámců),
- míry spolehlivosti,
- přenosového zpoždění,
- spojovaného či nespojovaného charakteru,
- charakterem přenosu (lze dělat broadcasting?).

Protokol IP se zaměřuje na „holé minimum“. Standardy TCP/IP jsou skutečně otevřené, i když nikdo pořádně neví, co to přesně znamená. Nejsou „v rukou“ jediné firmy, vznikají (jsou přijímány) na základě všeobecného konsensu. Specifikace těchto protokolů jsou veřejným vlastnictvím, za jejich využití se neplatí žádné licenční poplatky, texty specifikací mají povahu volně šiřitelných dokumentů (dokumenty RFC - Request for Comment). Standardy vznikají v rámci „sdružení“ IETF (Internet Engineering Task Force), což je dosti volné společenství odborníků, zainteresovaných na vývoji TCP/IP. Nad IETF stojí „dozorčí rada“, IESG (Internet Engineering Steering Group), která řídí práci jednotlivých skupin v rámci IETF a nad IESG stojí IAB (Internet Architecture Board), která vydává vlastní standardy. V současnosti vlastní technická řešení vznikají spíše u firem a firmy předkládají své řešení k IETF. Když je řešení

uznáno za potřebné a vhodné, může být přijato jako standard Internetu a tím se standardizované řešení stává „veřejným vlastnictvím“. Každý standard má formu dokumentu RFC. V současnosti existují řádově tisíce dokumentů RFC, avšak ne všechny dokumenty RFC jsou standardy! Většina má povahu informačních materiálů, návodů, doporučení. Dokumenty RFC jsou volně šiřitelné a nikdy se nemění, jejich obsah zůstává vždy stejný, lze je snadno archivovat a šířit, nikdy nevznikají neaktuální verze a když je potřeba nějaké řešení změnit, vydá se nový dokument RFC a ten ve své hlavičce prohlásí předchozí dokument RFC za „přežitý“ (obsolete). K jedné a téže problematice se v různých okamžicích mohou vztahovat různé dokumenty RFC. Každý dokument STD se vždy vztahuje k určité konkrétní problematice. Obsahem dokumentu STD je ten dokument RFC, který v daném okamžiku řeší příslušnou problematiku. Koncepce protokolů TCP/IP vznikla zhruba v letech 1977-8. Internet přešel na TCP/IP k 1.1.1983 a od té doby se koncepce nijak principiálně nezměnila. Další vývoj má charakter zdokonalování a obohacování, přibývají zejména nové služby jako NFS Gopher, WAIS, WWW a další. Stávající služby se obohacují o další možnosti např. el. pošta o podporu netextových formátů (standard MIME). V posledním období je TCP/IP silně kritizován a to z několika pohledů:

- **Internet není bezpečný**, neboli protokoly TCP/IP nezajišťují takovou míru bezpečnosti, jakou by si (někteří) uživatelé představovali. Je však faktem, že při vzniku koncepce TCP/IP nebyl požadavek zabezpečení vznesen a autoři se soustředili hlavně na efektivnost přenosů. Malá míra bezpečnosti spočívá v tom, že přenášená data nejsou šifrována, ani jinak zabezpečena proti odposlechu a zneužití. Protokol IP nijak nešifruje to, co přenáší. V době akademického Internetu malá úroveň bezpečnosti nevadila, avšak v době komerčního využití to vadí hodně! Změna by znamenala úpravu protokolu IP! Již dnes ale existují možnosti zvýšení míry zabezpečení a to šifrováním na aplikační úrovni kdy aplikace si sama zabezpečí data ještě před jejich odesláním. Jiný způsob je pomocí zabezpečených tunelů, kdy se vytváří zabezpečené virtuální kanály (tunely) a přenášená data (IP pakety) se zašifrují, vloží do normálních IP paketů a takto přenesou.
- **Filosofie TCP/IP nepočítá s mobilitou uživatelů,** tzn. že nelze mít jednu IP adresu a cestovat s ní po světě. IP adresy jsou vázány na „geografické“ (topologické) umístění. Přímé využití mobilních komunikačních technologií je problematické např. GSM (musí se obcházet tím, že se GSM využívá stejně jako běžná telefonní síť).
- **Nedostatek adres**, kdy autoři TCP/IP zvolili 32-bitový adresový prostor (IP adresy jsou 32-bitové). Přitom autoři pamatovali na existenci různě velkých sítí které potřebují různé počty adres a vznikly třídy A, B, C, ... Původní způsob přidělování adres z tohoto adresového prostoru nebyl příliš hospodárný. Autoři ani tak neudělali chybu, jako spíše nedocenili obrovitost svého úspěchu, neodhadli, jak ohromný zájem bude o připojování k Internetu. Problémem je hrozící vyčerpání IP adres, který se začal řešit dočasnými opatřeními, mající za úkol zpomalit úbytek adres. Tím je mechanismus CIDR (řeší i další problémy), tzv. privátní IP adresy. Zásadním koncepčním řešením, které by problém definitivně odstranilo je nalezení nové verze protokolu IP (IPnG, IP next Generation, resp. IPv6).

- **Charakter přenosu** - přenosové protokoly TCP/IP jsou orientovány na přenos elektronické pošty, souborů apod. , mají „dávkový“ (nárazový) charakter, negarantují, za jak dlouho jsou data doručena, ani s jakou pravidelností budou doručovány jednotlivé části dat. To velmi vadí multimediálním přenosům, přenosům živého zvuku a obrazu. Zvyšováním přenosových kapacit lze nepříznivý efekt zmírnit, ale ne odstranit zcela. Díky tomu se stalo možné například telefonování po Internetu. Řešením jsou až specializované přenosové protokoly, podporující přenos v reálném čase RTP a RSVP.

9.2.4 Srovnání TCP/IP a referenčního modelu ISO/OSI

Protokol TCP/IP a referenční model ISO/OSI lze porovnat ve třech oblastech:

- ISO/OSI a jeho součásti vznikají stylem „od složitého k jednoduššímu“, nejprve se požaduje hodně, a pak se musí ubírat, vznikají problémy s kompatibilitou „podmnožin“. K přijetí standardu není nutné ověření praktické realizovatelnosti. Naopak TCP/IP vzniká stylem „od jednoduchého ke složitějšímu“ nejprve se přijme jednodušší řešení, pak se ev. přidává. Existuje záruka kompatibility alespoň na úrovni „společného minima“, pro přijetí standardu je nutné ověření praktické realizovatelnosti dokonce i praktické provozní zkušenosti.
- Standardy ISO jsou prodávány a jsou opravdu hodně drahé. Uplatňuje se strategie: *„chci abys dodržoval moje standardy, a musíš mi nejprve hodně zaplatit, abych ti vůbec řekl v čem spočívají“ a* výsledek tomu odpovídá. Naopak standardy TCP/IP (i související dokumenty) jsou dostupné volně a zdarma. Uplatňuje se strategie: *„když chci něco prosadit, musím k tomu maximálně usnadnit přístup“ a* tato strategie funguje.
- Referenční model ISO/OSI je vhodný jako model pro studium sítí a to s vynecháním relační a prezentační vrstvy jsou protokoly „ušity na míru“ síťovému modelu. Pro praktické použití je jen velmi málo vhodný. Naopak TCP/IP je výhodný pro praktické použití, jako model pro studium sítí již není až tak výhodný, síťový model TCP/IP je „ušit na míru“ konkrétním protokolům.

Kontrolní otázka:

1. Jaké jsou základní koncepční odlišnosti modelu ISO/OSI a TCP/IP?

Úkoly k zamyšlení:

1. Proč se ve většině síťových modelů používá rozdělení do vrstev?
2. Čím je rozdělení do vrstev specifické a odlišné od jiných způsobů dekompozice systémů?
3. Co považujete za nejdůležitější faktor, který vedl k tomu, že se v praxi výrazně více rozšířil model TCP/IP, než RM ISO/OSI?

Shrnutí obsahu kapitoly

V této kapitole jste se seznámili s referenčním modelem počítačové sítě ISO/OSI a jeho základními vlastnostmi, s principem rozdělení funkcí obecné počítačové sítě do vrstev a s názvy a funkcemi jednotlivých vrstev referenčního modelu, od první vrstvy fyzické až po poslední, aplikační. Dále s nejdůležitějšími protokoly prakticky používanými v počítačových sítích. Jedná se o protokol NetBIOS a jeho rozšíření NetBEIU, dále protokolovou sadu SPX/IPX, a zejména rodinu protokolů TCP/IP, která se v posledních několika letech velmi rychle rozšiřuje a stává se prakticky standardní protokolovou sadou pro komunikaci ve všech typech sítí. Proto také byl protokolové sadě TCP/IP věnován v této kapitole největší prostor, seznámili jsme se tedy s historií TCP/IP, jejími standardy, vrstevnatou strukturou a aktuálními problémy.

10 Systém ochran

V této kapitole se dozvíte:

- Jakým způsobem chrání operační systém přístup k systémovým a uživatelským zdrojům, konkrétně operační paměť disky, zařízení, programy a data.

Po jejím prostudování byste měli být schopni:

- Charakterizovat metody ochrany jednotlivých objektů v operačním systému.
- Porozumět ochraně paměti.
- Porozumět principům návrhu bezpečných operačních systémů.

Klíčová slova této kapitoly:

Systém ochran operačního systému. Ochrana paměti. Zámky a klíče. Bezpečný operační systém.

Doba potřebná ke studiu: 2 hodiny

Průvodce studiem

Studium této kapitoly vyžaduje pochopení předchozích kapitol a popisným způsobem zde nastudujete základní principy systému ochran operačního systému a principy návrhu bezpečných operačních systémů.
Na studium této části si vyhraďte 2 hodiny. Po celkovém prostudování a vyřešení všech příkladů doporučujeme vypracovat korespondenční úkol.

Systém ochran operačního systému zabezpečuje mechanismy pro řízení přístupu k systémovým a uživatelským zdrojům. Zdroji a zároveň chráněnými objekty jsou zde:

- paměť,
- sdílená zařízení typu disky,
- seriově znovupoužitelná zařízení - tiskárny, pásky,
- spustitelné programy,
- sdílená data.

Metodami ochrany těchto objektů v operačních systémech jsou:

- fyzická separace - procesy pro vykonávání operací různého stupně utajení používají oddělená zařízení,
- časová separace - procesy různého stupně utajené jsou prováděny v různém čase,
- logická separace - operační systém zajišťuje oddělení jednotlivých procesů tak, že pro každý vytváří iluzi, že má celý počítač pro sebe,
- kryptografická separace - použitím kryptografických metod procesy provádějí ukrytí svých dat (např. sdílení komunikačních linek).

Samozřejmě je možná kombinace několika metod separace. Metody jsou seřazeny dle rostoucí (implementační) složitosti a zároveň dle klesající spolehlivosti.

Operační systém může poskytovat různé **úrovně ochrany objektů:**

- *žádná ochrana* - postačující pokud dochází k samovolné časové separaci,
- *isolace* - (semi)paralelně běžící procesy jsou zcela odděleny a vůbec o sobě vzájemně neví, systém zajišťuje úplné ukrytí objektů ostatních procesů,
- *sdílení všeho nebo ničeho* - vlastník objektu deklaruje, zda je objekt přístupný všem (public), nebo soukromý (private) a tedy viditelný jen pro něho,
- *sdílení s omezenými přístupy* – operační systém testuje oprávněnost každého pokusu o přístup k danému objektu, k danému objektu a subjektu existuje záznam zda subjekt má právo přistupovat k příslušnému subjektu,
- *sdílení podle způsobilosti* (share by capability) - nadstavba předchozího způsobu sdílení, rozsah oprávnění může dynamicky záviset na aktuálním kontextu,
- *limitované použití objektů* - nespecifikuje pouze zda subjekt smí přistupovat k danému objektu, ale i operace, které subjekt smí s objektem provádět.

Seznam je opět seřazen podle (implementační) složitosti, které tentokrát přímo úměrně odpovídá kvalita poskytované ochrany.

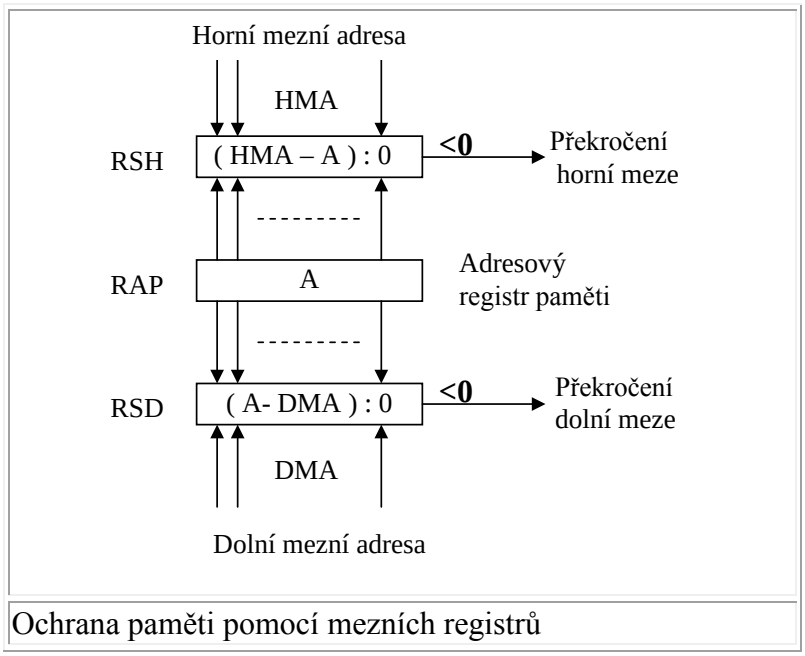
10.1 Ochrana paměti

Ochrana paměti je základním požadavkem pro zajištění bezpečnosti, má-li být spolehlivá, je nutná hardwarová podpora. HW podpora navíc poskytuje dostatečnou efektivitu ochrany. Způsoby ochrany paměti lze realizovat pomocí různých metod. Jedná se o:
- ohrady,
- relokace,
- mezní registry (Base/Bound),
- značkovaná (Tagged) architektura,
- ochrana segmentační metody,
- zámky a klíče.

Při ochraně paměti **ohradou (**fence) se stanoví hranice, operační paměť na jednu stranu od této hranice používá operační systém, na druhou stranu aplikační programy. Tato metoda je vhodná pro jednoduché jednouživatelské systémy, umožňuje však pouze chránit operační systém, nemůže být použita pro vzájemnou ochranu uživatelů většího systému. Implementace je velmi jednoduchá a to buď má systém pevně zabudovanou tuto hranici a nebo má tzv. *fence register*, jehož hodnotu porovnává s každou adresou, kterou aplikační program vygeneruje.

U **relokace** jsou programy vytvořeny tak, jako by v paměti ležely od adresy 0. V rámci procesu spouštění programu je ke všem odkazům v programu připočten relokační faktor a aplikace tak nemůže zasahovat do oblastí, v nichž leží operační systém. Metoda má však stejné nevýhody, jako předchozí způsob ochrany paměti.

U metody **mezních registrů (Base/Bound)** udávají dva mezní registry nejnižší a nejvyšší dostupnou adresu. Nastavuje je operační systém, když předává řízení procesu. Odkaz na paměť mimo rozsah způsobí vnitřní přerušení ("porušení ochrany paměti"). Nastavení mezních registrů musí být privilegovaná instrukce, jinak může program napsaný se špatným úmyslem číst nebo měnit paměťové oblasti jiných procesů.



Ochrana paměti pomocí mezních registrů

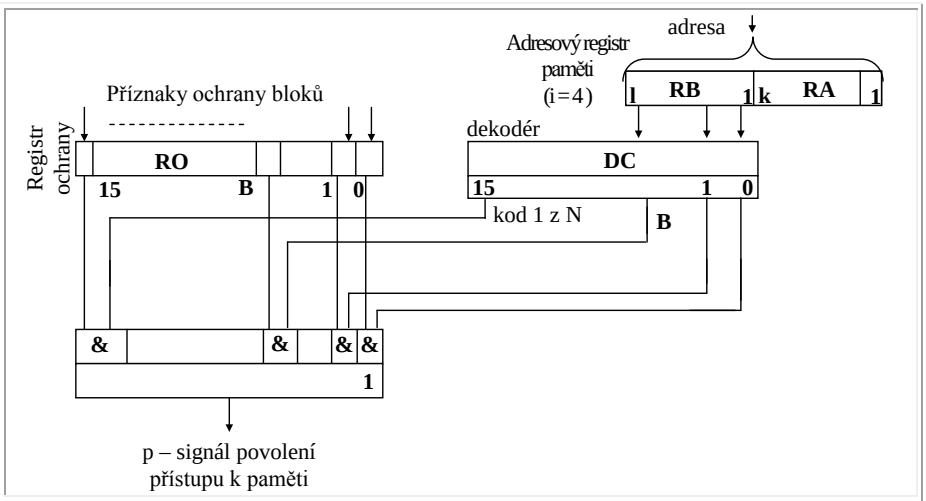
Tato metoda umožňuje vzájemné oddělení jednotlivých uživatelů, nechrání však kód aplikačního programu před chybou. Možným rozšířením je používat dva páry registrů, jeden pro vymezení oblasti pro kód procesu, druhý pro datovou zónu což však vytváří nemožnost selektivního sdílení pouze některých dat.

U **značkované (Tagged) architektury** je s každou adresou (slovem) v paměti stroje spojeno několik značkovacích (tag) bitů, jejichž obsah určuje typ zde uložených dat a povolené operace. Obsah značkovacích bitů je testován při každém přístupu k obsahu této adresy a mohou být měněny pouze privilegovanými instrukcemi. Určitou alternativou může být používání jednoho značkovacího bitů pro celý blok slov.

Při metodě přidělování paměti **segmentací** je celý program sestáven z několika bloků (segmentů) které mohou být nezávisle uloženy do paměti. Program potom generuje odkazy ve tvaru <*jméno_segmentu*><*ofset*>, kde *jméno_segmentu* je pomocí systémem udržovaného segmentového adresáře převedeno na adresu počátku segmentu, ke které je přičten *ofset* . Často je též ofset porovnán s velikostí segmentu, aby se zajistilo, že program nesahá “za segment”
Metoda již poskytuje dostatečné prostředky pro sdílení dat, navíc umožňuje ochranu kódu programu a případně i vybraných dat. Rovněž je schopna chránit uživatele navzájem.

