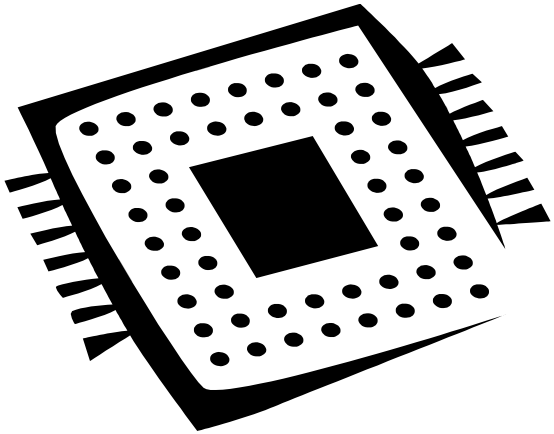
PB 169
Počítačové sítě a operační systémy

Plánování CPU

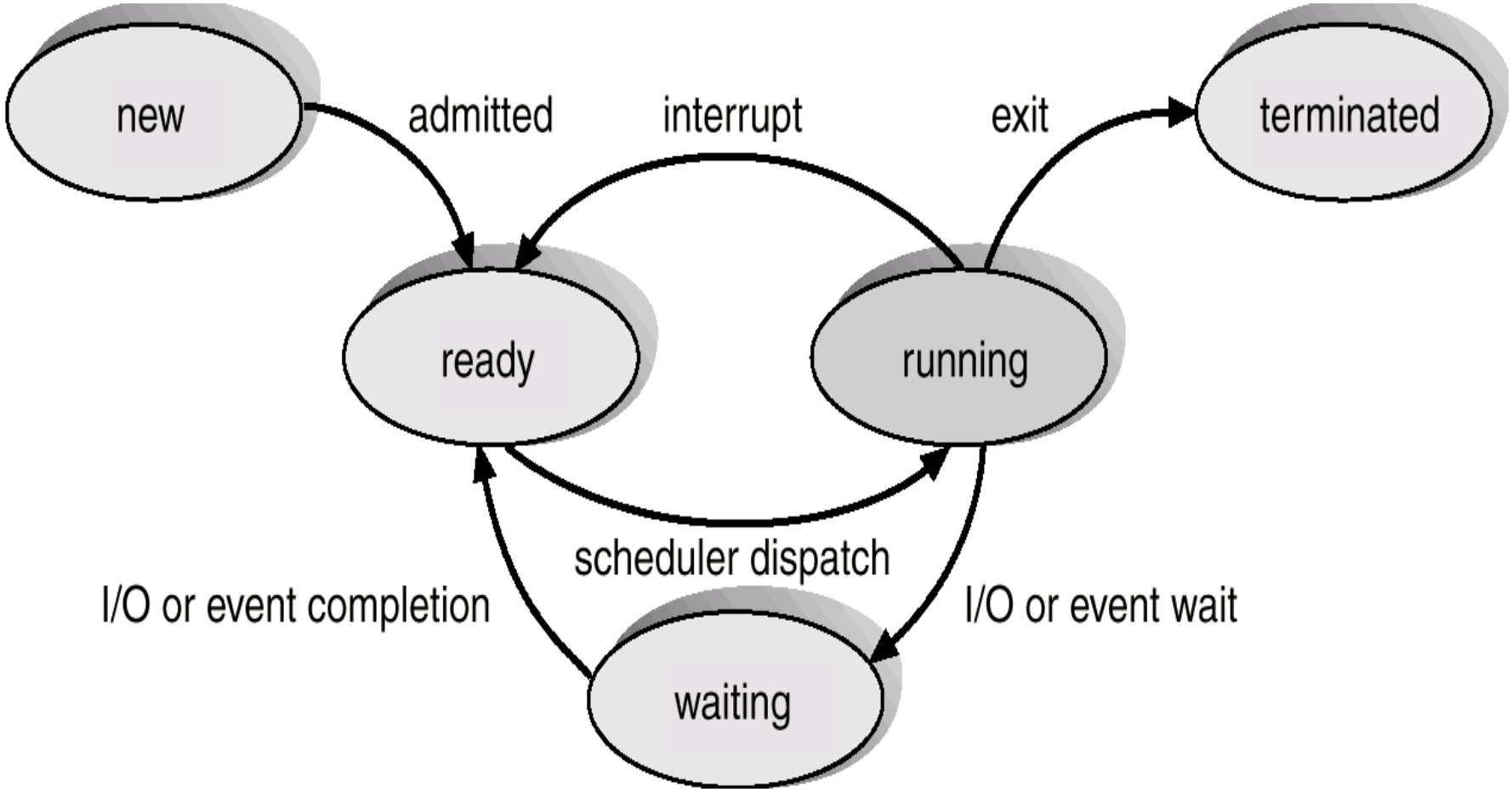
Správa paměti



Multiprogramování

- Multiprogramování zvyšuje využití CPU
- Pokud jeden proces čeká na dokončení I/O operace může jiný proces CPU využít
- Nejlepšího výsledku dosáhneme při vhodné kombinaci procesů orientovaných na I/O a na využití CPU

Stavy procesu



Plánování CPU

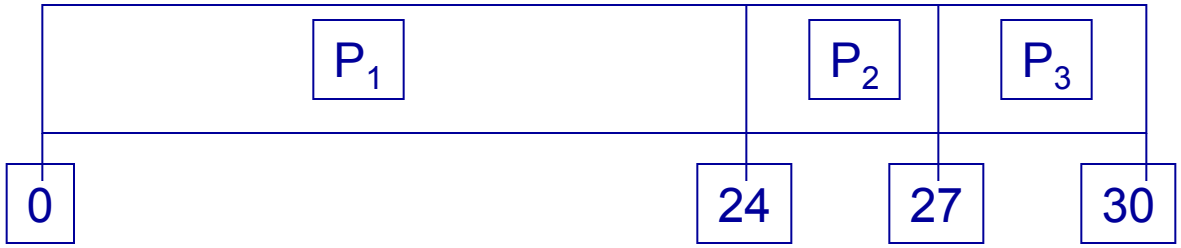
- Krátkodobý plánovač – dispečer
  - Vybírá proces, kterému bude přidělen CPU
  - Vybírá jeden z procesů, které jsou zavedeny operační paměti a které jsou „připravené“
  - Plánovací rozhodnutí může vydat v okamžiku, kdy proces:
    - 1. přechází ze stavu běžící do stavu čekající
    - 2. přechází ze 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)

Dispečer