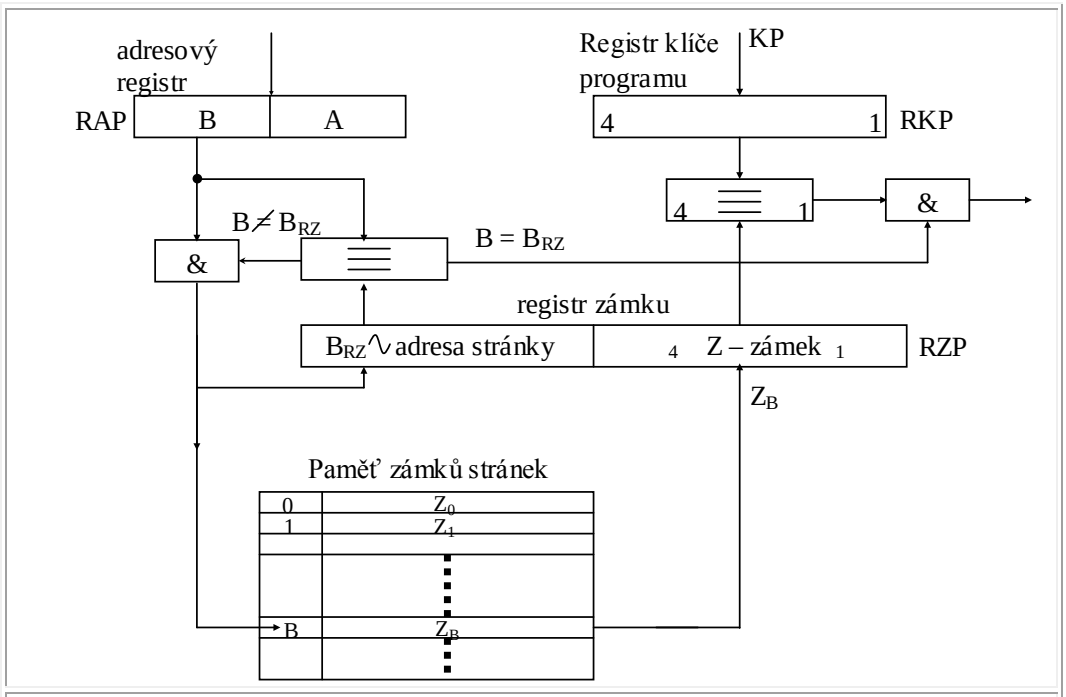
Ochrana paměti při segmentaci je pak realizována:

- mezním registrem na číslo segmentu,
- pro každý segment mezním registrem obsahujícím maximální povolenou adresu segmentu (limit velikosti segmentu); přičemž segmenty mohou mít různou délku,
- příznakovým registrem ochrany (následující obrázek).



Ochrana paměti – mezním registrem

Při metodě přidělování paměti pomocí **stránkování** mají jednotlivé segmenty (stránky) stejnou velikost. Adresování je dvousložkové a dáno <číslo_stránky><ofset>. Ochrana proti adresování za stránku je vyřešena samovolně tím, že nelze udělat větší ofset, než je velikost stránky. Možnost ochrany obsahu stránek je poněkud slabší než v případě segmentů, neboť není příliš jasná vzájemná souvislost obsahů stránek a rozdělení programu a dat do stránek. Proto je výhodnější použití metody ochrany paměti pomocí **zámků a klíčů**. Paměť je rozdělena na stránky pevné velikosti (např. 4 KB). Každé stránce paměti je přiřazen zámek (= celé číslo). Procesor má speciální registr, který slouží jako klíč. Proces může používat pouze ty stránky paměti, které mají zámek nastavený na stejnou hodnotu, jako je klíč. Operační systém může používat univerzální klíč číslo 0, který umožňuje přístup k libovolné stránce paměti.



Ochrana paměti pomocí zámků a klíčů

10.2 Ochrana obecných objektů

S rozvojem multiprogramování vzrůstá škála objektů, které je třeba chránit

- paměť,
- soubory a data na záznamových zařízeních,
- běžící programy,
- adresáře souborů,
- hardwarová zařízení,
- různé datové struktury (stack,…),
- interní tabulky operačního systému,
- různé instrukce.

Na rozdíl od problému ochrany paměti, zde nemusí existovat centrální arbitr, přes kterého jsou směrovány všechny přístupy, navíc typů přístupů může být celá řada

Cílem ochrany objektů je:

- *Kontrolovat každý přístup* - subjekt může pozbýt přístupová práva a tedy je nutno mu zabránit v dalším používání objektu.
- *Povolení co nejmenších práv* - subjekt by měl mít pouze nejmenší možná oprávnění nutná ke korektnímu plnění jeho úkolu a to i v případě, že případná další práva by pro něj byla bezcenná. Toto uspořádání snižuje možnost průniku v případě selhání části ochranného mechanismu.
- *Ověření přijatelného používání* - někdy je daleko podstatnější než přidělení či odepření přístupu moci kontrolovat, co subjekt s daným objektem provádí

10.2.1 Mechanismy ochrany obecných objektů

Zde se budeme zabývat běžnými mechanismy ochrany obecných objektů založených na:

- adresářích,
- seznamu oprávnění,
- přístupové matici,
- způsobilosti,
- procedurálně orientovaném přístupu.

Základní metodou ochrany souborů je vytváření chráněných **adresářů (directory).** Každý soubor má svého vlastníka, který k němu vlastní veškerá práva, včetně práva určovat rozsah oprávnění ostatních uživatelů k tomuto souboru. S každým uživatelem je spojena specielní struktura - *adresář* - obsahující odkazy na všechny soubory, k nimž má daný uživatel nějaké oprávnění, včetně popisu tohoto oprávnění. Nevýhodou může být velký rozsah adresářů a velmi obtížná správa a úpravy takto přidělovaných oprávnění. Rovněž udržení přehledu o tom, kdo k danému souboru má jaká práva může být problematické. Tato metoda lze snadno rozšířit na libovolné objekty a subjekty.

Opačným přístupek k problému ochrany je vytváření **seznamu oprávnění (Access Control List)**. Tentokrát je s každým *objektem* udržován seznam informací, které subjekty k němu mají jaká oprávnění. Metoda umožňuje snadno přidělovat implicitní práva subjektům, případně skupinám subjektů. Při vhodném označení subjektů a použití expanzních znaků může být tato metoda dostatečně pružná.

Př: Pepek_Group1_Troja
*_Group1_*

Seznamy zpravidla bývají udržovány setříděné tak, že záznamy s expanzními znaky jsou na konci a tak stačí hledat první shodu s identifikací subjektu a použít tímto záznamem specifikované oprávnění.

Další metodou je **přístupová matice (Access Control Matrix).** Řádky matice odpovídají jednotlivým subjektům, sloupce objektům. V políčku daném řádkem a sloupcem je záznam o úrovni oprávnění odpovídajícího subjektu k příslušnému objektu. Přístupová matice je zpravidla velmi velká záležitost (zhusta řídká).

Způsobilost budeme chápat jako nefalšovatelný token, jehož vlastnictví dává vlastníkovi specifická práva k danému objektu. Token lze chápat jako lístek do kina. Jednou z metod zajštění nefalšovatelnosti je, že tokeny se nepředávají přímo subjektům, ale jsou udržovány v chráněné oblasti paměti, přístupné pouze systému. Při přístupu k objektu tak systém zkontroluje existenci příslušného tokenu. Tento postup lze urychlit tím, že zvlášť udržujeme seznam **„způsobilostí“** právě běžícího procesu. Výhodou metody je, že dovoluje definovat nové dosud neznámé způsoby používání objektů a přidělovat odpovídající oprávnění. Nevýhodou je opět poněkud obtížná správa těchto tokenů, zejména odebrání „způsobilosti“ je netriviální operace.

Procedurálně orientovaný přístup namísto přidělování obecného přístupu k subjektu (čtení, zápis, …) může přidělovat právo používat některých funkcí z rozhraní, prostřednictvím kterého je objekt zpřístupňován. Metoda podporuje koncept skrývání a zapouzdřování informací. Nevýhodou je jistá ztráta efektivity a rychlosti přístupu.

10.2.2 Ochrana souborů

Každý multiuživatelský systém musí poskytovat mechanismus na ochranu souborů. Dřívější operační systémy vycházely z principu, že uživatelé jsou v zásadě důvěryhodní a protože nechtějí, aby jim někdo něco udělal s jejich soubory, nedělají to ostatním. Přístup ke specifickým souborům může administrátor vázat na zadání hesla.

Uživatelé jsou podle svého zaměření, pracovního zařazení apod. vhodně **rozděleny do skupin**. Pro účely ochrany souborů je svět rozdělen na vlastníka souboru, skupinu, do které vlastník patří a ostatní uživatele. Předpokládá se, že uživatelé v rámci skupiny potřebují sdílet data. Při vytvoření souboru vlastník specifikuje, jaká práva přiděluje sobě, uživatelům ve stejné skupině a ostatním uživatelům. Metoda je jednoduchá, snadno implementovatelná leč neposkytující dostatečně jemné rozlišení. Navíc je většinou nutné, aby každý uživatel byl právě v jedné skupině, jinak nastávají problémy s přidělováním práv skupinám.

Při vytvoření souboru vlastník specifikuje **hesla**, potřebná pro jisté módy přístupu k souboru, heslo zašle uživatelům, kteří mají mít přístup. Systém splní žádost o přístup k souboru pouze tomu, kdo se prokáže odpovídajícím heslem. Nevýhodou je, že v případě zapomenutí není možno zjistit, jak heslo vypadalo, v případě, že dojde k vyzrazení hesla je složité nastavit nové a stejně obtížné je odejmout právo přístupu.

Určitou modifikací přidělování práv jako v případě ochrany po skupinách je navíc umožněno stanovit, že (spustitelný) soubor smí být **prováděn s oprávněním vlastníka**. Prostřednictvím rutin běžících s oprávněním vlastníka lze řízeně přistupovat k souborům, ke kterým uživatel přímý přístup nemá. Problémem popsaných schemat je jistá těžkopádnost, uživatel nemůže selektivně přidělovat práva jistým uživatelům k jistým skupinám souborů. Kontrolní matice a podobné metody jsou zase příliš rozsáhlé a obtížně spravovatelné.

Další modifikací je vytvoření **seznamu oprávnění.** Ke každému souboru může uživatel vytvořit „seznam oprávnění“ udávající kdo má jaká práva. Každý uživatel je členem jedné skupiny, navíc administrátor může vytvořit skupinu typu „obecný identifikátor“, a tuto skupinu mohou uživatelé uvádět v „seznamech oprávnění“ a tyto mohou být též použity pro přidělování přístupu k ostatním systémovým zdrojům.

10.2.3 Autentizace subjektů

Shora popsané bezpečnostní mechanismy odvíjejí svoji činnost od informace, ***kdo*** žádá jaký přístup. Je tedy nutné mít mechanismus pro autentizaci subjektů. Jedním z nejzákladnějším, jednoduchým myšlenkově i implementačně je pomocí **hesla.**
Ověřuje se platnost páru

<identifikace:heslo>.

Mechanismus přijetí této dvojice by neměl poskytovat útočníkovi zbytečné informace o systému a je vhodné, aby vyhodnocoval až korektnost zadání celé dvojice. Někdy je proces autentizace záměrně pomalý - sekundy až desítky sekund - což znemožňuje hádání hesel. Tento mechanismus by neměl podávat informace o příčině chyby a neměl by umožňovat neomezené zkoušení. Přihlašování do systému lze navíc omezit na určitý čas či místo, dále omezením počtu současných přihlášení daného uživatele do systému.

Mechanismus musí mít možnost kontrolovat korektnost zadaného hesla, tedy je nutné aby udržoval informace o všech heslech. Ukládání hesel do textových souborů v podobě dvojice <identifikace:heslo> je velmi nevhodné. Hesla lze zjistit z odcizených záloh, z dumpů paměti při pádech systému nebo dojde-li chybou některé komponenty systému k vyzrazení souboru hesel.

Proto je vhodné soubory hesel zašifrovat. Pro šifrování je možné použít konvenčních šifer, nebo lépe kryptografických hašovacích funkcí. Soubory zašifrovaných hesel mohou být pak volně přístupné. V případě, že by u dvou uživatelů se stejným heslem vyšla stejná šifra, je vhodné před šifrováním k heslu přidat náhodný řetězec (salt), kdy salt je uchováván zároveň se zašifrovaným heslem a při verifikaci vždy přidán k zadanému heslu.

Pro lepší úschovu hesel má uživatel namísto konstantní fráze přiřazenu konstantní funkci (vhodný matematický výpočet, dešifrování soukromým klíčem, apod…). V procesu autentizace pak obdrží od systému náhodně zvolené vstupní parametry a odpoví výsledkem. Tato metoda je obzvláště vhodná pro vzájemnou autentizaci strojů.

10.3 Návrh bezpečných operačních systémů

Na kvalitě operačního systému závisí bezpečnost celého mechanismu ochrany dat. Operační systém kontroluje chování uživatelů a programů a v konečném důsledku zpřístupňuje utajované informace. Proces vývoje bezpečného operačního systému lze rozdělit do několika fází:

- *bezpečnostní modely* - vytvoří se formální modely prostředí a zkoumají se způsoby, jak v tomto prostředí zajistit bezpečnost,
- *návrh* - po zvolení vhodného modelu je hledán vhodný způsob implementace,
- *ověřování* - je třeba ověřit, že navržená implementace skutečně odpovídá teoretickému modelu,
- *implementace* - praktické a důkladné provedení shora uvedených teoretických úvah.

10.3.1 Bezpečnostní modely operačních systémů

První fází tvorby bezpečného operačního systému je volba vhodného bezpečnostního modelu. Základními požadavky bezpečnosti jsou: **utajení, integrita, dostupnost.**

Jednoúrovňové modely jsou vhodné pouze pro případy, kdy stačí jednoduché ano/ne rozhodování, zda danému subjektu poskytnout přístup k požadovanému objektu. K tomu se používají modely **monitoru**:

- subjekt při přístupu k objektu vyvolá tzv. **monitor** a předá mu žádost jakou akci s kterým objektem chce provést,
- monitor žádost vyhodnotí a na základě informací o přístupových právech vyhoví či nikoliv.

Výhodou tohoto modelu je jednoduchost a snadná implementovatelnost, nevýhodou je, že proces poskytující služby monitoru je volán při každém přístupu k libovolnému objektu, což systém velmi zatěžuje. Další nevýhodou je, že tento model je schopen kontrolovat pouze přímé přístupy k datům, ale není schopen zachytit přístup k proměnným uvnitř souboru.

Tuto nevýhodu lze odstranit následujícím modelem. Ve fázi vývoje bezpečnostního modelu je prováděno testování všech modulů, zda jejich výstupy závisí na interakcích se senzitivními daty a případně jakým způsobem. Z těchto dílčích výsledků je sestavován celkový graf závislostí. Veškeré požadavky na systém procházejí inteligentním filtrem, který zjišťuje, zda nedochází k nežádoucí kompromitaci informací.

Víceúrovňové modely vycházejí z několika stupňů senzitivity a "oprávněnosti" přístupu k objektům a subjektům. V jednoúrovňovém modelu jsme měli jednoduché vztahy objekt je/není senzitivní, subjekt má/nemá přístup k danému objektu.

Jeden ze základních víceúrovňových systémů vychází z principu, že každá informace je zařazena do některé z kategorií utajení (např. *unclassified, confidental, secret, top secret*), které jsou disjunktní. Tento model se někdy nazývá „**Military security model**". Silné uplatnění zde má *princip nejmenších privilegií* - každý subjekt má mít pouze taková oprávnění, aby mohl konat svoji práci. Všechny chráněné informace jsou rozděleny podle obsahu do *oblastí* (compartments), přičemž každá informace může být i v několika oblastech zároveň. Klasifikací informace potom rozumíme dvojici:

<*stupeň_utajení, oblasti*>.

Aby subjekt mohl používat požadovanou informaci, musí mít dostatečné oprávnění které má stejný tvar jako klasifikace tj. <*stupeň_utajení, oblasti*>, tedy daný subjekt smí používat informace až do *stupeň_utajení* v těchto *oblastech*. Platí tedy:

$$O \leq S \Leftrightarrow st\_utaj_O \leq st\_utaj_S \wedge oblast_O \subseteq oblast_S ,$$

kde relace $\leq$ odpovídá *oprávnění* subjektu *S* k danému objektu *O*.

Zobecněním tohoto modelu je tzv. **svazový model** (Lattice model) kde relace $\leq$ je částečným uspořádáním. Množina klasifikací všech informací v systému tvoří svaz, stejně tak množina oprávnění všech subjektů. V různých oblastech se používá různých svazů, např. v komerční oblasti jsou obvyklé stupně utajení *public, company confidental, high security*, rovněž rozdělení do oblastí se liší případ od případu. svazový model je často používaným modelem v mnoha prostředích

Další model se zabývá tokem informací uvnitř systému a nazývá se „**Bell-LaPadula model**". Model popisuje takové povolené přesuny informací, aby bylo zajištěno jejich utajení. Pro každý subjekt *S* resp. objekt *O* v systému je definována bezpečnostní třída *C*(*S*) resp. *C*(*O*). Subjekt *S* může číst objekt *O* právě když

$$C(O) \leq C(S).$$

Subjekt *S* mající právo čtení k objektu *O* může zapisovat do objektu *P* právě když

$$C(O) \leq C(P).$$

Tento model je použitelný v systémech, které paralelně zpracovávají informace různého stupně utajení.

Duálním modelem k Bell-LaPadula modelu který navíc řeší integritu dat je **Biba model**.
Nechť pro každý subjekt *S* resp. objekt *O* v systému je definována integritní bezpečnostní třída *I*(*S*) resp. *I*(*O*). Subjekt *S* může modifikovat objekt *O* právě když

$$I(O) \leq I(S).$$

Subjekt *S* mající právo čtení k objektu *O* může zapisovat do objektu *P* právě když

$$I(O) \geq I(P).$$

Biba model se zabývá zajištěním integrity a tedy i důvěryhodnosti dat. Bepečnostní třída entity v podstatě popisuje míru její důvěryhodnosti pro ostatní. Tento model však vůbec neřeší utajení dat.

Vzhledem k tomu, že nalezení kompromisu mezi zajištěním integrity a utajení je problematikou již dosti složitou, uvedeme si zde dva modely, které jsou více teoretické než prakticky realizovatelné.

Graham-Denning model pracuje s množinou subjektů *S*, množinou objektů *O*, množinou práv *R* a přístupovou maticí *A*. Každý objekt má přiřazen jeden subjekt nazývaný *vlastník*, každý subjekt má přiřazen jiný subjekt nazývaný *kontroler*. Model definuje následující práva:

- *vytvořit objekt* - povoluje subjektu vytvořit v systému nový objekt,
- *vytvořit subjekt, zrušit objekt, rušit subjekt* - obdobně jako předchozí,
- *číst přístupová práva* - povoluje subjektu zjistit aktuální přístupová práva jistého subjektu k určitému subjektu,
- *přidělit přístupová práva* - dovoluje vlastníku objektu přidělit jistá práva k objektu určitému subjektu,

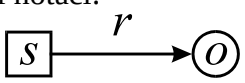
- *zrušit přístupová práva* - dovoluje vlastníku objektu resp. kontroleru subjektu odebrat danému subjektu jistá práva k objektu resp. Subjektu,
- *předat přístupová práva* - dovoluje subjektu předat některé ze svých práv jinému subjektu (každé oprávnění může být předatelné či nikoliv, obdrží-li subjekt předatelné právo, může jej dále předat jako předatelné či nepředatelné).

Následující tabulka uvádí podmínky nutné pro vykonání operací s přístupovými právy.

vytvořit objekt *o*	-
vytvořit subjekt *s*	-
zrušit objekt *o*	vlastník je v *A*[*x*,*o*]
zrušit subjekt *s*	vlastník je v *A*[*x*,*s*]
číst přístupová práva *s* k *o*	kontroler je v *A*[*x*,*s*], nebo vlastník v *A*[*x*,*o*]
zrušit přístupové právo *r* subjektu *s* k *o*	kontroler je v *A*[*x*,*s*], nebo vlastník v *A*[*x*,*o*]
přidělit *s* právo *r* k objektu *o*	vlastník je v *A*[*x*,*o*]
předat přístupové právo *r* nebo *r** k objektu *o* subjektu *s*	*r** je v *A*[*x*,*o*]

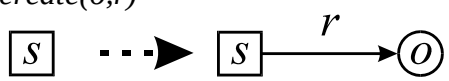
*r** označuje předatelné právo

Druhý teoretický model je **Take-Grant systém,** který pracuje s čtyřmi základními primitivami: *create, revoke, take, grant.* Předpokládáme, že systému obsahuje množinu subjektů *S*, množinu objektů *O*, objekty dělíme na aktivní (zároveň i subjekty) a pasivní (nejsou subjekty) a množinu práv *R*. Pro popis operací použijeme následující notaci:

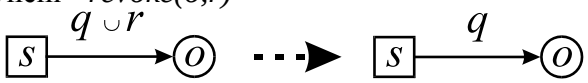


Subjekt *s* má k objektu *o* oprávnění *r*. Pak:

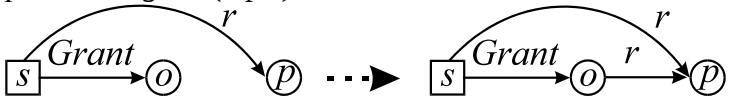
- vytvoření objektu - *create(o,r)*



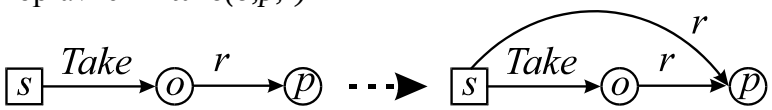
- odebrání oprávnění - *revoke(o,r)*



- předání oprávnění - *grant(o,p,r)*



- převzetí oprávnění - *take(o,p,r)*



Výhodou popsaného systému je, že umožňuje v subpolynomiálním čase řešit dotazy na dostupnost jistého objektu pro daný subjekt.

10.3.2 Návrh bezpečného operačního systému

Implementace bezpečnostních mechanismů je v přímém rozporu s efektivitou celého systému. Operační systém vykonává několik úzce souvisejících činností souvisejících s bezpečnostní:

- autentizace uživatelů,
- ochrana paměti - mezi uživateli i v rámci jednoho uživatelského prostoru,
- řízení přístupu k souborům a I/O zařízením - ochrana před neautorizovaným přístupem,
- alokace a řízení přístupu k obecným objektům - zajištění bezproblémového současného přístupu více uživatelů k stejnému objektu,
- zabezpečení sdílení - zejména zajištění integrity a konzistentnosti,
- zajištění spravedlivého přístupu - o harwarové prostředky se opírající mechanismus zajišťující, že všichni uživatelé dostávají přidělen procesor a ostatní systémové zdroje,
- meziprocesová komunikace a synchronizace - systém poskytuje mechanismus pro bezpečnou komunikaci mezi procesy.

Bezpečnost musí být brána v potaz ve všech aspektech návrhu systému a musí být zapracována již v prvotním návrhu. Je velmi obtížné ji "přidat" do hotového návrhu. Je vhodné mít na paměti následující principy:

- Nejmenší práva - každý subjekt by měl mít pouze nezbytná práva.
- Ekonomický návrh - bezpečnosní systém má být malý a jednoduchý, pak je testovatelný a věrohodný.
- Otevřený návrh - bezpečnostní mechanismus by měl být veřejně známý (a oponovaný) a měl by záviset na bezpečnosti co nejméně objektů.
- Úplné zprostředkování - veškeré přístupy k objektům zprostředkovává a testuje operační systém.
- Povolování operace - co není výslovně povoleno, je zakázáno.
- Rozdělené oprávnění - přístup k objektům by měl záviset na více podmínkách (např. správná autentizace a vlastnictví klíče).
- Nejmenší sdílené prostředky - sdílené moduly jsou potenciálním kanálem pro únik informací, mělo by jich být co nejméně.
- Snadná použitelnost - mechanismus není obcházen.

V této souvislosti zde uveďme některé principy, použité u různých architektur operačních systémů.

Ve virtuální architektuře operačního systému se provádí logické oddělení uživatelů, které poskytuje dojem fyzické separace. Pomocí mechanismu stránkování jsou cela odděleny adresní prostory jednotlivých uživatelů, každý uživatel vidí pouze svůj prostor, do každého z uživatelských prostorů je mapována oblast paměti obsahující vlastní systém, čímž vzniká dojem, že uživatel má celý systém sám pro sebe. Některé virtuální architektury poskytují nejen virtuální paměť, ale provádí virtualizaci celého počítače - I/O zařízení, file-systému a dalších zdrojů. Simulovaný stroj pak může mít naprosto odlišné vlastnosti od vlastností skutečného počítače. Poskytovaná ochrana je tedy daleko silnější. Řada virtuálních architektur je navrhována tak, aby na jednom fyzickém počítači bylo možno provozovat zároveň několik různých operačních systémů. To představuje další vrstvu ochrany, neboť pokud se uživateli podaří

proniknout ochrannými mechanismy operačního systému, který používá, získá přístup pouze k této jediné doméně, neboť virtuální stroj mu zabrání v přístupu k celému počítači.

U monolitické architektury provádí jádro operačního systému nejzákladnější funkce - standardně synchronizace, meziprocesová komunikace, zasílání zpráv a obsluha přerušení. Zabezpečení jádra je implementováno uvnitř jádra, což má několik důvodů:

- oddělení od zbytku systému zjednodušuje ochranu mechanismu,
- všechny bezpečnostní funkce jsou shromážděny v jednom kusu kódu, tedy implementace bepečnosti je kompaktní,
- jádro nebývá velké, tedy implementace je snadno ověřitelná,
- je snazší provádět testování a změny bepečnostního mechanismu,
- přes jádro procházejí veškeré žádosti o přístup ke všem objektům (volání odpovídajících modulů), tedy je možno zachytit každý přístup.

Jádro hlídá zejména:

- aktivaci procesů - zajišťování context switchingu, realokací paměti, access kontrol listů,
- střídání domén - procesy často provádějí volání procesů běžících v jiné bezpečnostní doméně, za účelem získání senzitivních informací,
- ochrana paměti - je nutné hlídat všechny odkazy na paměť, aby nedocházelo k narušení bezpečnostních domén,
- I/O operace.

Ve vrstvené architektuře operačního lze jednotlivé vrstvy chápat jako soustředné kruhy, čím blíže je vrstva středu, tím je důvěryhodější a bezpečnější. Ne všechny bepečnostní funkce (např. autentizace uživatele) jsou implementovány uvnitř bezpečnostního jádra. Bezpečnostní jádro spolupracuje s okolními spolehlivými vrstvami, které by měli být formálně ověřeny a přinejmenším dobře otestovány. Každá vrstva používá služby nižších vrstev a sama vyšším vrstvám poskytuje služby jisté úrovně bezpečnosti, stejná funkce může být implementována v několika vrstvách zároveň.

Jednotlivé kruhy jsou číslovány od 0 (jádro), čím důvěryhodnější proces, tím nižší číslo kruhu, do kterého patří. Kruhy jsou soustředné a překrývající se - proces patří do kruhu *k* a všech dalších, ve středu je hardware počítače. Každá procedura, nebo oblast obsahující data se nazývá *segment. O*chrana segmentu založena na trojici $\langle b_1, b_2, b_3 \rangle$, $b_1 \leq b_2 < b_3$, nazývané závora kruhu (ring bracket), (b_1, b_2) nazýváme přístupová závora (access bracket), (b_2, b_3) potom závora volání (gate extension, call bracket). Nechť programová rutina patří do kruhu *k,* pokud $k = b_1$, může pracovat přímo s daty tohoto segmentu, pokud $b_1 < k \leq b_2$, může pracovat přímo s kopií dat a pokud $b_2 < k \leq b_3$, může k datům přistupovat pouze prostřednictvím definovaného rozhraní (gate).

Tento základní mechanismus, nazývaný též nondiscretionary nebo mandatory control může být dále doplněn o další doplňkové (discretionary) mechanismy - např. k daným datům smějí přistupovat pouze jmenovité procesy, procesy

patřící do okruhu přístupové závory mohou volně číst, ale zapisovat pouze za specifických podmínek apod.

10.4 Průniky operačním systémem

Místem největšího počtu průniků je mechanismus zpracování I/O operací a to z těchto důvodů:

- mnohá I/O zařízení jsou do značné míry inteligentní a nezávislá na zbytku systému, provádějí optimalizaci své činnosti, jejich řadiče často spravují více takovýchto zařízení,
- kód I/O operací je často velmi rozsáhlý, je těžké jej řádně testovat, někdy je dokonce nutné používat kód dodaný výrobcem zařízení,
- v zájmu rychlosti a efektivity I/O operace občas obcházejí bezpečnostní mechanismy operačního systému, jako stránkování, segmentaci apod.,
- velká část I/O operací je znakově orientovaná, v zájmu efektivity se často příslušné kontroly neprovádějí s každým přijatým znakem, ale pouze při startu operace.

Druhým problémem je hledání kompromisu mezi důkladnou izolací uživatelů a nutností umožnit sdílení dat. Tento kompromis bývá obtížně formalizovatelný, nejasnosti návrhu pak mohou být příčinou "děr" v implementaci.

Ne vždy je možné provádět kontroly oprávněnosti s každou operací, často je kontrola prováděna pouze jednou během provádění celého bloku akcí, pokud se v této době uživateli podaří změnit parametry, může dojít k průniku.

Další skulinu v bezpečnosti může způsobit snaha o obecnost možného nasazení systému - aby bylo možno systém používat pro nejrůznější úkoly, ponechají návrháři často mechanismus, pomocí kterého si uživatel může systém přizpůsobit. Tento mechanismus ovšem může být zneužit.

Kontrolní otázky:

1. Vysvětlete, co se musí v operačním systému chránit a proč?
2. Popište systém ochran paměti.
3. Popište proces vývoje bezpečného operačního systému.

Úkoly k zamyšlení:

1. Zamyslete se nad bezpečnostním modelem operačního systému ve Vašem počítači a zvažte, na co by jste si měli dávat větší pozor, aby nebyl systém ohrožen.

Korespondenční úkol:

1. Navrhněte kritéria posuzování bezpečnosti operačního systému.

Shrnutí obsahu kapitoly

V této kapitole jste se seznámili s obecnými principy ochrany přístupu operačního systému k systémovým a uživatelským zdrojům,.

11 Uživatelské rozhraní

V této kapitole se dozvíte:

- Jaká uživatelská rozhraní se používají v operačních systémech.
- Proč je grafický systém velmi důležitou částí moderního operačního systému?
- Jaké funkce má grafické rozhraní operačního systému.

Po jejím prostudování byste měli být schopni:

- Rozlišit rozdíly mezi řádkovým a grafickým interpretem příkazů
- Charakterizovat vrstvy grafického systému.
- Porozumět principům vytváření oken.
- Popsat funkce dialogového okna.

Klíčová slova této kapitoly:

Interpret příkazů. Vrstvený grafický systém. Okno.

Doba potřebná ke studiu: 2 hodiny

Průvodce studiem

Studium této kapitoly je jednoduché a popisným způsobem zde nastudujete základní úlohy a funkce grafického systému a systému oken.
Na studium této části si vyhraďte 2 hodiny. Po celkovém prostudování a vyřešení všech příkladů doporučujeme vypracovat korespondenční úkol.

Z uživatelského pohledu se při troše zjednodušení dá říci, že operační systém se skládá ze dvou základních prvků: z toho, co nám ukazuje a z toho, co nám nabízí. V této části se zaměříme na služby, které operační systém nabízí uživateli.

11.1 Interpret příkazů

Součástí každého operačního systému je tzv**. interpret příkazů.** Jedná se o speciální program, jehož úkolem je přímo komunikovat s uživatelem, přebírat od něj příkazy a s využitím služeb operačního systému je plnit. Interpret příkazů bývá obvykle zcela standardní aplikací. Někdy má mírně výsadní postavení např. tím, že často není možné interpret příkazů ukončit.

Existují v zásadě dva základní typy příkazových interpretů:

- znakový (řádkový),
- grafický.

Historicky daleko starší **řádkový interpret** je dědictvím po terminálech sálových počítačů a pracuje velmi jednoduchým způsobem. Uživatel zapíše řádek, má přitom k dispozici základní editační příkazy. Teprve po odeslání řádku stisknutím klávesy „Enter“ je obsah řádku interpretován jako příkaz. Typickým příkladem operačního systému, vybaveného řádkovým interpretem příkazů, je UNIX nebo MS DOS. Takový typ komunikace maximálně

vyhovuje velmi zkušenému uživateli nebo programátorovi, jemuž obvykle umožňuje vytvářet příkazové dávky (tj. skupiny příkazů, které budou zpracovány automaticky v zadaném pořadí) a dává mu daleko rychlejší přístup k většině služeb. Začátečník je naproti tomu s řádkovým intepretem příkazů ztracen, protože na rozdíl od zkušeného uživatele nezná desítky nejrůznějších příkazů a jejich parametrů z hlavy a musí neustále listovat v manuálech.

Začátečníkům a méně zkušeným uživatelům proto dnes vychází naprostá většina operačních systémů vstříc druhou alternativou příkazového interpretu, kterou je **grafické uživatelské rozhraní**. V něm se řada příkazů volí prostřednictvím 'odpovídajících' akcí vyvolaných pomocí myši - soubor se např. smaže odtažením jeho ikony nad ikonu koše na odpadky, zkopíruje na jiný disk přemístěním jeho ikony nad ikonu požadovaného disku a podobně. Ostatní akce, které by se tímto způsobem vyjadřovaly obtížně, jsou k dispozici ve formě nabídek (menu), kde se uživatel může jednotlivými příkazy doslova přebírat. Parametry příkazů se pak zadávají pomocí jakýchsi formulářů, které uživatel vyplní; těmto formulářům říkáme dialogy nebo dialogová okna. Starší grafické příkazové interprety spoléhaly na pouhou přehlednost a intuitivní ovládání; modernější systémy se snaží vzdorovat i těm nejméně nápaditým uživatelům navíc i mohutnými systémy nápověd.

Znakově orientované rozhraní umožňuje vytvářet příkazové dávky. Grafická uživatelská rozhraní podobnou možností obvykle nedisponují a proto jsou vybaveny speciálními programy, které umožňují vytváření záznamů akcí uživatele.

Pro toho, kdo si dokáže zapamatovat jména a parametry několika desítek příkazů, jsou příkazy prostřednictvím řádkového interpretu přístupné často rychleji a pohodlněji než prostřednictvím grafického rozhraní. Tvůrci grafických uživatelských rozhraní jsou si tohoto handicapu vědomi a snaží se jej dohnat pomocí tzv. klávesových zkratek (hot keys, keyboard shortcuts). Princip klávesových zkratek je velmi jednoduchý: většinu často používaných příkazů můžeme vyvolat pouhým stisknutím vhodné kombinace kláves, aniž bychom museli hledat příkaz v nabídkách.

Klávesové zkratky jsou samozřejmě výborné (grafické uživatelské rozhraní bez nich hraničí s nepoužitelností), ale mají také své nevýhody. Snad hlavní z nich spočívá v tom, že se přece jen hůře pamatují než anglická slova, která často tvoří příkazy řádkových interpretů.

Řádkový interpret umožňuje velmi jednoduché vkládání příkazů, které grafické rozhraní není prakticky schopné nabídnout.

Prostřednictvím řádkového interpretu můžeme kdykoli spustit kterýkoli program, který je na počítači vůbec k dispozici. Grafické uživatelské rozhraní nám naproti tomu umožní spustit pouze program, jehož ikonu momentálně vidíme na obrazovce.

Rozumná grafická uživatelská rozhraní si pomáhají řadou technik, z nichž asi nejrozšířenější je spojování programů a jejich datových souborů. Operační

systém dokáže zjistit, který program má zpracovávat určený datový soubor; uživateli pak stačí tento soubor otevřít a operační systém vyhledá a spustí potřebný program sám. Přesto však je spouštění programů - zvláště těch méně často používaných - obecně pohodlnější v řádkovém interpretu.

Většina moderních operačních systémů proto oba přístupy kombinuje tak, že uživatel má k dispozici grafické uživatelské rozhraní; jakmile však potřebuje provádět některé akce, které se v grafickém rozhraní realizují obtížně, může vyvolat jediným příkazem řádkový interpret.

11.2 Příkazy

Řádkové interprety často rozlišovaly tzv. vnitřní a vnější příkazy. Vnitřní příkazy byly přitom skutečně interpretovány, zatímco vnější příkazy byly naprosto běžnými programy. Jejich volání však mělo stejnou syntaxi se zápisem příkazů vnitřních, takže uživatel je za běžných okolností vůbec nemusel rozlišovat.

V operačním systému MS DOS např. existují příkazy COPY a XCOPY. Příkaz XCOPY je daleko 'chytřejší', jejich základní chování je však naprosto stejné: napíšeme-li příkaz

copy a:data.txt b:

nebo příkaz

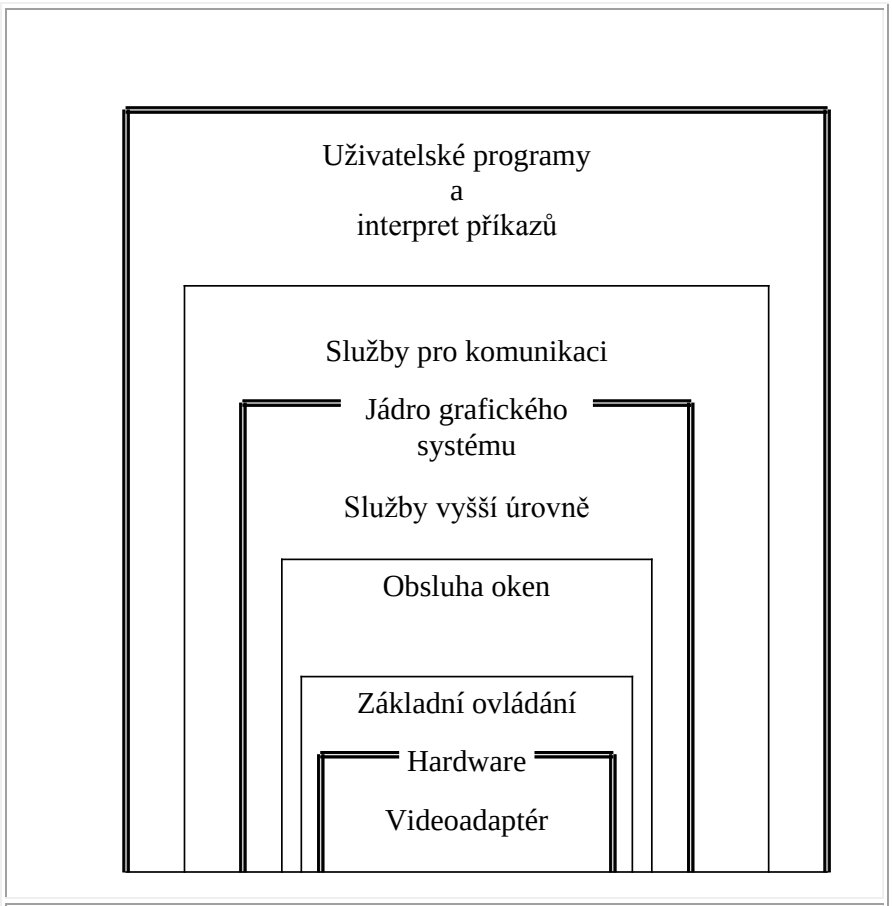
xcopy a:data.txt b:

zkopíruje se v obou případech soubor „data.txt“ z disku „a“ na disk „b“.

Vnější příkazy v grafických rozhraních dost dobře nelze použít; většina služeb, které tyto příkazy původně zajišťovaly, proto musí být integrována v grafickém příkazovém interpretu. Zbývající vnější příkazy - zpravidla to bývají administrativní prostředky pro správce systému - jsou buď ponechány jako přístupné pouze prostřednictvím řádkového interpretu, nebo se postupně mění v plnohodnotné aplikační programy, které se již netváří jako součást příkazového interpretu.