- Výstupní modul krátkodobého plánovače nebo plánovač sám, který předává procesor procesu vybranému krátkodobým plánovačem
- Předání zahrnuje:
  - přepnutí kontextu
  - přepnutí režimu procesoru na uživatelský režim
  - skok na odpovídající místo v uživatelském programu pro opětovné pokračování v běhu procesu
- Dispečerské zpoždění (Dispatch latency)
  - Doba, kterou potřebuje dispečer pro pozastavení běhu jednoho procesu a start běhu jiného procesu

Algoritmus FCFS

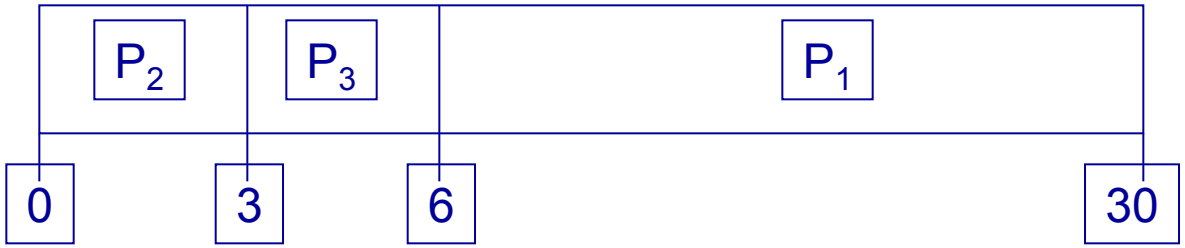
- Algoritmus „Kdo dřív přijde, ten dřív mele“ (First Come, First Served), FCFS
- Máme 3 procesy P1 (vyžaduje 24 dávek CPU), P2 (vyžaduje 3 dávky CPU), P3 (vyžaduje 3 dávky CPU)
- Procesy vznikly v pořadí: P1, P2, P3
- Ganttovo schématické vyjádření plánu:



- Doby čekání: P1 = 0, P2 = 24, P3 = 27
- Průměrná doba čekání: (0+24+27)/3 = 17

Algoritmus FCFS (2)