11.3 Grafické uživatelské rozhraní

Pro grafické uživatelské rozhraní je důležitý grafický systém počítače. Jen u skutečně nejjednodušších systémů může být celý grafický subsystém jediným nedílným celkem. Jinak je zapotřebí jej rozdělit do několika vrstev, z nichž každá zajišťuje vlastní skupinu úkolů a slouží vrstvě vyšší (vzpomeňme si na obecnou vrstvenou strukturu operačního systému - na obrázku vidíme podobný pohled, avšak jednotlivé vrstvy tentokrát odpovídají samostatným částem grafického subsystému).



Vrstvený grafický systém

Na nejnižší úrovni musí být jednoduchý systém základních grafických služeb umožňujících vlastní zápis základních grafických objektů na obrazovku. Je-li počítač osazen kvalitním grafickým procesorem, nemusí být tento systém vůbec zapotřebí. V další vrstvě musí stát systém zajišťující práci s obrazovkovými okny a/nebo s virtuálními obrazovkami.

Na vrstvě obsluhující okna každopádně leží zodpovědnost za korektní spolupráci s interaktivními vstupními zařízeními (jako je klávesnice nebo myš) i se samotnými procesy - ty totiž někdy potřebují vědět, v jakém stavu jsou právě jejich okna.

Na další úrovni je velmi vhodné implementovat vrstvu vyšších služeb grafického systému. Tato vrstva umožní programátorům aplikací pracovat skutečně s grafickými objekty (jako je čára, čtverec, kruh, plocha nebo třeba koule) a ne s nějakými obrazovými body, jejichž počet i barva závisí na grafickém adaptéru i na jeho momentálním režimu práce.

V jistém smyslu nejvyšší úrovní je vrstva služeb uživatelského grafického rozhraní. Jedná se o prostředky, které programům usnadní komunikaci s uživatelem prostřednictvím nabídek (menu), dialogových oken a řady dalších, dnes již do značné míry standardizovaných, prvků.

Tato vrstva je velmi důležitá nejen pro usnadnění práce aplikačním programátorům, ale především proto, aby bylo ovládání všech aplikací podobné a konzistentní. Přímým důsledkem neexistence této vrstvy například v MS DOSu je, že každý program ovládá grafický výstup jinak. Uživatel, který musí střídavě pracovat s různými grafickými systémy a systémy ovládání má mnohdy zhoršenou orientaci.

Ačkoli jsme minulou vrstvu nazvali nejvyšší, zmíníme se ještě o jedné. Je jí grafický interpret příkazů uživatele, který v grafických systémech stojí na místě „Stellu“ systémů orientovaných textově.

Na kvalitě a ergonomii interpretu příkazů totiž do značné míry záleží, bude-li se uživatelům se systémem pracovat pohodlně a dosáhnou-li snadno vysoké efektivity práce.

Grafické uživatelské rozhraní využívá grafických služeb nižších úrovní a nabízí základní operace pro komunikaci s uživatelem. Patří sem tedy skupina grafických prvků, jejichž prostřednictvím operační systém uživateli ukazuje momentální stav jednotlivých subsystémů a nabízí přípustné povely, i mechanismus spolupráce se vstupními zařízeními - jako je myš, klávesnice a v některých případech i další zařízení.

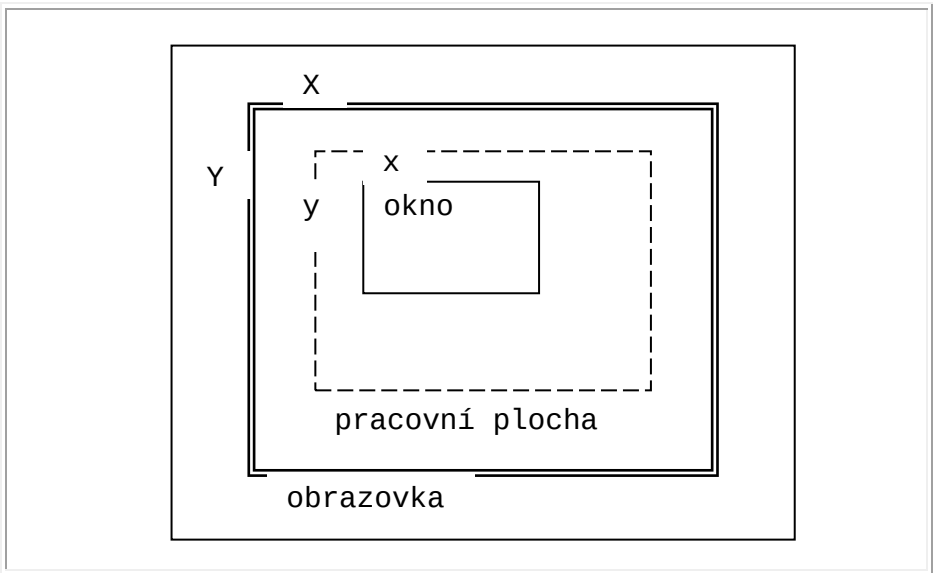
Ačkoli je variabilita mezi různými systémy grafického uživatelského rozhraní poměrně vysoká, využívají všechny několika shodných základních prvků. např. okno, nabídka (menu) nebo ikona. Praxe již dostatečně prokázala užitečnost těchto základních stavebních kamenů. Jednotlivé systémy grafického uživatelského rozhraní se navzájem odlišují.

Vzhled základních prvků jednoho systému se obvykle více či méně liší od vzhledu základních prvků systémů ostatních. Funkce základních prvků grafického rozhraní je určena daleko pevněji než jejich vzhled - je např. zvykem, že ikona reprezentuje nějaký objekt. Skutečně se jen velmi zřídkakdy setkáme s případem, že by ikona byla použita v jiném kontextu. Často se odlišuje i způsob vzájemného propojení a spolupráce základních prvků.

Nyní postupně probereme základní prvky grafických uživatelských rozhraní. U každého z nich popíšeme, čím je typický a se kterými jeho vlastnostmi se setkáme prakticky v každém prostředí. Pozorný čtenář si zajisté přiřadí tomuto obecnému popisu konkrétní příklad rozhraní se kterým pracuje.

11.4 Okna

Okna jsou dnes nejrozšířenějším a nejúspěšnějším mechanismem pro virtualizaci obrazovky. Okno je vlastně samostatné grafické výstupní zařízení. Z hlediska programu je okno obvykle pravoúhelníkem určitých rozměrů (s některými dalšími atributy), do kterého lze zapisovat libovolné grafické informace. Uživatel pak víceméně nezávisle na programu, který s oknem pracuje - určí postavení okna na obrazovce, jeho rozměry a viditelnost. Operační systém se sám postará o správné zobrazení obsahu okna (určeného programem) na obrazovce.



Okno na obrazovce

Nejčastěji program pracuje s plochou okna, která je vidět na obrazovce (nebo která by byla vidět, kdyby okno nebylo částečně zakryto jinými objekty). Má-li tedy okno sloužit jako „průhled“ na nějakou větší pracovní plochu - jak tomu také v naprosté většině případů bývá - musí se o to postarat programátor sám. V rámci programu je známa souřadnice okna uvnitř této pracovní plochy; na jejím základě program sám pozná, co (tedy kterou část plochy) má do okna kreslit.

Tuto situaci nám ozřejmí několik obrázků. Základem je předchozí obrázek, který ukazuje celou situaci. Dvojitý rámeček odpovídá celé obrazovce, plnou čarou je znázorněno okno a přerušovaná čára ukazuje pracovní plochu, jejíž část vidíme „skrz“ okno (není samozřejmě nikde psáno, že tato pracovní plocha musí být menší než obrazovka, jako tomu je na našem obrázku - pracovní plocha „za oknem“ může být samozřejmě zcela libovolně velká). Vzájemné postavení obrazovky a pracovní plochy je přitom určeno dvojicí souřadnic - první z nich, označená v obrázku jako [X,Y], určuje polohu okna na obrazovce. Druhá souřadnice, označená jako [x,y], naproti tomu udává relativní polohu okna vůči jeho pracovní ploše.

Jestliže nyní použijeme některý z prostředků pro změnu polohy okna na obrazovce, změníme samozřejmě pouze souřadnice [X,Y], zatímco souřadnice [x,y] zůstanou beze změny. Naopak můžeme použít jiných prostředků pro změnu průhledu oknem do jeho pracovní plochy; pak se ovšem změní jen souřadnice [x,y].

Popisovali jsme samozřejmě zcela obecný případ - existuje řada situací, kdy pracovní plocha okna přesně odpovídá oknu samotnému a na žádné straně jej nepřesahuje; souřadnice [x,y] pak jsou ovšem neměnné a mají hodnotu [0,0].

Na konec tohoto odstavce poznamenejme, že nejmodernější systémy umožňují programátorovi pracovat přímo s pracovní plochou okna; zajištění přemísťování okna po této ploše a zobrazení správného průhledu je již pouze věcí operačního systému a programátor se o ně nemusí starat.

Dosud jsme tiše předpokládali, že okna jsou součástí obrazovky. To však nemusí nutně být pravda. Okno může být přemístěno (změnou souřadnic [X,Y]) tak, že jeho část nebo dokonce celé bude mimo obrazovku. Program si může vyžádat okno, které je principiálně mimo obrazovku - to se může hodit např. v případě, kdy programátor nechce, aby na obrazovce bylo vidět postupné vykreslování výstupu. Nejprve všechny potřebné akce provede nad neviditelným oknem mimo obrazovku a jeho obsah pak přenese jedinou rychlou akcí do některého z „normálních“ zobrazovaných oken.

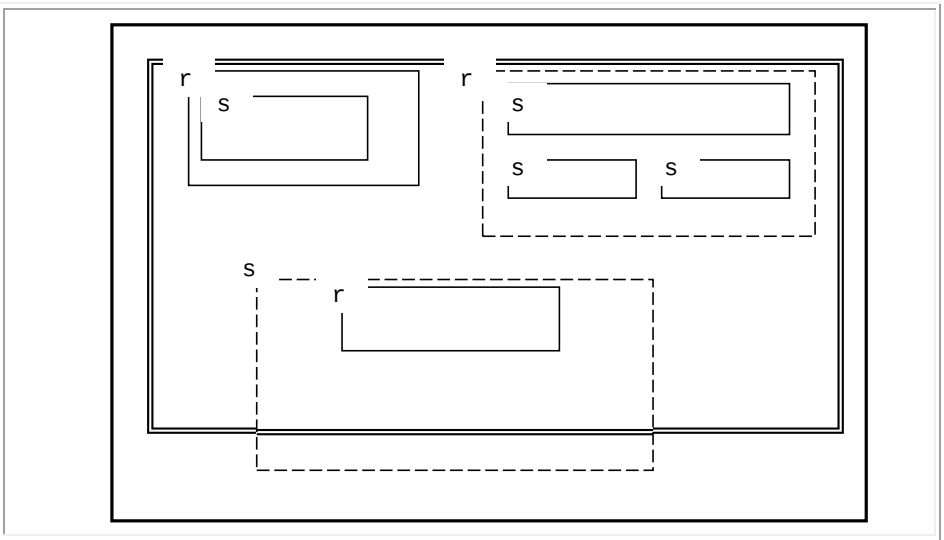
Moderní grafické systémy rozšiřují možnosti programátora zavedením tzv. **hierarchie oken.** To znamená, že kromě „horizontálních“ vztahů mezi jednotlivými okny, určených jejich vzájemnou polohou na obrazovce, existují i „vertikální“ vztahy, kdy jedno okno může být součástí jiného. To umožňuje poměrně pohodlným způsobem implementovat např. automatické přemísťování okna vůči jeho pracovní ploše.

Okna jsou uložena ve stromové hierarchické struktuře, ve které má každé okno svého předchůdce (tzv. rodiče) a libovolný počet svých následníků (tzv. synů). Operační systém pak automaticky zajišťuje následující mechanismy. Souřadnice „syna“ na obrazovce jsou relativní vůči souřadnicím jeho „rodiče“. Díky tomu je velmi snadné vytvářet skupiny oken, které lze pohodlně přemísťovat jako jediný celek - stačí, jsou-li všechna tato okna „syny“ jediného „rodiče“ a přemísťujeme-li právě tohoto „rodiče“. Relativní souřadnice zajistí, že všichni „synové“ se přemísťují zároveň s ním.

Má-li jak „rodičovské“, tak „synovské“ okno nějaký vlastní obsah, překrývá obsah „syna“ na obrazovce obsah „rodiče“.

Obsah „synovského“ okna není nikdy zobrazován mimo souřadnice „rodiče“. Právě tato služba totiž umožňuje pohodlně zajistit automatické zobrazování „průhledu“ do pracovaní plochy. „Rodič“ je vlastní okno, zatímco „syn“ reprezentuje právě pracovní plochu. Na obrazovce samozřejmě vidíme jen tu část pracovní plochy, která je uvnitř souřadnic „rodiče“. Pouhou změnou souřadnic „syna“ (které nejsou ničím jiným než souřadnicemi [-x,-y]) zajistíme změnu průhledu.

Na následujícím obrázku si můžeme prohlédnout nejčastější případy využití hierarchie oken. „Rodičovské“ okno je v levém horním rohu vždy označeno písmenem **r**, „synovské“ písmenem **s**; okno znázorněné plnou čarou je přímo viditelné na obrazovce, okno znázorněné přerušovanou z různých důvodů viditelné není.



Hierarchie oken

První příklad v levém horním rohu obrazovky je nepochybně nejčastější – „rodičovské“ okno obsahuje rámeček, název a další pomocné prvky, zatímco v „synovském“ okně je uložen vlastní obsah okna. Z hlediska uživatele počítače se obě okna jeví jako okno jediné, obsahující řadu ovládacích prvků rámujících vlastní obsah: každé okno je orámováno, má záhlaví s názvem okna a v pravé části okna vidíme i pruh posuvníku.

Druhý příklad v pravém horním rohu obrazovky ukazuje „neviditelné“ rodičovské okno (neviditelné proto, že mu programátor prostě nepřiřadil žádný vlastní obsah), jehož jediným účelem je spojit několik „synovských“ oken do jediného celku, který se bude snadno pohybovat po obrazovce. Jednotlivá okna se mohou samozřejmě překrývat - nejedná se jen o překrytí „rodičovského“ okna „synovským“ oknem, ale také o vzájemné překrývání oken na stejné úrovni hierarchie. V souvislosti s tímto překrýváním musí operační systém zajistit řadu věcí.

Aktivujeme-li některý proces, bylo by jistě vhodné, aby operační systém všechna jeho okna přemístil do popředí před momentálně méně zajímavá okna ostatních procesů. Operační systém proto musí jako součást každého okna udržovat také informaci o tom, kterému procesu okno patří.

Je vhodné si uvědomit, že se nejedná o jediný důvod pro udržování informací o vlastníkovi. Je např. běžné, že klepneme-li myší do okna některého z neaktivních procesů, operační systém proces ihned aktivuje. Často také operační systém umožňuje skrýt všechna okna zvoleného procesu nebo všechna okna všech neaktivních procesů.

Jednotlivé procesy samozřejmě nevědí nic o tom, je-li některé z jejich oken momentálně překryto oknem jiného procesu (nebo dokonce jiným oknem téhož procesu nebo ne). Lze tedy do okna kreslit a je věcí operačního systému zajistit, aby se grafika objevila na správném místě a ve správnou dobu. Nepřipadá tedy v úvahu situace, kdy program kreslí do zcela zakrytého okna,

a jeho kresba se přesto objeví na „povrchu“ některého z oken, které původní okno zakrývají.

Nejjednodušší grafické systémy nabízejí programátorům prostředky pro korektní kreslení do okna, nejčastěji ve formě služby, která omezí výstupní prostor zadaným obdélníkem (tzv. „clip“). Grafický výstup mimo zadaný obdélník se prostě nekreslí.

Tento mechanismus má dvě nevýhody, velmi úzce vzájemně související. V případě, že programátor službu „clip“ použije špatně, může k výše popsané situaci dojít. Z toho vyplývá, že programátor se musí velmi aktivně starat o správné „cupování“, a to je právě druhá nevýhoda - zbývá mu totiž samozřejmě méně času a pozornosti na řešení vlastního problému. Služba „clip“ je sice stále k dispozici, je však pouze jakousi nadstavbou i v případě, že jí programátor nevyužije, bude kreslení automaticky omezeno jen na viditelnou plochu okna, do kterého program právě kreslí. Programátor může „clipováním“ tuto plochu nanejvýše omezit, ale v žádném případě rozšířit. Jestliže byla část okna zakryta a uživatel přeuspořádal obrazovku tak, že se stala viditelnou nebo jestliže jsme zviditelnili dosud ukryté okno, je nutné překreslit jeho obsah.

Existuje několik metod, kterými může operační systém překreslení zajistit. Jednodušší grafické systémy umožňují využití jen jedné či dvou z nich, komplexnější systémy umožňují programátorovi zvolit momentálně nejvýhodnější metodu z následujících:

- zákaz zakrývání oken,
- volání procesu pro překreslení okna,
- zálohování obsahu okna,
- další.

Nejjednodušší, ale obvykle neakceptovatelná metoda prostě **zakáže zakrytí části okna** - pak samozřejmě není zapotřebí okno překreslovat. U moderních multitaskových systémů samozřejmě tato metoda vůbec nepřipadá v úvahu - jejím důsledkem je mimo jiné totiž to, že dokud je takové okno zobrazeno, nemůžeme aktivovat jiný proces. S touto metodou se proto setkáme jen u dialogových oken starších systémů.

Poměrně nenáročnou metodou je i **volání procesu, jemuž okno patří**, aby si potřebnou část okna laskavě překreslil sám. V řadě případů je to jediná možná metoda, a proto ji podporují snad všechny grafické systémy. Její jedinou nevýhodou je to, že ztěžuje práci programátorům a přidává jim další úkoly, o které se musí starat - samozřejmě na úkor vlastního řešeného problému. Moderní systémy proto tam, kde to dává smysl, nabízejí pro tuto metodu i různé alternativy.

Jednou ze zmíněných alternativ je **zálohování obsahu okna** - operační systém samozřejmě může všechny grafické operace provádět nad pomocným oknem mimo obrazovku a na obrazovku pak přenášet pouze ty, které odpovídají viditelným částem okna. Je-li pak zapotřebí některou část okna překreslit, stačí pouze přenést odpovídající úsek z pomocného 'mimoobrazovkového' okna. Tuto metodu podporuje volitelně většina grafických systémů. Nevýhodou je

určité zpomalení grafických operací a především značná spotřeba paměti - např. pro zálohování okna zabírajícího celou obrazovku bychom na barevném systému potřebovali řádově megabyty paměti. Praktické využití této velmi pohodlné metody proto zůstává omezeno na poměrně velmi malá okna.

Proces také může určit pouze to, že okno má být při překreslování smazáno nebo naopak vyplněno zadanou barvou. Toho je možné velmi pohodlně využít v případě, že má okno obsahovat pouze rámeček pro synovské okno (nebo pro několik synovských oken).

Zdokonalenou verzí minulé metody je případ, kdy proces operačnímu systému předá hotový obrázek, jímž má být okno automaticky při každém překreslení pokryto. To je samozřejmě velmi výhodné u různých informačních oken, jejichž obsah se mění jen málokdy nebo vůbec.

Spíše pro zajímavost uveďme další zdokonalení: proces může operačnímu systému předat ne jeden, ale libovolnou sekvenci obrázků, které pak operační systém v určených intervalech kreslí jako podklad okna. Získáme tak okno, které bude automaticky překreslováno a jeho podklad bude navíc animován.

Poslední možnost se podobá zálohování. Tentokrát si však operační systém nepamatuje skutečný obsah okna, ale grafické operace, které byly nad oknem provedeny. Je-li zapotřebí obsah okna obnovit, prostě všechny grafické operace zopakuje. Při implementaci této metody však narazíme na řadu technických i principiálních problémů (překreslení dlouho používaného okna s dlouhou řadou příkazů by např. zabralo neúměrně dlouhou dobu); obvykle proto tato metoda nebývá k dispozici.