- Varianta jiná: procesy vznikly v pořadí P2, P3, P1
- Ganttovo schématické vyjádření plánu:



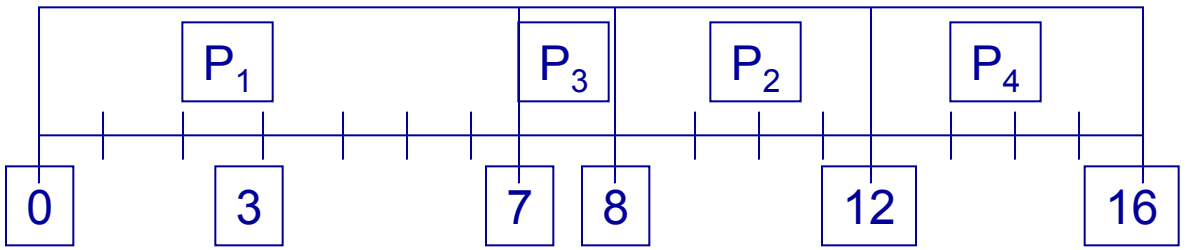
- Doby čekání: P2 = 0, P3 = 3, P1 = 6
- Průměrná doba čekání: (6+0+3)/3 = 3
- To je mnohem lepší výsledek než v předchozím případě, i když se jedná o stejné procesy a stejný plánovací algoritmus
- Krátké procesy následující po dlouhém procesu ovlivňuje „konvojový efekt“

Algoritmus SJF

- Algoritmus Shortest-Job-First
- Musíme znát délku příštího požadavku na dávku CPU pro každý proces
- Vybírá se proces s nejkratším požadavkem na CPU
- Dvě varianty:
  - nepreemptivní, bez předbíhání
    - jakmile se CPU předá vybranému procesu, tento nemůže být předběhnut žádným jiným procesem, dokud přidělenou dávku CPU nedokončí
  - preemptivní, s předbíháním
    - jakmile se ve frontě připravených procesů objeví proces s délkou dávky CPU kratší než je doba zbývající k dokončení dávky právě běžícího procesu, je právě běžící proces ve využívání CPU předběhnut novým procesem
    - tato varianta se rovněž nazývá Shortest-Remaining-Time-First (SRTF)
- SJF je optimální algoritmus (pro danou množinu procesů dává minimální **průměrnou** dobu čekání)

Příklad *nepreemptivního* algoritmu SJF

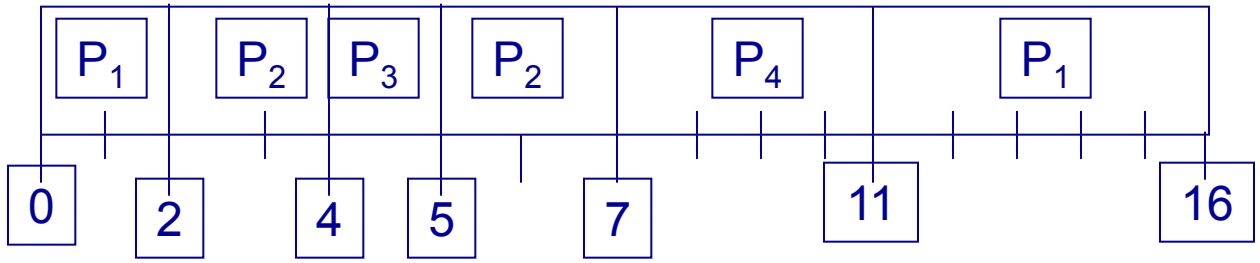
- Proces Doba příchodu Délka dávky CPU
  - P1 0.0 7
  - P2 2.0 4
  - P3 4.0 1
  - P4 5.0 4
- Ganttovo schématické vyjádření plánu:



- Průměrná doba čekání: (0+6+3+7)/4 = 4

Příklad *preemptivního* algoritmu SJF

- Proces Doba příchodu Délka dávky CPU
  - P1 0.0 7
  - P2 2.0 4
  - P3 4.0 1
  - P4 5.0 4
- Ganttovo schématické vyjádření plánu:



- Průměrná délka čekání: (9+1+0+2)/4 = 3

Prioritní plánování

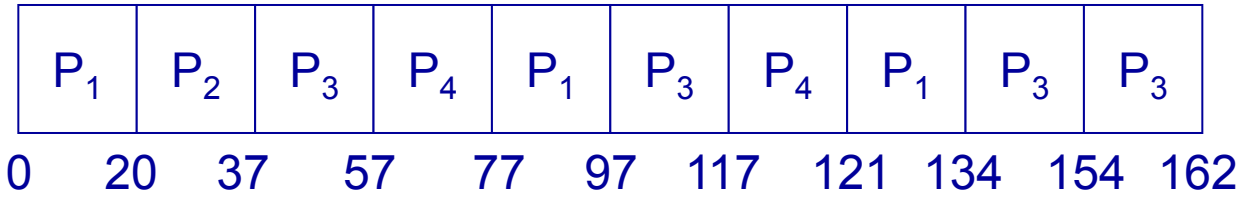
- S každým procesem je spojeno prioritní číslo
  - prioritní číslo – preference procesu pro výběr příště běžícího procesu
  - CPU se přiděluje procesu s největší prioritou
  - nejvyšší prioritě obvykle odpovídá nejnižší prioritní číslo ☺
- Opět dvě varianty
  - nepreemptivní, bez předbíhání
    - jakmile proces získá přístup k CPU nemůže být předběhnut jiným procesem dokud dávku neukončí
  - preemptivní, s předbíháním
    - jakmile se ve frontě připravených procesů objeví proces s vyšší prioritou než je priorita běžícího procesu, je běžící proces předběhnut
- SJF je prioritní plánování, prioritou je předpokládaná délka příští CPU dávky
- stárnutí
  - procesy s nižší prioritou se nemusí nikdy provést
  - řešení: zrání – priorita se s postupem času zvyšuje

Round Robin (RR)

- Každý proces dostává CPU na malou jednotku času – časové kvantum
  - 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 procesů n procesů a časové kvantum je *q*, pak každý proces získává *1/n* doby CPU, najednou nejvýše *q* časových jednotek
- Žádný proces nečeká na přidělení CPU déle než *(n-1)q* časových jednotek
- Výkonnostní hodnocení
  - q velké → ekvivalent FIFO
  - q malé → velká režie; v praxi musí být q musí být dostatečně velké s ohledem na režii přepínání kontextu

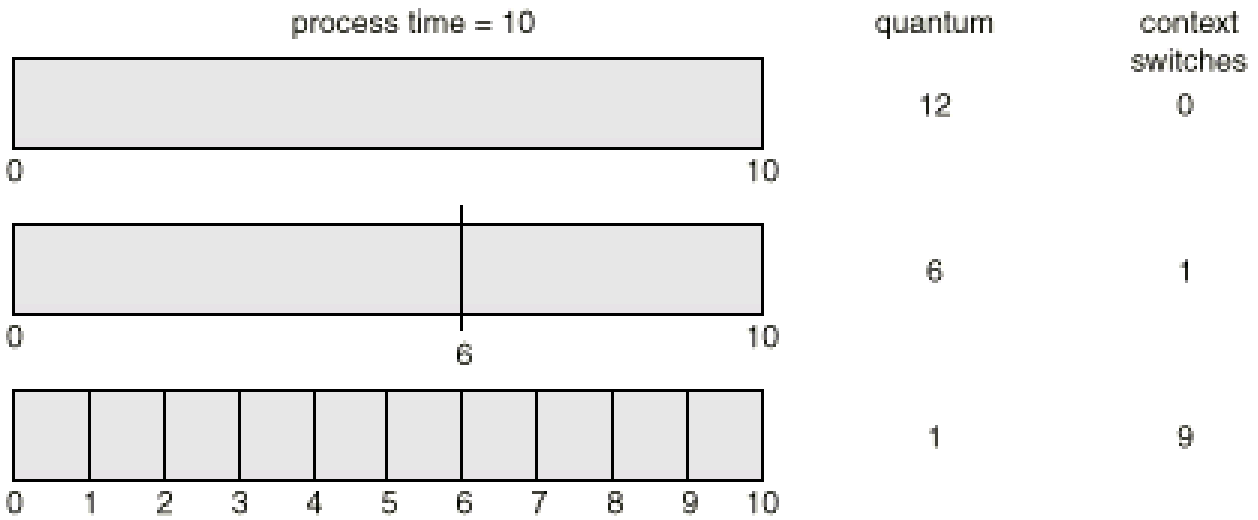
Příklad RR s časovým kvantem = 20

- Proces Délka dávky CPU
  - P1 53
  - P2 17
  - P3 68
  - P4 24
- Ganttovo schématické vyjádření plánu:



- Typicky se dosahuje delší *průměrné* doby obrátky než při plánování SJF, avšak doba odpovědi je výrazně nižší

Časové kvantum a doba přepnutí kontextu



- Příklad: doba přepnutí kontextu = 0,01
- Ztráty související s režií OS při q = 12, 6 a 1 jsou 0,08; 0,16 a 1 %

Příklad: Win32 (1)

- Z pohledu Win32:
  - procesy při vytvoření přiděleny do jedné z následujících tříd
    - Idle
    - Below Normal
    - Normal
    - Above Normal
    - High
    - Realtime
  - Vlákna dále mají relativní prioritu v rámci třídy, do které patří
    - Idle
    - Lowest
    - Below_Normal
    - Normal
    - Above_Normal
    - Highest
    - Time_Critical

Příklad: Win32 (2)

- Plánovací algoritmus je řízen především prioritami
  - 32 front (FIFO seznamů) vláken, která jsou „připravena“
    - pro každou úroveň priority jedna fronta
    - fronty jsou společné pro všechny procesory
  - když je vlákno „připraveno“
    - buď běží okamžitě
    - nebo je umístěno na konec fronty „připravených“ procesů ve své prioritě
  - na jednoprocesorovém stroji vždy běží vlákno s nejvyšší prioritou
- V rámci jedné prioritní skupiny se plánuje algoritmem round-robin pomocí časových kvant