V typických grafických systémech ovládaných myší bývá okno složeno z následujících prvků (ne každé okno ovšem musí obsahovat všechny najednou):

Jméno nebo **titulek** okna. Jméno okna je obvykle velmi stručnou informací o tom, co okno vlastně obsahuje. V titulku okna obsahujícího data načtená z nějakého souboru tak bývá nejčastěji jméno tohoto souboru a podobně.

Podtitulek. Některé systémy grafického uživatelského rozhraní umožňují zobrazit pod jménem okna ještě jeden řádek textu. Ten může buď určovat podrobněji obsah okna, nebo shrnovat zajímavé informace o okně jako celku. Okna zobrazující soubory tak často mají v podtitulku uveden počet zobrazených souborů a jejich celkovou velikost. Podtitulek můžeme samozřejmě vytvořit v libovolném okně jako součást jeho pracovní plochy. Existují však grafické systémy, které zobrazování podtitulku podporují přímo. Z hlediska uživatele programu je to naprosto lhostejné, programátorovi to může uspořit trochu práce.

Ovladač přemístění okna. Okno je umístěno 'někde' na pracovní ploše. Je zřejmé, že konkrétní umístění okna na pracovní ploše by nemělo ani tolik záviset na programu, který okno zobrazil, jako na požadavcích uživatele, který s oknem pracuje. Okna se na pracovní ploše navíc mohou překrývat. Je proto

zapotřebí, aby měl uživatel kdykoli možnost pozměnit rozložení oken na obrazovce podle toho, co momentálně potřebuje vidět a co chce naopak ponechat zakryté. Z tohoto hlediska je velmi výhodné, máme-li možnost libovolně přemísťovat i okna obsahující dialogy a výstražná hlášení. Snad všechna grafická uživatelská rozhraní pro přemístění okna využívají titulku. Jestliže ukážeme myší na titulek některého okna, stiskneme tlačítko a táhneme myší, přesunuje se obvykle zároveň celé okno a zůstane na místě, na kterém tlačítko myši uvolníme. Běžné operační systémy přemísťují po obrazovce pouze obrys okna a obsah doplní na cílové místo až po dokončení akce. Operační systémy, které mají velmi kvalitní a výkonný systém primitivních grafických operací, si však mohou dovolit přesunovat skutečně celá okna včetně obsahu - to je samozřejmě pro uživatele daleko pohodlnější.

Ovladač změny velikostí okna. Obdobná situace jako s polohou okna na pracovní ploše je i s jeho velikostí. Jestliže není velikost okna pevně určena typem zobrazovaných dat (tak tomu bývá u tzv. dialogových oken, kterými se budeme zabývat později), je opět vhodnější ponechat nastavení velikostí okna na uživateli systému. Důvody jsou stejné, jako tomu bylo v minulém případě - uživatel ví daleko lépe, kolik údajů chce v okně vidět a jak velkou část pracovní plochy chce oknem zakrýt, než program, který okno nabízí. Obvykle proto existuje i standardní metoda pro změnu velikostí okna. Nejčastěji se jedná o zvýrazněné okraje a/nebo hrany okna, které je možné uchopit myší a přetáhnout na jiné místo; okno však přitom zůstává na místě - měníme tedy pouze jeho velikost.

Nastavení standardní velikosti okna. Jakkoli je obecně lepší ponechat nastavení okna na uživateli, jsou poměrně časté i případy, kdy je program schopen nastavit velikost okna nějakým velmi smysluplným způsobem. Okno tak například může být roztaženo na co možná největší velikost - ta ovšem záleží na obrazovce, na které se okno právě nalézá, jinou možností je zvětšení či zmenšení okna tak, aby právě zobrazilo celý svůj obsah a nezabíralo přitom zbytečně mnoho místa. Může proto existovat standardní ovládací prvek, který se zeptá programu, který okno vytvořil, na ideální velikost okna a pak okno na tuto velikost nastaví. Další aktivace téhož ovladače obvykle nastaví opět původní velikost okna.

Závěr okna. Jestliže již s oknem nechceme pracovat, ale přitom ještě nechceme ani ukončit program, který okno původně vytvořil, musíme jej zavřít. Ve většině případů proto bývá součástí okna i speciální ovladač, jehož aktivací okno skutečně zavřeme. Naopak okna, která nelze zavřít (protože obsahují nějakou důležitou informaci, zobrazovanou po celou dobu práce programu), nebo okna, která se zavírají jiným způsobem (dialogová okna, zavíraná automaticky po výběru některého z voličů), ovladač pro zavírání samozřejmě nemají.

Ikonizace okna. Zvláště u multitaskových operačních systémů se snadno stane, že máme najednou na obrazovce příliš mnoho oken a ve vzniklém zmatku se již špatně vyznáme. Zavírat nebo ukrývat okna se nám ale nechce - rádi bychom je měli k dispozici na jediné klepnutí myší. Některé systémy grafického rozhraní řeší tento problém tzv. ikonizací. Ikonizované okno

z obrazovky nezmizí, pouze se zmenší na maličkou ikonu, kterou může uživatel snadno umístit někam do rohu obrazovky, kde nebude překážet (velmi často na podobné místo uloží ikonu sám operační systém). Jakmile však opět potřebujeme s oknem pracovat, stačí vybrat ikonu myší a okno je opět celé k dispozici.

Pracovní oblast okna. Hlavní součástí okna bývá obdélník, který obsahuje data nakreslená do okna programem. Tento obdélník je obvykle menší než oblast, kterou celé okno zabírá, protože na této ploše musí koexistovat se všemi ostatními součástmi okna popsanými v předcházejících odstavcích. Pracovní oblast okna může být interpretována dvěma způsoby: buď jako omezený obdélník - tak tomu bývá v dialogových oknech, o kterých budeme hovořit zanedlouho - nebo jako obdélníkový průhled na daleko větší plochu, na které jsou zobrazena všechna požadovaná data.

Posuvníky a **šipky.** Jestliže však je okno pouze průhledem do nějaké rozsáhlejší plochy, musí existovat obecný a konzistentní způsob, jak oknem po této ploše posunovat (někdo může dát přednost alternativní představě pevného okna a posunování plochy za ním) tak, abychom si mohli postupně prohlédnout celou plochu. Pro tento úkol se zpravidla používají posuvníky ('scroll bars'), šipky a jezdec posuvníku ('thumb' nebo 'slider'). Horizontální posuvníky posunují obsah okna doleva nebo doprava, vertikální posuvníky posunují obsah okna nahoru nebo dolů. Posuvník je obvykle reprezentován dlouhým a úzkým obdélníkem, který reprezentuje celou pracovní plochu za oknem, a jezdcem, který je umístěn na posuvníku a jeho pozice ukazuje pozici okna na pracovní ploše. Kvalitní grafické systémy zároveň informují uživatele relativní velikostí jezdce vůči posuvníku o relativní velikostí okna vůči ploše. Součástí posuvníku obvykle bývá několik ovladačů, které umožňují posun plochy jedním či druhým směrem o celé okno a o malý úsek.

Panely. Někdy může být vhodné, aby okno obsahovalo několik „synovských“ oken. Tato se dělí o pracovní prostor „rodičovského“ okna a každé z nich může obsahovat vlastní data, do značné míry nezávislá na ostatních oknech. V rámci uživatelského rozhraní často „synovská“ okna nazýváme panely. Panely mají nejčastěji pouze pracovní plochu, někdy mívají vlastní posuvník. Jen zcela výjimečně mívají panely vlastní titulek a ostatní ovládací prvky mívá pouze nadřízené okno.

11.5 Ikony

Ikona je malý obrázek zabírající zpravidla plochu jednoho až pěti čtverečních centimetrů. Velmi často bývá doprovázena stručným titulkem.

Ikony se používají na mnoha různých místech a pro mnoho různých úkolů; nejčastěji však reprezentují **objekty** nebo **příkazy.** Hlavním účelem ikon je umožnit rychlou identifikaci objektu na základě jeho grafické reprezentace, a ne pouze na základě jeho jména.

Většina grafických systémů disky, složky a soubory zobrazuje jako ikony doprovázené stručným titulkem obsahujícím jméno souboru (nebo disku nebo

složky). Sama ikona přitom napovídá, o co se jedná - ikona disku má tvar diskety nebo pouzdra s pevným diskem, ikona složky složku skutečně zobrazuje, ikona souboru může připomínat list papíru nebo knihu.

11.6 Nabídky

Není samozřejmě možné zajistit, aby všechny přípustné příkazy byly proveditelné nějakou akcí nad objekty viditelnými na pracovní ploše. Je například velmi jednoduché reprezentovat pomocí operací nad objekty kopírování či mazání souborů. Obtížné by však bylo zadávat tímto způsobem příkazy potřebné pro formátování disku nebo změnu atributů jednotlivých souborů. Podobné příkazy jsou proto obvykle soustředěny do **nabídek** (menu).

Většina grafických uživatelských rozhraní disponuje nabídkami složenými ze dvou úrovní:

- hlavní nabídka neobsahuje přímo příkazy, ale spíše názvy jednotlivých skupin příkazů.
- Teprve výběrem názvu některé skupiny se otevře tzv. vedlejší nabídka obsahující příkazy z vybrané skupiny.

Většina moderních grafických rozhraní dnes disponuje tzv. hierarchickými nabídkami, kdy vedlejší nabídky mohou obsahovat nejen příkazy, ale i názvy dalších skupin příkazů. Vybereme-li takový název, otevře se vedlejší nabídka na další úrovni, a můžeme vybírat příkazy z ní. Nejlepší systémy nabídek již nerozlišují nabídku hlavní a nabídky vedlejší. Kterákoli z nabídek může být zobrazena; při spuštění programu to standardně samozřejmě bývá nejvyšší nabídka v hierarchu (obdoba bývalé hlavní nabídky), uživatel si však sám může 'vytáhnout' nižší nabídky z hierarchie na obrazovku. Kterákoli nabídka přitom samozřejmě může obsahovat příkazy i názvy skupin příkazů (které vyvolají otevření další nabídky).

Kromě takovéto 'základní' nabídky mohou programy obsahovat další nabídky vázané na některé konkrétní objekty. Na okraji okna obsahujícího editovaný text může být například umístěn nadpis nabídky typů písma. Otevřeme-li pak tuto nabídku a vybereme-li z ní nějaké písmo, zobrazí se text v okně tímto typem písma. Někdy také těmto nabídkám říkáme **rozevírací nabídky.**

11.7 Dialogová okna

Srovnáme-li grafické uživatelské rozhraní s textovým systémem, vidíme, že namísto zapisování názvů příkazů můžeme příkazy vybírat z nabídek. Dosud jsme se však neseznámili s žádným způsobem, jakým bychom v grafickém uživatelském rozhraní mohli určit parametry vyvolaného příkazu.

Pro tento účel slouží tzv. dialogová okna (často říkáme pouze „dialogy“). Dialogové okno obvykle je oknem a má alespoň některé z jeho atributů. Pracovní plocha dialogového okna pak obsahuje jednotlivé ovládací prvky, pomocí nichž můžeme snadno určit parametry příkazu.

Pro zadávání parametrů obsahují dialogová okna nejčastěji následující ovládací prvky (poznamenejme, že konkrétní program samozřejmě může vždy vytvořit dialogové okno s vlastními nestandardními ovladači):

Volič (button). Jedná se obvykle o obdélníček nebo ovál, který obsahuje stručný název nějaké akce (často také ikonu). Vybereme-li volič, akce se ihned provede. Typickým voličem je ovládací prvek „OK“, který u většiny systémů znamená „parametry nastaveny, a jedem“. Dialogová okna mívají často také volič označený „cancel“; po jeho zvolení se dialogové okno zavře a žádný příkaz vyvolán není.

Volič alternativy (radio button). Tento ovládací prvek slouží pro výběr jedné z několika předem daných alternativ. Vyskytuje se proto vždy ve skupinách a v každé skupině je jeden z voličů zřetelně označen jako aktivní. Ze skupiny voličů alternativy můžeme vybrat právě jeden; volič alternativy, který byl vybrán předtím, se automaticky deaktivuje. Nastavení voliče alternativy obvykle nemívá za následek přímou akci a slouží pouze jako parametr pro příkaz aktivovaný voličem „OK“.

Přepínač (check item). Přepínače se používají pro určení jednoduché volby typu ano/ne. Jednotlivé grafické systémy používají různé způsoby rozlišení zapnutého (ano) a vypnutého (ne) přepínače; nejčastěji bývá u přepínače malý čtvereček, který je u zapnutého přepínače zaškrtnut křížkem nebo jiným znakem, zatímco u vypnutého přepínače je prázdný.

Rozevírací nabídka (popup menu). Pro případy, kdy chceme umožnit uživateli volbu jedné z alternativ, ale nemáme v dialogovém okně dostatek volného místa pro skupinu voličů alternativy, nebo kdy není předem jasné, kolik a jakých alternativ vlastně budeme nabízet, slouží rozevírací nabídka. Jedná se vlastně o nadpis běžné nabídky. Při volbě tohoto ovladače se rozevře nabídka, ze které můžeme vybrat požadovanou alternativu. Ta pak bývá zobrazena vedle rozevírací nabídky nebo přímo uvnitř jejího nadpisu jako jeho nové jméno.

Seznam (list). Existují případy, kdy potřebujeme uživateli umožnit výběr ze skupiny alternativ, které nejsou předem známy (ale určíme je až za běhu programu). Alternativ je přitom příliš mnoho, než aby bylo pohodlné použít rozevírací nabídku. Typickým příkladem je otevírání souboru, kdy musí dialogové okno nabídnout uživateli všechny přípustné soubory, aby z nich mohl vybrat ten, který se má otevřít. Grafická uživatelská rozhraní proto umožňují umístit do dialogového okna seznam všech volitelných alternativ. Je-li jich více, než dokáže seznam naráz zobrazit, objeví se na okraji seznamu posuvník který můžeme standardním způsobem používat pro procházení celého seznamu.

Text (edit item). Při používání všech dosud popsaných řídících prvků pro zadávání parametrů je poměrně málokdy zapotřebí umožnit jako některý z parametrů vkládat libovolný text. Pro ty případy, kdy to však zapotřebí je (např. zadávání jména uživatele či hesla pro přístup k síti), nabízejí grafická rozhraní i speciální ovladač pro zadávání textu. Obvykle se jedná o malý

editorek, nejčastěji jednořádkový, ve výjimečných případech několikařádkový. Ve většině případů se s tímto editorkem pracuje přesně stejným způsobem jako s textem v libovolném textovém editoru.

Statické objekty. Pro lepší přehlednost obsahuje dialogové okno obvykle množství statických objektů - textů a obrázků sloužících jako nadpisy či nápovědy k ostatním objektům v okně. Tyto statické objekty pomáhají uživateli lépe se v dialogovém okně orientovat a bezchybně využívat jeho prvky.

11.8 Výstražné dialogy

Speciálním případem dialogového okna bývají tzv. výstražné dialogy. Nejčastěji se jedná o velmi jednoduchá dialogová okna, která obsahují jeden či dva obrázky, několik řádek textu a jeden, dva či tři voliče.

Výstražné dialogy bývají používány v případech, kdy je zapotřebí uživatele informovat o nějaké situaci (viz také indikátory popsané v příštím odstavci), nebo kdy je zapotřebí vyžádat si od něj potvrzení nějaké akce či volbu mezi několika alternativami.

Výstražné dialogy mají specifické postavení mezi dialogy v tom smyslu, že obvykle vyžadují speciální pozornost; grafické systémy je proto často podporují zvláštními službami.

11.9 Indikátory

V minulém odstavci jsme se seznámili s výstražnými dialogy zobrazovanými v případě, kdy systém hlásí uživateli nějakou situaci a požaduje od něj reakci ve smyslu „beru to na vědomí“.

Existuje však řada situací, ve kterých je vhodné uživatele informovat, nelze však očekávat žádnou reakci (přesněji řečeno, nemůžeme otravovat uživatele tím, že bychom od něj zbytečnou reakci požadovali). Typickým příkladem může být kopírování souborů. Uživatel jistě přivítá, bude-li v průběhu kopírování informován o tom, kolik se toho už zkopírovalo a který soubor je právě na řadě.

Pro zobrazení informací tohoto typu se často používají tzv. indikátory. Grafická uživatelská rozhraní poměrně málokdy zajišťují automaticky služby indikátorů a ponechávají jejich vytváření a obsluhu na programátorovi.

Indikátor může být v jednoduchém případě tvořen prostou textovou informací („Moment, prosím ... pracuji“); v praxi je vhodnější, jestliže indikátor udává nějakou vhodnou grafickou formou postup prováděné operace. Takové indikátory bývají nejčastěji obdélníkové nebo kruhové.

11.10 Kurzory

Součástí grafického uživatelského rozhraní je téměř vždy i kurzor, který na obrazovce sleduje pohyby myši. Nejčastěji bývá reprezentován stylizovanou šipkou. Velmi často však je uživatelské rozhraní vytvořeno tak, že tvar kurzoru napovídá typ operace, kterou budeme provádět. Uveďme několik typických příkladů:

- Snad nejčastější je to, že je-li kurzor nad oblastí, do které můžeme zapisovat text, změní se na znak připomínající velké písmeno I.
- Při práci s grafikou bývá zvykem reprezentovat kurzor pomocí malého křížku. Pomocí takového kurzoru se pohodlněji vybírají přesné pozice potřebné při zpracování grafických dat.
- Při vybírání objektů se kurzor často změní na stylizovaný obrázek ruky s ukazujícím prstem.
- V době, kdy program pracuje a není momentálně schopen reagovat na podněty uživatele, se kurzor obvykle změní na nějaký obrázek, který tuto situaci indikuje (hodiny). V tomto smyslu slouží kurzor jako jednoduchý indikátor toho, že program pracuje.
- Při přemísťování objektu pomocí myši může kurzor měnit podobu podle toho, k jaké akci by došlo, kdybychom objekt 'pustili' - např. dvojitý čtvereček znázorní kopírování, 'rozsvícená' šipka přemístění souboru, dvojitá šipka vytvoření symbolického odkazu na objekt a obyčejná šipka znamená, že se nestane nic.

11.11 Aplikace

Součástí většiny dnešních operačních systémů je i několik programů, které lze jen těžko považovat za nutné příkazy operačního systému. Tyto programy spíše zajišťují, aby byl počítač jakžtakž použitelný i bez nákupu dalších komerčních programů.

Snad každý operační systém je kupříkladu vybaven jednoduchým textovým editorem - původně bylo důvodem pravděpodobně to, že většina operačních systémů obsahuje konfigurační údaje uložené v textových souborech; správce systému proto potřebuje textový editor jako základní prostředek pro svou činnost. Dnes řada systémů nabízí pro konfiguraci samostatné aplikace s pohodlným grafickým uživatelským rozhraním; editor však součástí systému zůstává jako jakýsi 'nadbytečný luxus', k němuž se často přidávají i další aplikační programy.

Jak jsme se zmínili, bývají součástí systému i programy sloužící systémovému administrátorovi. Jejich služby zahrnují konfiguraci sítě, správu uživatelských kont a přístupových práv jednotlivých uživatelů k nejrůznějším zařízením, správu sdílení souborů a řadu dalších prostředků.

Mnoho operačních systémů obsahuje i více či méně luxusní terminálový program, umožňující navázat spojení s jiným počítačem prostřednictvím jednoduché sériové linky (nebo modemu a telefonu).