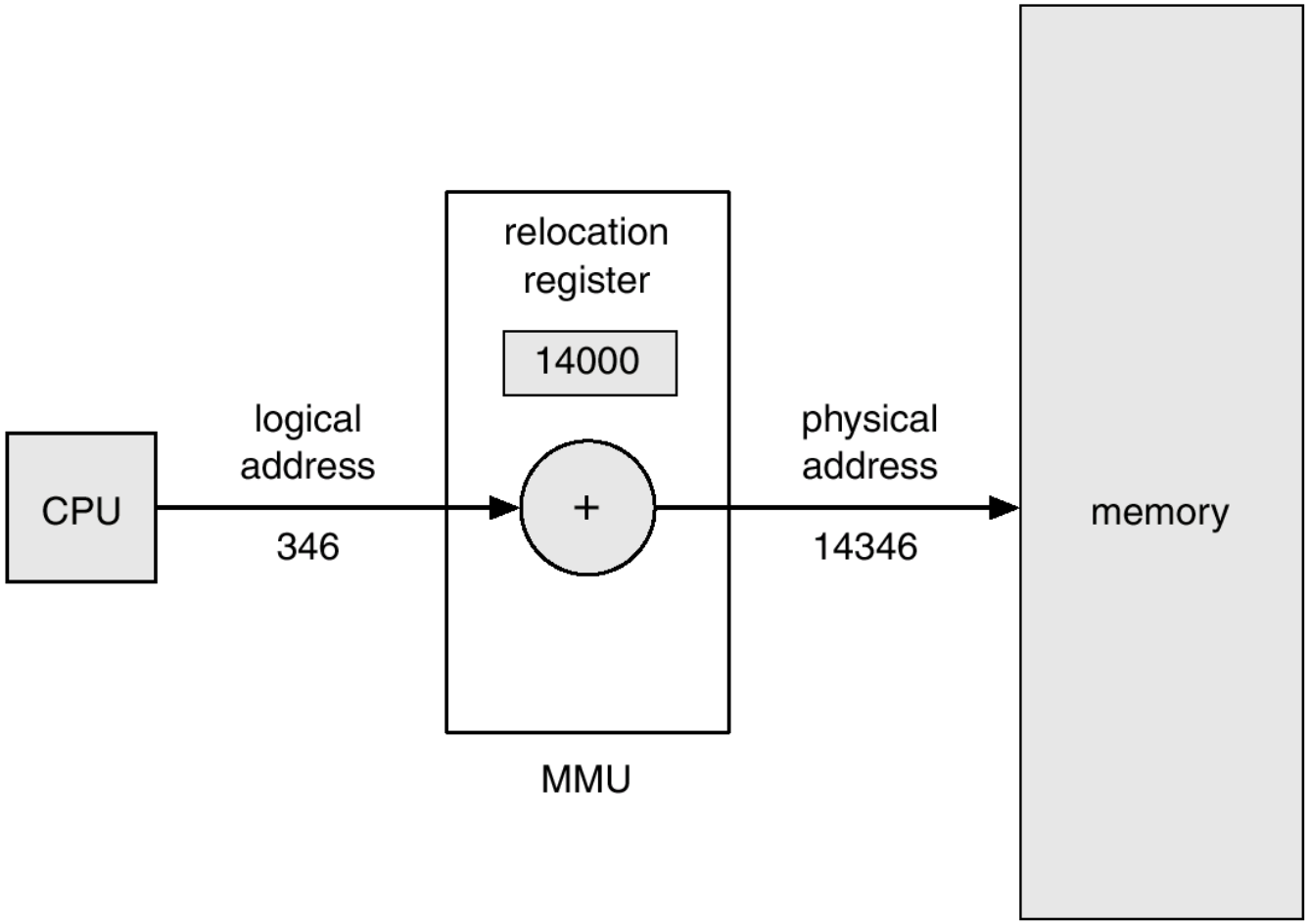
Paměť: principy, základy

- Pro běh procesu je nutné, aby program, který je vykonáván byl umístěn v operační paměti (hlavní paměti)
- Z programu se stává proces (aktivní entita schopná spuštění na CPU) provedením celé řady kroků
  - naplnění tabulek, umístění do operační paměti
  - vázání adres instrukcí a dat na adresy operační paměti

Memory-Management Unit

- Hardwarový modul převádějící logické adresy na fyzické adresy
- Uživatelský program pracuje s logickými adresami, uživatelský program nevidí fyzické adresy
- Připočítává se obsah „relokačního registru“ k adresám generovaným uživatelským procesem v okamžiku, kdy je předávána jako ukazatel do operační paměti