11.12 Vývojové prostředí

Je vhodné, aby nedílnou součástí operačního systému bylo i dobře navržené vývojové prostředí pro tvorbu aplikací. Nejenže to usnadní život programátorům, ale navíc to umožní snadné sdílení knihoven (nebo objektových balíků) mezi jednotlivými programátory a tedy další vzrůst produktivity.

Podaří-li se navíc připravit vývojové prostředí, které je snadno použitelné, intuitivní a lehce zvládnutelné, přináší to další výhodu - zkušenější uživatelé, kteří nejsou profesionálními programátory, si mohou sami vytvářet velmi jednoduché jednoúčelové programy.

Naprostá většina operačních systémů pro mikropočítače řeší vývojové prostředí interpretem jazyka BASIC, poměrně snadno přístupného i neprogramátorům. UNIX, který je určen spíše zkušenějším uživatelům má vývojové prostředí s jazykem C.

Kontrolní otázky:

1. Popište funkce vrstev grafického systému OS.
2. Vysvětlete funkce oken v OS.?

Úkoly k zamyšlení:

1. Zamyslete se nad výhodami a nevýhodami Windows z pohledu ovládání prostřednictvím oken.

Korespondenční úkol:

1. Odvoďte velikost paměti grafického adaptéru v závislosti na počtu bodů zobrazovaným na obrazovce a množství barev jednotlivých bodů.

Shrnutí obsahu kapitoly

V této kapitole jste se seznámili s důvody obecných znalostí principů operačních systémů. Důraz v této kapitole byl kladen na pochopení rozhraní člověk/stroj a proces/operační systém. Velká pozornost byla věnována vysvětlení zájmů operačních systémů.

12 Systém služeb

V této kapitole se dozvíte:

- Jaké služby využívá operační systém a jak jsou implementovány?

Po jejím prostudování byste měli být schopni:

- Charakterizovat základní služby operačních systémů.
- Popsat způsob implementace služeb v operačním systému.
- Porozumět pojmům – knihovna, server, sdílená knihovna, objekt, textová služba, národní prostředí, schránka, systém drag and drop – ve vazbě na systém služeb.
- Popsat řešení sdílených a kopírovaných dat.

Klíčová slova této kapitoly:

Služba OS, sdílená knihovna, server, schránka, systém darg and drop..

Doba potřebná ke studiu: 2 hodiny

Průvodce studiem

Studium této kapitoly je potřebné pro objasnění dalších funkcí operačního systému tzv. služeb.
Na studium této části si vyhraďte 2 hodiny. Po celkovém prostudování a vyřešení všech příkladů doporučujeme vypracovat korespondenční úkol.

Operační systém má ještě jednu velmi důležitou úlohu. Musí zajistit maximální množství často požadovaných služeb tak, aby každý programátor nemusel vytvářet vždy znovu a znovu stejný kód. Jak se postupně vyvíjely operační systémy, objevovalo se samozřejmě čím dál tím více takových služeb. I když nejdůležitější částí každého operačního systému zůstává správa prostředků, mohou být co do objemu ostatní služby daleko rozsáhlejší.

Kromě vlastního množství služeb je samozřejmě nesmírně důležitá i jejich univerzálnost a možnost jejich mírného přizpůsobení konkrétním požadavkům právě vyvíjeného programu tak, aby programátor byl nucen vytvářet skutečně jen a pouze nový kód. Zde se znovu a daleko výrazněji objevuje výhoda objektově orientovaných operačních systémů, jako je dědičnost objektových programátorských prostředků, a dává službám těchto systémů daleko větší flexibilitu, než tomu je u obdobných služeb systémů ostatních.

12.1 Jaké služby potřebujeme

Od samého začátku samozřejmě všechny operační systémy nabízely programátorům alespoň základní služby pro přidělování a uvolňování operační paměti (nemáme-li k dispozici virtuální adresový prostor, ani by to bez nich nešlo) a poměrně komfortní služby pro práci se soubory. Prostřednictvím ovladačů zařízení, připojených k systému, měli programátoři navíc k dispozici i služby pro práci s těmito zařízeními. Málokdy se však jednalo o služby dostatečně vysoké úrovně. Multitaskové operační systémy pak samozřejmě

musí nabízet služby správce procesů. Pro současné operační systémy to ani zdaleka nestačí. Operační systém musí nabízet poměrně velmi luxusní služby alespoň v následujících oblastech:

- Grafické operace, práce s okny a graficky orientovaná komunikace s uživatelem. Do této oblasti patří např. volba stylu písma, volba barvy, editace textu v rozsahu jediného řádku i na úrovni kompletního editoru a celý komplex tiskových služeb (v rozumném operačním systému je tisk textu pouze speciálním případem tisku grafiky).
- Práce s textovými řetězci a s bloky dat v operační paměti. Nejrůznější přesunování textů nebo skupin bytů, vyhledávání v nich a jejich kombinace patří snad mezi nejčastější základní operace, které kterýkoli program provádí. Zpracování textu pak je základní součástí prakticky všech programů, které vůbec nějak komunikují s uživatelem.
- Tato problematika velmi úzce souvisí s graficky orientovanou komunikací s uživatelem - je totiž nutné si uvědomit, že práce s textem nekončí u knihoven jazyka C a standardních služeb pro kopírování či připojení textového řetězce. Služby moderního operačního systému musí být schopné pracovat s formátovaným textem obsahujícím údaje o použitém fontu, velikostí a typu písma, o zarovnání jednotlivých odstavců, o použité barvě .
- Není-li operační systém určen výhradně pro procesory vybavené jednotkou pro výpočty v pohyblivé řádové čárce (nebo alespoň odpovídajícím koprocesorem), je zapotřebí, aby obsahoval rozsáhlou skupinu služeb pro práci s neceločíselnými hodnotami.
- Téměř pro každé zařízení (jako příklad jmenujme zvukový vstup a výstup nebo systémové hodiny) by měl operační systém nabízet skupinu služeb, které samy zpracují nejběžnější požadavky kladené na zařízení. S našimi příklady by tedy systém jistě neměl ponechávat na každém programátorovi, aby se sám staral o generování tónů požadované výšky, intenzity, délky a barvy nebo o přehrávání samplovaného zvukového záznamu. Analogicky pro zvukový vstup by měl systém zajistit dekódování a případné uložení samplovaného záznamu.
- Nesmírně důležitou oblastí je podpora práce v cizím jazyce. Operační systém by měl nabízet řadu prostředků umožňujících jak práci s dokumenty psanými v jiném jazyce, než pro který byl systém původně navržen, tak i vlastní komunikaci s uživatelem v jeho jazyce. Nejmodernější operační systémy pak mohou s každým ze svých uživatelů komunikovat jinou řečí podle jeho osobní volby.
- Jednou z nejčastějších problematik řešených na počítačích je ukládání a opětovné vyhledávání údajů - tedy databáze. Dokonce i řada nedatabázových úloh se dá vyřešit daleko pohodlnějším a efektivnějším způsobem, je-li možné jako základ řešení použít již hotový, fungující a rychlý databázový systém. Programátoři by proto v moderním operačním systému měli mít k dispozici i poměrně velmi rozsáhlý balík databázových služeb.
- Velmi významnou oblastí je i komunikace mezi programy na vyšší úrovni. Jednotlivé procesy samozřejmě mohou snadno komunikovat s využitím aparátu zpráv nebo semaforů, které jim nabízí správce procesů. Je však zapotřebí umožnit aplikacím, aby si mohly bez

extrémního úsilí a explicitní dohody jejich programátorů navzájem předávat data ke zpracování, informace o stavu těchto dat a o jejich případné modifikaci a podobně.

Uvedený seznam pochopitelně není vyčerpávající. Není nepravděpodobné, že s rozvojem výpočetní techniky se budou povinné služby (ve smyslu konkurenceschopnosti operačního systému) rozšiřovat a že během času budou zahrnovat např. univerzální prázdný expertní systém nebo automatické překlady z jednoho jazyka do druhého. V každém případě se již dnes pomalu mezi povinné služby operačního systému řadí třeba korektor pravopisu. Nejmodernější systémy jdou ještě mnohem dále. Kterékoli aplikaci pracující s textem, je přístupný kompletní výkladový slovník s mnoha desítkami tisíc pojmů.

12.2 Implementace služeb

V klasických operačních systémech existovaly v zásadě dvě možnosti implementace služeb. Buď se mohlo jednat o plnohodnotné systémové služby, které operační systém zajišťoval na vyžádání, nebo mohly být služby uloženy na knihovnách, odkud se při vytváření aplikačních programů připojily k jejich kódu.

Dnešní operační systémy obvykle disponují dvěma dalšími prostředky pro implementaci služeb. Jedná se o servery a o sdílené knihovny. Nejmodernější objektově orientované operační systémy pak mají k dispozici navíc také aparát komunikace objektů.

V praxi operační systém pro zajištění služeb obvykle kombinuje všechny popsané metody. Podívejme se proto postupně na jejich princip, výhody a nevýhody.

12.2.1 Služby systému

Služby systému jsou obvykle vyvolávány pomocí speciální instrukce procesoru, která přepne pracovní režim z uživatelského do systémového a předá řízení předem zvolené rutině operačního systému. Ta na základě vstupních parametrů zjistí, kterou službu program požadoval a předá řízení rutině, která služby zpracuje. Nejčastěji se používají instrukce pro vyvolání programového přerušení.

Parametry se při vyvolání systémových služeb nejčastěji předávají v registrech. To je výhoda z hlediska rychlosti zpracování služby, je to však nepohodlné v případě, kdy potřebujeme předávat větší množství formátovaných parametrů a musíme je nějak umístit do několika bezformátových registrů. Systémové služby proto v praxi bývají velmi často „zabaleny“ do velmi jednoduchých služeb (umístěných na klasických knihovnách), které pouze překódují parametry z pohodlného tvaru uloženého nejčastěji na zásobníku do registrů a pak zavolají odpovídající systémovou službu.

Další a nejvýznamnější nevýhodou systémových služeb je právě to, že jsou přímo součásti operačního systému. Takový operační systém by měl obrovský

počet služeb a tudíž by se sám nevešel do paměti (pokud bychom neměli k dispozici virtuální paměť). Navíc by v obrovském operačním systému - i přesto, že by byl samozřejmě složen a řady modulů - přece jen narůstalo riziko chyb.

Hlavní výhodou systémových služeb je jejich „právo dělat cokoli“. Jsou-li služby součástí systému, je také jejich kód zpracováván v systémovém režimu procesoru. Mohou tedy přímo ovládat jednotlivá zařízení nebo kanály, mohou maskovat jednotlivá přerušení, mohou přeprogramovávat jednotku řízení paměti a podobně. Systémové služby proto musí v každém případě pokrýt všechny úlohy, pro které je provádění takovýchto potenciálně nebezpečných akcí nezbytné. Ostatní úkoly - zpravidla daleko vyšší úrovně - pak mohou být realizovány jinak.

Při určitém zjednodušení může být dobrým příkladem třeba práce s diskem. Služby systému by mohly zajišťovat pouze přečtení a zápis požadovaného bloku dat z disku a na disk (protože pro splnění tohoto úkolu je samozřejmě nutné přímo komunikovat s řadičem disku). Všechny ostatní úkoly by mohly být realizovány mimo vlastní systém.

12.2.2 Klasické knihovny

Program se obvykle vytváří ve dvou krocích:

- Nejprve se pomocí překladačů vytvoří jednotlivé moduly. V modulech samozřejmě je množství volání nejrůznějších služeb. Na každém takovém místě je instrukce volání podprogramu bez cílové adresy, namísto této adresy je v modulu uloženo jméno požadované služby. Modul může nějakou službu také sám nabízet. Pak je jeho součástí informace o jménu služby a o relativní adrese odpovídajícího podprogramu uvnitř modulu.
- Druhým krokem je spojení všech modulů dohromady. Spojovací program přitom má k dispozici nejen moduly, které vytvořil programátor aplikace, ale i množství modulů, které jsou uloženy právě v systémových knihovnách. Program pak vybere všechny moduly z knihoven, které obsahují služby volané z „aplikačních“ modulů, a ze všech modulů dohromady vytvoří hotový program. Všechny instrukce volání podprogramů přitom doplní správnými adresami.

Ve výsledném programu je tedy uložen kompletní kód služeb z knihoven, stejně, jako by jej programátor zapsal při tvorbě programu.

To je samozřejmě současně hlavní nevýhodou klasických knihoven. Máme-li totiž potom sto programů používajících služby z klasických knihoven na disku, máme na disku sto jedna kopií téhož kódu (sto v programech, sto prvá kopie je uvnitř knihovny, která je na disku obvykle uložena také). Zavedeme-li v multitaskovém prostředí třicet takovýchto programů do paměti, budeme v ní mít opět třicet kopií téhož kódu.

Žádnou významnější výhodu klasické knihovny nemají (aparát sdílených knihoven - není-li pro něj využit systém virtualizace adres - může znamenat určité zvýšení režie; není to však obvyklé). Asi hlavním důvodem, proč

klasické knihovny dosud ani v moderních operačních systémech nevymizely je to, že jsme na jejich používání zvyklí.

V dnešních operačních systémech klasické knihovny obvykle obsahují pouze kratičké pomocné funkce, které převedou normální volání podprogramu na vyvolání systémové služby nebo na odeslání zprávy serveru.

Klasické knihovny mohou sloužit i pro dosažení kompatibility se staršími systémy nebo pro usnadnění práce programátorům, kteří jsou zvyklí na klasické prostředky a nechtějí přecházet na objektově orientované.

12.2.3 Servery

Na úvod je třeba si uvědomit, že server nemusí být určen pouze pro obsluhu nějakého fyzického zařízení, ale může také naprosto stejným způsobem nabízet služby ovládající nějaké 'zařízení' logické - např. systém souborů nebo databází.

Výhody a nevýhody serverů se do jisté míry podobají výhodám a nevýhodám služeb samotného operačního systému. Hlavní rozdíl spočívá v tom, že servery obvykle nepracují v systémovém režimu procesoru (a musí tedy samy využívat systémových služeb pro přímé ovládání zařízení). Zůstává společná nevýhoda zabrané paměti (není-li k dispozici virtuální paměť) i nevýhoda snadnějšího zavedení chyb do velkého programu, jakým takový server obvykle bývá. Vzhledem k tomu, že pro každý úkol může být určen samostatný server, však tato nevýhoda není ani zdaleka tak markantní, jako tomu bylo v případě systémových služeb.

Servery jsou relativně samostatné. Kód ze sdílených knihoven je naproti tomu součástí programu, který jej využívá. Servery proto častěji slouží tam, kde je zapotřebí provádět nejen akce na přímou výzvu programu, ale i akce asynchronní, „o vlastní vůli“. Dobrým příkladem může být třeba správa souborů (s asynchronní obsluhou cache pamětí) nebo inteligentní ovladač obrazovky (který požadovaný výstup dokresluje také asynchronně).

12.2.4 Sdílené knihovny

Z programátorského hlediska je v případě opakovaného používání stejného podprogramu a tím i udržování tolika kopií jediného kusu kódu nešikovné. Řešení je právě ve sdílení knihoven. Jejich funkce je podobná funkci klasických knihoven, právě jen s jediným podstatným rozdílem. Není zapotřebí, aby na disku nebo v paměti byly více než jednou (máme-li k dispozici systém virtuální paměti, stačí jediná kopie pro disk i pro paměť dohromady).

Řešení spočívá v tom, že spojovací program svou práci nedodělá a ponechá to na zavaděči, který ukládá programy při spuštění do operační paměti. Celý mechanismus pak vypadá přibližně takto:

- Nejprve se pomocí překladačů vytvoří jednotlivé moduly, naprosto stejně jako tomu bylo v případě knihoven klasických.

- Druhým krokem je opět spojení všech modulů dohromady. Spojovací program však tentokrát vyřeší pouze vzájemné odkazy mezi moduly, které vytvořil programátor aplikace. Sdílených knihoven si nevšímá a odkazy na služby, které jsou na sdílených knihovnách uloženy, ponechá beze změny.
- Teprve při zavádění programu do paměti zavaděč zjistí, které ze sdílených knihoven jsou zapotřebí. Pak ověří, nejsou-li již tyto knihovny v paměti (jako důsledek zavedení některého z minulých programů). Jestliže tomu tak není, do paměti je zavede.
- Potom teprve zavaděč uloží do paměti i kód spouštěného programu, přitom jeho instrukce pro volání podprogramů ze sdílených knihoven doplní správnými adresami, odvozenými od umístění sdílené knihovny v paměti.

Z využití sdílených knihoven vyplývá navíc jedna podstatná výhoda, která přináší do systémů se sdílenými knihovnami určitý rys systémů objektových. Jestliže totiž nahradíme starou verzi operačního systému verzí novější, budou všechny programy - i ty, které byly dohotoveny v době, kdy se nikomu ještě o nové verzi systému nesnilo - automaticky při zavádění spojovány s novými knihovnami a budou tedy automaticky využívat jejich výhod (odstraněných chyb, doplněných nových funkcí a podobně).

12.2.5 Objekty

Objekt je z hlediska pravého objektového systému jakási „černá skříňka“. Objektu můžeme poslat nějakou zprávu a můžeme si počkat na odpověď. Objekt může také na základě zprávy začít zpracovávat nějakou časově náročnější akci a její ukončení může naopak zprávou ohlásit nám ('my' - tedy hlavní program - jsme samozřejmě také objektem).

Implementace některého z objektově orientovaných programovacích jazyků existují dnes snad již pro každý operační systém. To však je do jisté míry „podvod“ - z hlediska programátora totiž překladač převede plnoprávné objekty na naprosto obvyklý kód využívající běžné volání podprogramů.

Naprosto jiná je situace v pravých objektových operačních systémech. Tam objekt zůstane objektem - tedy nezávislou „černou skříňkou“ - i po překladu a operační systém sám nabízí pohodlné a efektivní prostředky pro komunikaci mezi nimi. Pak je ovšem snadné využít právě těchto prostředků pro volání služeb. Odpovídající kód je ukryt v sadě hotových objektů, dodávaných s operačním systémem.

Toto řešení je zhruba na půli cesty mezi sdílenými knihovnami a servery - objekt není (respektive nemusí být) samostatným procesem jako server. Je však daleko samostatnější než podprogram, uložený na sdílené knihovně. Z hlediska samotného volání služeb zde žádné výrazné rozdíly nejsou, objekty se podobají spíše knihovnám, sám aparát komunikace s nimi je podobnější komunikaci se servery. Zde je třeba si uvědomit, že hlavní výhodou objektů proti ostatním metodám je dědičnost, která programátorům umožňuje velmi pohodlným a intuitivním způsobem mírně modifikovat činnost služeb, které využívají.

12.3 Obsah služeb

Zatímco obsah některých skupin služeb (jmenujme např. služby pro neceločíselnou aritmetiku) je celkem jasný, k některým dalším oblastem je vhodné uvést ještě podrobnější rozbor. Podíváme se proto postupně na textové služby, na podporu práce v národním prostředí, na databázový systém a na komunikaci na vyšší úrovni.

12.3.1 Textové služby

Textové služby můžeme velmi snadno rozdělit do dvou skupin. V první z nich jsou jednoduché služby pro nejrůznější přesuny dat v operační paměti; na nich není dohromady nic zajímavého (snad jen může být vhodné si uvědomit, že se nejedná jen o pohodlí programátora - operační systém může často přesuny větších bloků dat zajistit rychleji než aplikační program, protože mívá k dispozici speciální vybavení.

Druhá skupina je komplikovanější. Jedná se o zpracování formátovaných textů zahrnujících různé fonty, velikostí písma, barvy a podobně. Je nutné, aby sám operační systém práci s formátovaným textem podporoval, a to ze dvou důvodů:

- Práce s formátovaným textem je příliš častým případem, než aby se vyplatilo ji programovat pokaždé znovu.
- Druhým a hlavním důvodem je komunikace programů. Jde o to, aby bylo možné bez problémů přenášet formátovaný text mezi dvěma aplikacemi, jejichž tvůrci se na ničem nedomluvili.

Služby pro práci s formátovaným textem (i některé služby pro práci s textem neformátovaným) navíc velmi úzce souvisejí s podporou národního prostředí, které věnujeme příští odstavec. Součástí mnoha služeb je totiž také interpretace jednotlivých znaků (např. při převodu malých písmen na velká nebo při dělení odstavce na řádky). Operační systém musí v takovém případě korektně bez extrémního úsilí programátora konkrétní aplikace zajistit správnou interpretaci národních znaků.

12.3.2 Národní prostředí

Podpora národního prostředí se dá snadno a nenásilně rozdělit na dvě skupiny služeb:

- První skupina umožňuje uživateli počítače zpracovávat dokumenty v různých jazycích.
- Druhá skupina služeb pak je určena k tomu, aby uživatel mohl s počítačem komunikovat ve své rodné řeči.

Obě skupiny jsou velmi komplexní. Mezi hlavní součásti prvé skupiny patří:

- Možnost **zobrazovat speciální znaky**, potřebné pro požadovaný jazyk, na obrazovce počítače. Český uživatel dobře ví, že to není zcela triviální ani s našimi diakritickými znaménky. Skutečné problémy však nastávají s východními abecedami, jejichž znaky jsou velmi komplikované a bývá jich značný počet (např. čínské a japonské

abecedy). Situace je ještě o to komplikovanější, že některé východní jazyky mají pro jediné písmeno různé 'obrázky' v závislosti na kontextu.

- Neméně důležitá je i možnost **přenést texty** v požadovaném jazyce na všechna potřebná výstupní zařízení - může se jednat o tiskárnu, fax, osvitovou jednotku nebo třeba stroj na výrobu kreditních karet. Zde je velkou výhodou grafický subsystém se službami vysoké úrovně, který umožňuje využití jediného mechanismu zobrazování pro kterékoli výstupní zařízení. Pak stačí vyřešit zobrazování národních znaků jen jednou a není nutné se jím zabývat pro každé nové výstupní zařízení vždy znovu a znovu.
- Nemalým problémem je i vhodné **ovládání klávesnice**. Řada jazyků má příliš mnoho znaků, než aby bylo možné je rozumným způsobem poskládat na běžnou počítačovou klávesnici. Je proto nutné zavádět tzv. mrtvé klávesy, různé pracovní režimy a další speciální prostředky pro vstup znaků.

Úplná podpora národních prostředí musí zajišťovat také:

- Klasifikaci a převody znaků. Mnoho programátorů v jazyce C např. používá často standardní službu 'isupper', která ověří, je-li zadaný znak velkým písmenem. Málokdo si však přitom uvědomí, že pokud tato služba nepozná také velká písmena 'Á', 'Č', 'Ď', ..., není její implementace korektní.
- Totéž platí pro převody malých a velkých písmen. Standardní systémové služby musí převádět korektně všechny znaky, včetně národních, a musí být automaticky přístupné všem programátorům prostřednictvím standardních funkcí.

Řadu stále složitějších a složitějších problémů vyvolává tak základní úkol, jakým je netřídění několika slov nebo vět. Operační systém samozřejmě musí třídění zajišťovat sám v závislosti na aktivním jazyce. Při implementaci třídicích rutin je však zapotřebí:

- Nejprve zajistit, aby vůbec všechna písmena měla správné pořadí pro češtinu tedy např. musí být 'á' před 'b', ačkoli má ve všech běžně užívaných kódováních vyšší kód.
- Pak narazíme na písmena, složená z dvojice znaků, která se však z hlediska třídění chovají skutečně jako jediné písmeno (například v češtině máme „ch“).
- Po vyřešení obou problémů zjistíme, že třídění v některých jazycích - mezi které čeština patří - dosud není korektní. Správné třídění v nich totiž z jakýchsi prapodivných a nepochopitelných důvodů není lexikografické, ale daleko složitější - např. správné pořadí českých slov.
- Vyřešíme-li přece jen všechny problémy a implementujeme korektní třídění, musíme vyřešit také problém, kde jej použít a kde ne. Má být např. seznam souborů netříděn lexikograficky nebo podle aktivního jazyka?

Součástí podpory národního prostředí je i překlad automaticky generovaných údajů (jako jsou data, jména dní a měsíců spod.). Často je překlad velmi obtížný, protože v některých jazycích - mezi nimiž samozřejmě čeština patří na

čestné místo - se slova ohýbají, takže jedno slovo může v různých kontextech vypadat jinak.

Komplexní textové služby na národním prostřední závisejí také velmi podstatným způsobem - uvědomme si například, že nemalé množství jazyků standardně píše zprava doleva a některé píší dokonce ve sloupcích. Služby operačního systému by samozřejmě měly podporovat i toto.

Nikoli nepodstatnými službami systému je i podpora pro dělení slov na konci řádků a korektor (spellchecker).

Druhou skupinou služeb, které musí operační systém zajistit, je podpora komunikace s uživatelem ve zvoleném jazyce.

Zásadním problémem je pochopitelně vlastní překlad textů. Přitom však je nutné si uvědomit, že je často zapotřebí přeložit i obrázky, které mohou mít výrazně odlišný význam v jiné kulturní oblasti. Černobíle zobrazená pěticípá hvězda např. v českém uživateli vyvolá zcela jiné asociace a představy než v Američanovi.

To jsou však všechno problémy, které musí řešit a vyřešit ten, kdo bude připravovat operační systém a jeho obslužné programy pro práci v některém konkrétním jazyce. Sám operační systém má jiný úkol - musí obsahovat podporu pro práci vícejazyčných aplikací.

Vícejazyčná aplikace je program, který obsahuje komunikační prvky pro několik různých jazyků. Teprve ve chvíli spuštění se ve spolupráci s operačním systémem automaticky zvou jeden z těchto jazyků - ten, který uživateli nejlépe vyhovuje - a kdykoli pak program použije některý z komunikačních prvků, jistí systémové služby automaticky použití prvku z tohoto jazyka.

Tvorba vícejazyčných aplikací je samozřejmě v zásadě možná pod libovolným operačním systémem, který vůbec aplikaci umožní zjistit, jaký jazyk je právě platný.

12.3.3 Databáze

Přístup k datovým souborům na úrovni binárního nebo textového souboru se sekvenčním nebo přímým přístupem, jakým disponuje naprostá většina dnešních operačních systémů, je samozřejmě věc nutná a potřebná, ale zdaleka ne postačující.

Moderní operační systém musí zajišťovat také služby vyšší úrovně pro práci s formátovanými soubory, obsahujícími databáze s proměnnou délkou záznamu, se sekvenční nebo s indexsekvenční organizací.

Tento typ souborů může s výhodou využít naprostá většina aplikačních programů. Je proto zbytečné, aby každý programátor vytvářel vlastní databázový systém - zbytečně by tak utrácel sí1y, které může v kvalitnějším operačním systému věnovat na řešení konkrétního problému.

I v případě, že existuje řada nejrůznějších databázových systémů, které pracují pod daným operačním systémem a nabízejí programátorům své služby, narazíme na jeden zásadní problém. Databáze vytvořené pod různými systémy budou vzájemně nekompatibilní a čas, který jejich programátoři ušetří na vlastní tvorbě databázového systému opět ztratí při programování nejrůznějších převodníků z jednoho systému do druhého.

U operačních systémů určených pro výkonnější pracovní stanice samozřejmě z analogických důvodů nestačí pouhá podpora indexsekvenčních souborů. Bylo by vhodné, aby operační systém obsahoval plnohodnotný databázový server, schopný obsluhovat rozsáhlé a komplikované databáze a schopný nabízet své služby po síti (a - alespoň pokud se neobjeví dokonalejší systém - podporující jazyk SQL).

Takový server je často velmi komplikovaný a drahý program a jeho zařazení do systému by neúměrně zvýšilo cenu celého operačního systému pro tu skupinu uživatelů, kteří jeho služby nepotřebují. Řada moderních operačních systémů proto tento problém řeší tak, že obsahuje luxusní služby pro zcela standardizované využití kteréhokoli z komerčně přístupných databázových serverů. Jestliže si tedy uživatel server - a to jakýkoli - pořídí, stanou se služby serveru stejně snadno přístupné všem aplikacím a programátorům, jako by byl server přímo součástí operačního systému.

12.3.4 Komunikace programů

Komunikace programů na této úrovni je něco zcela jiného než komunikace procesů. Hlavním úkolem komunikace procesů bylo zajistit jejich synchronizaci. Tentokrát se o synchronizaci nezajímáme (přesněji řečeno, musí být zajištěna v nižších vrstvách tak, aby fungovala a my abychom se o ni nemuseli starat); naším úkolem je zajistit spolupráci několika programů nad společnými daty.

Čteme například nějakou zajímavou zprávu v elektronické poště a rádi bychom z ní některé úryvky přebrali do článku, který máme rozepsaný v nějakém textovém editoru. Tuto úlohu lze samozřejmě vyřešit uložením zprávy do souboru, načtením tohoto souboru do textového editoru a vybráním požadovaných úseků; to však je velmi nepohodlné. Pokud bychom četli tutéž zprávu v moderním operačním systému, můžeme každý zajímavý úsek přímo v elektronické poště označit, jedním příkazem přenést do systémové komunikační oblasti (tzv. schránky), pak aktivovat editor a dalším příkazem text ze schránky vložit na libovolné místo v rozepsaném článku.

V další zprávě můžeme narazit na jméno, které je nám povědomé, ale nemůžeme si jej vybavit. Zkusíme jej tedy vyhledat v databázi svých známých, ale ouha - pro vyhledávání musíme jméno opsat ručně (s rizikem případného překlepu). Pak jméno opět jediným příkazem uložíme do schránky a v databázi pohodlně obsahem schránky odpovíme na otázku „co se má hledat“.

Představme si, že čteme formátovaný text s různými fonty, velikostmi, typy písma a rádi bychom jej přenesli do několika dalších dokumentů. Některé z

nich však jsou formátované a tam bychom rádi formátování zachovali. Jiné obsahují jen ASCII text a tam chceme přenést alespoň text bez formátu.

Je nutné si uvědomit, že komunikaci skutečně musí zajišťovat sám operační systém. Bylo by totiž sice poměrně jednoduché, kterýkoli z dále popsaných mechanismů implementovat bez podpory systému. Problém je ale v tom, že by fungoval pouze mezi několika málo programy. Mají-li být systémy komunikace uživateli počítač skutečnou pomůckou, musí korektně pracovat s naprostou většinou aplikací.

Ukažme si některé systémy komunikace mezi programy, které jsou dnes k dispozici u většiny operačních systémů. Jedná se především o:

- schránku,
- inteligentní schránku,
- spojení dat,
- systém drag and drop,
- datové služby.

12.3.4.1 Schránka

Rozumně implementovaná schránka by měla být schopna obsahovat zároveň data v několika formátech. To je proto, že ve chvíli, kdy ukládáme data do schránky, není jasné, kdo je ze schránky bude odebírat. Ukládáme-li tedy např. obrázek v kreslicím programu, měl by program do schránky uložit obrázek dvakrát. Jednou ve vektorovém formátu, zachovávajícím všechny informace o struktuře obrázku a umožňujícím jej dále upravovat, a jednou jako bitovou mapu (snímek obrazovky). Bude-li později chtít údaje ze schránky převzít program, který nedokáže komplikovanější vektorový formát zpracovávat, bude mít k dispozici alespoň bitovou mapu.

Z toho ovšem plyne i určitá nevýhoda schránky. Poměrně snadno se může stát, že některý program data ze schránky nebude schopen převzít, ačkoli v principu by je zpracovat mohl. Jestliže třeba náš kreslicí program do schránky uloží obrázek ve vektorovém formátu s barevnou informací a zároveň jako bitovou mapu, nebudeme moci obsah schránky použít v černobílém vektorovém kreslicím programu.

Tento problém by se dal řešit (nebo spíše minimalizovat) tak, že by každý program do schránky ukládal data ve všech možných formátech. To však přináší jinou nevýhodu: taková schránka by zabrala neúměrně mnoho paměti a přenos dat do ní by trval velmi dlouho.

Bylo by zřejmě daleko výhodnější, kdyby se oba programy domluvily na tom, v jakém formátu se vlastně mají data předávat. Takový systém opravdu existuje a podíváme se na něj v následujícím odstavci.

12.3.4.2 Inteligentní schránka

Schránka moderních operačních systémů je implementována trochu jiným způsobem, i když z pohledu uživatele funguje naprosto stejně. Program, ze kterého chceme data přenést, ve skutečnosti do schránky žádná data neukládá

(a odpovídající příkaz díky tomu také trvá mnohem kratší dobu). Namísto toho však do schránky uloží informaci o všech formátech, ve kterých je schopen data exportovat.

Teprve ve chvíli, kdy se pokusíme přebrat obsah schránky v cílovém programu, dojde k vlastnímu přenosu. Cílový program nejprve zjistí, zda dokáže zpracovat data v některém z formátů zapsaných ve schránce. Jestliže ano, vyžádá si prostřednictvím operačního systému od zdrojového programu data právě v tom formátu, na kterém se oba shodli.

Tento mechanismus přináší dvě výhody:

- Data se přenášejí pouze v jediném formátu, ne ve všech potenciálně potřebných; tak se šetří operační paměť i čas.
- Díky tomu si programy mohou dovolit nabízet daleko více datových formátů. Je tedy mnohem větší pravděpodobnost, že se na některém z nich dva programy shodnou.

Inteligentní schránka má nicméně i jednu nevýhodu, kterou pozorný čtenář již jistě odhalil sám. Ve chvíli přenosu musí oba programy běžet. Klasická schránka naproti tomu může obsahovat data uložená některým programem ještě dávno poté, co byl program ukončen. Proto může inteligentní schránka snadno fungovat u preemptivních operačních systémů, kde současný běh řady procesů nepřináší žádné problémy. Proto by mohla být jen s obtížemi implementována na kooperativním multitaskingu, kdy je uživateli na obtíž již případ, kdy na pozadí běží jeden jediný proces.

12.3.4.3 Spojení dat

Schránka - ať již klasická nebo inteligentní - je velmi pohodlná, dokud přenášíme hotové údaje z jednoho místa na druhé. Jakmile však pracujeme paralelně na dvou věcech (např. na textu a na obrázku k němu), které budou tvořit společný výsledek, je neustálé přenášení nových verzí únavné. Navíc se nám může stát, že nejnovější verzi přenést prostě zapomeneme a celý výsledek bude chybný.

Moderní operační systémy pro takový případ zavádějí tzv. **spojení dat**. Jeho mechanismus je podobný jako mechanismus inteligentní schránky; spojení mezi oběma programy však není přerušeno ani po přenosu dat. Zdrojový program pak po každé změně předaných dat o změně informuje cílový program, takže ten může zcela automaticky změnu reflektovat.

V praxi to funguje tak, že na začátku práce provedeme spojení dat, a pak již zcela libovolně pracujeme na obou částech projektu (nebo na všech, je-li jich více než dvě). Operační systém se přitom zcela transparentně postará o to, aby všechna spojená data byla na správném místě vždy v té nejnovější verzi.

12.3.4.4 Systém drag and drop

Systém drag and drop (asi nejvýstižnější překlad „táhni a pusť“ nezní příliš dobře, přidržíme se proto anglického termínu) operačního systému je vlastně

velmi příjemným grafickým uživatelským rozhraním nad (trochu rozšířeným) mechanismem inteligentní schránky.

Jedná se o to, že uživatel může pomocí myši „uchopit“ téměř libovolný objekt v kterémkoli okně a „přetáhnout“ jej jinam - třeba do okna úplně jiného programu. Tam pak objekt „pustí“. Operační systém se přitom postará o to, aby se data, která objekt reprezentoval, přenesla do patřičného programu a na správné místo.

Čteme-li tedy např. v systému elektronické pošty zprávu a líbí se nám v ní nějaký obrázek, nemusíme se starat o schránku. Obrázek prostě myší „přetáhneme“ do článku, který máme rozepsaný v textovém editoru, a je hotovo.

12.3.4.5 Datové služby

Jedná se o logické rozšíření mechanismu inteligentní schránky na poskytování libovolných služeb.

Při využívání datových služeb spolu kooperují dva programy. Jeden z nich nabízí data pomocí již známého mechanismu inteligentní schránky; tomuto programu budeme říkat „klient“. Druhý program - budeme mu říkat „datový server“ - nabízí služby. Každá služba může pracovat nad daty určitého typu (nebo určitých typů) a je identifikována jménem.

Jestliže si uživatel pracující s „klientem“, vyžádá datové služby, spojí si operační systém automaticky typy dat, které klient nabízí a datové služby, které nabízejí všechny „datové servery“. Uživateli pak nabídne seznam jmen všech služeb, které mohou zpracovávat některý z nabízených typů dat.

Jestliže uživatel zvolí některou ze služeb, aktivuje operační systém patřičný „datový server“ a pomocí mechanismu inteligentní schránky mu předá „klientova“ data, takže server je může v rámci služby téměř libovolným způsobem zpracovat.

Libovolný program se může snadno stát „datovým serverem“. Existuje proto obrovské množství datových služeb - od vyhledání slova ve výkladovém slovníku nebo ve slovníku synonym přes zařazení textu do osobního zápisníku až po automatický překlad obrázku (např. přijatého faxu) na text „datovým serverem“, který má schopnost OCR.

Kontrolní otázky:

1. Vyjmenujte alespoň pět služeb, které musí mít operační systém s grafickým interpretem příkazů.
2. Čím se liší implementace služby pomocí serveru a sdílenou knihovnou?
3. Proč operační systém musí podporovat národní prostředí?
4. Jakými způsoby lze v operačním systému zabezpečit komunikaci mezi programy?

Úkoly k zamyšlení:

1. Zamyslete se nad příkazy textového interpretu (např. MS DOS) a zdůvodněte si, proč jsou vždy v anglickém jazyce a nejsou překládány do národního prostředí.

Korespondenční úkol:

1. Představte si rozhraní člověka s počítačem řízeným hlasem. Napište některé příkazy, které by musel operační systém interpretovat, aby mohl vykonávat nejzákladnější funkce.

Shrnutí obsahu kapitoly

V této kapitole jste se seznámili s důvody obecných znalostí principů operačních systémů. Důraz v této kapitole byl kladen na pochopení rozhraní člověk/stroj a proces/operační systém. Velká pozornost byla věnována vysvětlení zájmů operačních systémů.

13 Rejstřík pojmů

Adresa:
Představuje číslo nebo symbol, který odkazuje na určité paměťové místo (paměťovou buňku) nebo na jiný zdroj dat, případně některé zařízení počítače.

Adresový prostor:
Oblast paměti pro ukládání programu a dat procesu.

Akcelerátor:
Moderní grafické adaptéry osobních počítačů se pro náročné grafické aplikace, kde složité vykreslované objekty podstatně zpomalují práci, vybavují tzv. akcelerátory, speciálními elektronickými obvody, které přebírají některé často se opakující vykreslovací funkce. Dociluje se tak zrychlení - akcelerace zobrazování.

Algebra BOOLEOVA:
Dvojhodnotová logická algebra, která používá základní logické operace – logický součin, logický součet a negaci.

Algoritmus:
Přesný a úplný návod k řešení nějaké úlohy s omezeným počtem kroků.

API:
Application Program Interface. Sada podprogramů, protokolů a nástrojů pro vytváření programových aplikací. U operačních systémů, jako je např. Windows, se API poskytuje programátorům, aby mohli vytvořit aplikace shodné s prostředím operačního systému. Ačkoliv API slouží programátorům, mají z něj největší výhody uživatelé, protože API zajišťuje stejné uživatelské rozhraní (interface) všech aplikací.

Architektura počítače:
Způsob vzájemného uspořádání a propojení funkčních jednotek v počítači.

ASP,PHP:
Activ Server Page, Hypertext Preprocesor. Skripty, které se vkládají do vzorů webových stránek na serveru. Podle požadavků se potom dynamicky generují celé stránky. Často se používají ve spojení s databázemi.

Assembler:
Jazyk symbolických instrukcí, programovací jazyk prakticky nejnižší úrovně, představující mnemotechnický popis strojových instrukcí procesoru.

ATM:
Asynchronous Transfer Mode - asynchronní přenosový režim

BIOS (ROM BIOS) (Basic Input Output System):
Programové vybavení uložené v paměti ROM (EPROM, EEPROM, Flash) zajišťující nejzákladnější funkce (např. zavedení OS).

bit:
1 bit (binary digit - dvojková číslice) je základní jednotka informace. Poskytuje množství informace potřebné k rozhodnutí mezi dvěma

možnostmi. Jednotka bit se označuje **b** a může nabývat pouze dvou hodnot - 0, 1.

BMP:

Bit Mapped Picture - bitově mapovaný, rastrovaný obraz. Obraz je v paměti interpretován jako body s definovanou barvou. 1. grafický standard pro operační systémy Windows, OS/2. 2. rozšíření názvu souboru (extension) obsahujícího bitově orientovanou grafiku. Opakem bitově mapovaného obrazu je obraz vektorový.

Bridge:

Zařízení určené k propojení dvou samostatných sítí. Most filtruje lokální komunikaci a propouští pouze data směřující do vzdálené části.

Browser:

Prohledávací (procházecí) program k prohlížení souborů. Prohlížeč, internetový prohlížeč. Program, který pohodlně zprostředkovává přístup k datům na Internetu, zvláště pak na World Wide Web.

Buňka paměťová:

Elementární jednotka paměti (paměťové místo pro jedno slovo), která je označena adresou, jejíž prostřednictvím je možno danou paměťovou buňku vyhledat a zpracovat.

Boot:

Zavádění systému, zaváděcí proces. Zavádění systému je proces, který probíhá po zapnutí počítače a teprve po jeho zdárném ukončení je možno počítač používat. Termín boot a bootstrap znamená zavedení a spuštění diskového operačního systému. Tento termín vznikl z anglické fráze ”pull yourself up by your own bootstraps” tedy „vytáhnout se sám za jazyk vlastních bot“.

Byte:

Jednotka informace, která se označuje **B** a platí 1 B = 8 b.

Cache:

Rychlá vyrovnávací paměť mezi procesorem a pomalejším zařízením (hlavní pamětí, pevným diskem, disketovou mechanikou apod.). Tato paměť umožňuje zrychlení práce procesoru. V použití paměti cache (český slang. používá ”kešovat”).

Client/server:

Filozofie uspořádání místní počítačové sítě. Počítače, propojené v síti jsou rozděleny na datové stanice (server) a pracovní stanice (client, workstation). Počítače tak mají vyhrazeny určité funkce, které vytvářejí jakousi hierarchickou strukturu sítě. Opak filozofie sítě peer to peer, kdy počítače propojené do sítě jsou si v jistém smyslu rovny, resp. mohou navzájem vystupovat současně jako datová stanice i klient.

Cyklus paměti:

Je to nejmenší možný časový interval mezi dvěma po sobě jdoucími příkazy k činnosti paměti.

Cyklus instrukční:

Cyklus opakování výběrové a prováděcí fáze řadiče.

Čítač instrukcí:

Registr řadiče, z jehož obsahu se odvozuje adresa příští instrukce.

Čtení paměti:

Přesunutí obsahu adresované paměťové buňky do pracovního registru počítače.

Data (údaje):

Informace, které se sestávají ze sledu znaků a vyjadřující údaje, hodnoty, čísla, znaky, symboly, grafy, ...

Defragmentace:

Proces systematického přeuspořádání dat zapsaných v paměťovém médiu. Může spojit prázdné místo, tedy vytvořit co nejlepší podmínky pro zápis nových dat. Může také spojit data, tedy seskupit již zapsaná data za sebe, čímž se zrychlí jejich čtení, a konečně může provést tzv. úplnou defragmentaci, tedy spojit a uspořádat již zapsaná data a umístit je tak těsně vedle sebe, že vznikne spojité prázdné místo pro zápis dalších nových dat. Programy pro defragmentaci jsou součástí většiny operačních systémů. Obdobně je defragmentace potřebná u některých metod přidělování operační paměti – např. přidělování bloků proměnné velikosti.

Diagram vývojový:

Blokové vyjádření určitého procesu zachycující v hrubých rysech jeho strukturu a návaznost činností.

DMA (Direct Memory Access):

Přímý přístup do operační paměti. DMA kanály slouží pro kopírování bloků dat mezi pamětí a rychlým I/O zařízením

Download:

Naplnění paměti v nějakém externím periferním zařízení daty. Příkladem může být download - zavedení znakové sady do paměti tiskárny.

Editor:

Program určený pro editaci - úpravy datových, v užším pojetí hlavně textových souborů. K základním funkcím textového editoru patří zápis textu, mazání, přepisování, vepisování, kopírování a přesouvání textových bloků, dále automatické vyhledávání textových řetězců, případně jejich záměna atd.

Firmware:

Programové vybavení, které tvoří součást technického vybavení. Toto programové vybavení až na naprosté výjimky nemůže být uživatelem modifikováno.

Font:

Znaková sada (v obecnějším významu grafická) jedné velikosti a typu písma. Fonty lze dělit především na bitově mapované a vektorové, na obrazovkové a tiskové, dále je možno fonty rozlišovat podle programového vybavení, pro něž jsou určeny, podle typu tiskárny atd.

GDI:

Graphics Device Interface - grafické rozhraní, které používají programy v prostředí Windows k zobrazování na monitoru, případně na jiných s grafikou pracujících zařízeních.

Hardware:

Technické vybavení počítače - souhrnný název pro veškerá fyzická zařízení, kterými je počítač vybaven.

Hub:

Rozbočovač. Zařízení používané u počítačových sítí k rozbočení (vytvoření více přípojných míst).

HTTP:

Hypertext Transfer Protocol. Základní protokol celého moderního webu. Pomocí něho se přenáší webové stránky do prohlížeče. Zabezpečenou variantou je HTTPS (HTTP over SSL – Secure Societ Layer).

Informace:

- data, která se strojově zpracovávají,
- vše co nám nebo něčemu podává (popř. předává) zprávu o věcech nebo událostech, které se staly nebo které nastanou.

Instrukce:

Předpis k provedení nějaké (většinou jednoduché) činnosti realizovatelný přímo technickým vybavením počítače (např. přičtení jedničky, uložení hodnoty do paměti apod.)

Integrovaný obvod:

Elektronická součástka realizující určité množství obvodových prvků neoddělitelně spojených na povrchu nebo uvnitř určitého spojitého tělesa, aby se dosáhlo ucelené funkce elektronického obvodu.

LAN:

Local Area Network - Místní počítačová síť. Místně omezená síť pro přenos dat. Omezení se zpravidla vztahuje na určitý základ - budovu nebo komplex budov. Místní sítě LAN pracují s přenosovými rychlostmi řádově desítky a stovky megabitů za sekundu.

MIPS:

Megainstruction per sekond, Million Instruction Per Sekond, oba výklady zkratky vyjadřují totéž - jednotku výkonnosti výpočetního systému v provedených milionech instrukcí za sekundu.

Modem:

Umělé slovo, akronym vzniklý ze slov MOdulátor a DEModulátor. Modem je zařízení pro přenos dat, zajišťující přenos po analogových vedeních na principu modulace a následné demodulace analogového signálu číslicovými daty. U osobních počítačů se modemu používá pro přenos dat mezi počítači po telefonních vedeních.

MPEG:

Moving Pictures Experts Group. Označení pro formát vytvořený pracovní skupinou MPEG. Principem tohoto formátu je ukládání změn, ke kterým došlo v následujícím snímku obrazové sekvence. Tato technika, označovaná DTC je typu ztrátové komprese, protože některá data jsou odstraněna, ztracena. To ovšem nevede k žádnému pro lidské oko patrnému zhoršení vjemu.

Multiprogramový systém:

Systém v němž může být více procesů najednou ve stavu provádění. Proces je ve stavu provádění, jestliže byl zahájen a nebyl ještě dokončen nebo zastaven *(popř. ukončen chybou)*. Proces může být ve stavu provádění, ale ve skutečnosti nemusí být právě prováděn, tj. některé mezivýsledky jsou spočítány, ale procesor provádí v daném okamžiku některý jiný proces. Současný běh více procesů je zdánlivý, protože v daném okamžiku může procesor provádět vždy jen jeden z nich.

Multitasking:

Současný provoz více úloh na jednom počítači, kdy jedna úloha probíhá na popředí a ostatní probíhají na pozadí. Dovoluje lepší využití CPU. V případě, že uživatel pracuje interaktivně s nějakým programem, který většinu času čeká na zadání jeho požadavků, je možné, aby procesor prováděl např. nějaký náročný matematický výpočet. Je-li na počítači s jedním procesorem provozováno více programů, je procesor přidělován postupně vždy na určitou dobu, tzv. **časové kvantum** (asi 0.1 s), všem provozovaným programům. Podle způsobu práce rozlišujeme dva druhy multitaskingu:

- **kooperativní (nepreemptivní) multitasking**: procesor je operačním systémem přidělen jednomu programu, který jej má v držení tak dlouho, dokud jej sám nevrátí zpět operačnímu systému. Ten jej pak přidělí jinému programu. Nevýhodou je, že program nemusí procesor navrátit v dostatečně krátkém časovém úseku, což způsobí dojem, že ostatní programy nepracují. Ještě horší případ nastane ve chvíli, kdy program procesor nevrátí vůbec (např. zhavaruje). Tato situace vede ve většině případů k havárii celého systému.
- **preemtivní multitasking**: procesor je přidělen programu pouze na určitou dobu a po jejím uplynutí jej sám operační systém programu odebere a přidělí jinému programu. Z toho vyplývá, že nemohou nastat stavy uvedené u kooperativního multitaskingu. Nevýhodou tohoto řešení je vyšší náročnost na hardwarové vybavení počítače.

OLE:

Object Linking and Embedding spojování a vkládání objektů. Technika přenosu dat mezi aplikacemi v prostředí Windows. Spojení mezi objekty v aplikacích zaručuje ”živý” přenos změn provedených v jedné aplikaci (např. v tabulkovém kalkulátoru) do jiné aplikace (např. textového editoru s vloženou tabulkou).

Operační systém:

Ovládá základní technické prostředky počítače a vytváří vhodnější podmínky pro jejich využívání v uživatelských programech. Funkce operačního systému tvoří podstatnou složku činnosti počítače a mnozí uživatelé je ani nerozlišují od funkcí technického vybavení. Operační systém jsou ty programové moduly ve výpočetním systému, jež ovládají řízení prostředků, jimiž je tento výpočetní systém vybaven (procesory, operační paměť, vnější paměť, I/O zařízení, soubory dat). Tyto moduly „rozhodují spory“ *(např. o užití téhož prostředku více úlohami),* snaží se optimalizovat výkon a zjednodušují efektivní využívání výpočetního systému.

Paměť:

Zařízení, které slouží pro uchování informací (konkrétně binárně kódovaných dat). Množství informací, které je možné do paměti uložit, se nazývá **kapacita paměti** a udává se v bytech. Protože byte je poměrně malá jednotka, používá se často následujících předpon:

Předpona	Značka	Zápis	Mocnina (B)	Převod (B)

kilo	k, K	1 kB	2^{10} B	1024 B
mega	M	1 MB	2^{20} B	1048576 B
giga	G	1 GB	2^{30} B	1073741824 B
tera	T	1 TB	2^{40} B	1099511627776 B

Paměť bývá rozdělena na buňky určité velikosti, z nichž každá je jednoznačně identifikována svým číslem. Toto číslo se nazývá **adresa paměti** a velikost takovéto buňky, která má svou vlastní adresu, se označuje jako **nejmenší adresovatelná jednotka**. Paměti je možné rozdělit do následujících základních skupin:

- **Vnitřní (operační)**: paměť sloužící pro uchování momentálně zpracovávaných dat a programů. Realizovaná většinou pomocí polovodičových součástek.
- **Vnější (periferní)**: paměť sloužící k dlouhodobějšímu uchování dat. Realizovaná většinou na principu magnetického (popř. optického) záznamu dat. Ve srovnání s operační pamětí bývá přístup k jejím datům pomalejší.
- **RAM**: paměť určená ke čtení i zápisu dat.
- **ROM**: paměť určená pouze ke čtení dat.
- **Paměť s přímým přístupem**: paměť, která dovoluje přistoupit okamžitě k místu s libovolnou adresou.
- **Paměť se sekvenčním přístupem**: paměť, u které je nutné při přístupu k místu s adresou n nejdříve postupně přečíst všechna předcházející místa (0 až n-1).

Peer to peer:

Rovnocenná síťová komunikace (rovný s rovným) na stejné úrovni, tzn. oba či více účastníků mohou být vzájemně datovými stanicemi i klienty zároveň.

Pevný disk:

Diskové záznamové médium, označované dříve též někdy winchester nebo i do češtiny přejatým anglickým výrazem ”harddisk” nebo také jen ”hard”. Pevné disky mají relativně vysokou kapacitu (desítky až stovky GB). Konstrukčně představuje pevný disk vlastně svazek tuhých (zpravidla kovových) magnetických disků, rotujících běžně rychlostí 3600 - 10000 ot/min, uzavřený v utěsněném pouzdře, které má bránit přístupu prachu a nečistot. Nad jednotlivými disky svazku se rychle pohybují v nepatrné vzdálenosti od povrchu hlavičky pro čtení a zápis.

Počítač:

Stroj na zpracování informací.

POP3:

Post Office Protocol. Označení serveru (případně protokolu) pro stahování elektronické pošty ze serveru do klientského počítače.

Proces:

Instance úlohy, kterou vytváří procesor a která může být prováděna paralelně s jinými výpočty.

Program:

Algoritmus zapsaný v programovacím jazyce, který řeší nějaký konkrétní úkol. Jedná se o posloupnost instrukcí.

Přerušení (interrupt):

Proces, během kterého je procesor nucen zaznamenat nějakou událost.

Registr:

Velmi rychlé paměťové místo kapacity jednoho byte nebo jednoho slova.

Repeater:

Opakovač, zařízení, které umožňuje zvětšení dosahu sítě. Např. u místní počítačové sítě se opakovač použije při propojování dvou budov, které jsou vzájemně vzdáleny více, než by dovolovala běžná kabeláž.

Řadič (Controller):

Zařízení převádějící příkazy v symbolické formě (instrukce) na posloupnost signálů ovládajících připojené zařízení. Jedná se tedy o zařízení, které řídí činnost jiného zařízení.

Software:

Programové vybavení počítače - souhrnný název pro veškeré programy, které mohou na počítači pracovat. Software je možné rozdělit do dvou skupin:

- **Systémový software**: operační systémy, pomocné programy pro správu systému (utility), překladače programovacích jazyků.
- **Aplikační software**: programy umožňující řešení specifických problémů uživatele

Virtuální paměť:

Virtual memory. Simulovaná operační paměť např. na pevném disku, doplňující většinou skutečnou operační paměť. Používá se typicky tam, kde pro nějakou operaci s velkým objemem dat nedostačuje kapacita skutečné operační paměti.

Vstupní / výstupní (V/V) zařízení (I/O devices - Input / Output):

Zařízení určená pro vstup i výstup dat. Např.: disky (pevné, pružné), tiskárny, klávesnice atd.

WAN:

Wide Area Network - dálková počítačová síť. Síť typu WAN není omezena plošně a zpravidla navzájem propojuje jednotlivé místní sítě (LAN) i individuální účastníky na vzdálenost desítek až tisíců kilometrů.

Word:

Jednotka informace. Platí 1 W = 2 B = 16 b. Kromě této jednotky se také někdy užívá ještě 1 doubleword (DW), pro který platí 1 DW = 2 W = 4 B = 32 b.

14 Literatura

[1] Bach, M. J.: Principy operačního systému UNIX, Softwarové Aplikace a Systémy 1993, ISBN 80-901507-0-5
[2] Blatný, J. a kol.: Číslicové počítače I – II. Skripta VUT Brno. SNTL Praha 1983
[3] Čada, O.: Operační systémy. Grada 1993. ISBN 80-85623-44-7
[4] Hansen, P. B.: Principy operačních systémů. SNTL Praha 1979
[5] Hořejš, J., Brodský, J., Staudek, J.: Struktura počítačů a jejich programového vybavení. SNTL Praha 1981
[6] Klimeš, C. a kol.: Prostředky automatického řízení IIA. Počítačové systémy. Skripta VŠB Ostrava 1983
[7] Klimeš, C. a kol.: Prostředky automatického řízení IIB. Mikropočítačové systémy. Skripta VŠB Ostrava 1985
[8] Klimeš, C. – Burianová, E.: Základní pojmy z operačních systémů. Ostravská univerzita v Ostravě. 2003. ISBN 80-7042-862-7
[9] Klimeš, C.: Operační systémy 1a. Ostravská univerzita v Ostravě. 2003. ISBN 80-7042-850-3
[10] Klimeš, C. – Sochor. T.: Počítačové sítě 1. Ostravská univerzita v Ostravě. 2003. ISBN 80-7042-853-8
[11] Klimeš, C.: Operační systémy 1b. Ostravská univerzita v Ostravě. 2004. ISBN 80-7042-951-8
[12] Klimeš, C.: Logické obvody a systémy elektronických počítačů. Ostravská univerzita v Ostravě. 2004. ISBN 80-7042-952-6
[13] Klimeš, C. – Balogh, Z.: Principy operačných systémov. Edícia prírodovedec č. 186, FPV Nitra 2005. Vydavateľstvo Michala Vašku, Prešov, ISBN 80-8050-894-1
[14] Klimeš, C. – Turčáni, M.: Úvod do operačných systémov. Edícia prírodovedec č. 187, FPV Nitra 2005. Vydavateľstvo Michala Vašku, Prešov, ISBN 80-8050-895-X
[15] Král, J., Demner J.: Softwarové inženýrství, ACADEMIA Praha 1991
[16] Kurfirst M.: Historie operačních systémů Windows, Unix, Mac OS a Linux. http://www.mujmac.cz/art/polemiky/historie-operacnich-systemu-win-unix-macosx.html.
[17] Madnick, S.E., Donovan, J.J.: Operační systémy. SNTL Praha 1981
[18] Plášil, F.: Operační systémy. ČVUT Praha 1991. ISBN 80-01-00178-4
[19] Petzold, Ch.: Programování ve Windows. Computer Press Praha 1999. ISBN 80-7226-206-8
[20] Skočovský, L.: UNIX POSIX PLAN 9. Duo Press, Mnichovo Hradiště, 1998. ISBN 80-902612-0-5
[21] Staudek, J.: Operační systémy. Elektronická podoba přednášek. FI MU Brno, 2002
[22] Tanenbaum, A. S.: Operating Systems. Design and Implementation. Prentice Hall. 1997, ISBN 0-13-638677-6
[23] Tanenbaum, A. S.: Modern Operating Systems. Prentice Hall. 1992, ISBN 0-13-638677-6