Relokační registr



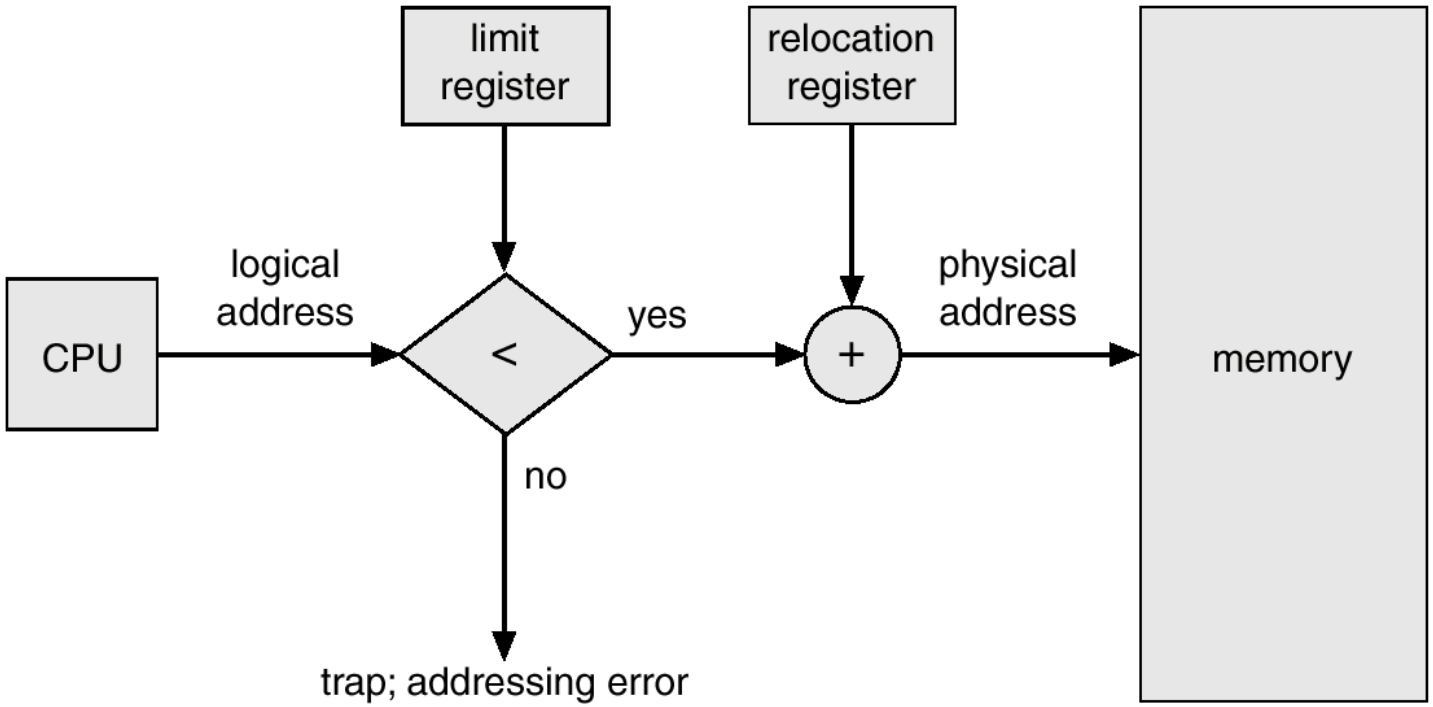
Adresový prostor

- Logický adresový prostor (LAP), fyzický adresový prostor (FAP)
  - LAP – (logická adresa, virtuální adresa) dána adresou ve strojovém jazyku, generuje CPU
  - FAP – (fyzická) adresa akceptovaná operační pamětí
- Logické a fyzické adresové prostory se shodují v době kompilace a v době zavádění
- Logické a fyzické adresové prostory mohou být rozdílné při vázání v době běhu

Souvislé oblasti

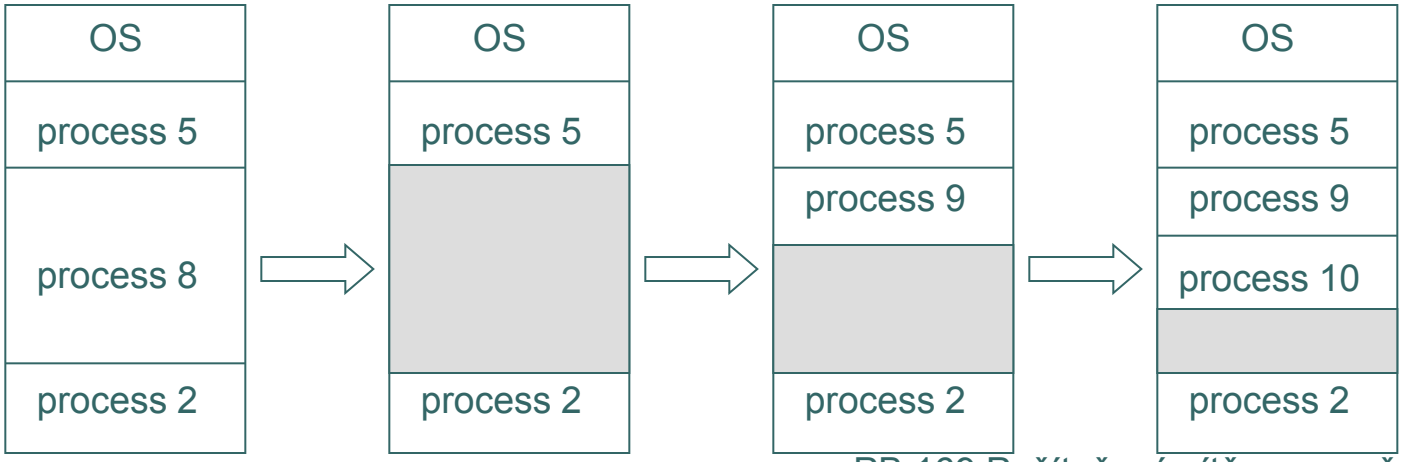
- Operační paměť se dělí do dvou sekcí
  - rezidentní OS, obvykle na počátku FAP s tabulkou ovladačů přerušení
  - uživatelské procesy
- Přidělování jedné souvislé části paměti
  - pro ochranu procesů uživatelů mezi sebou a OS lze použít schéma s relokačním registrem
  - relokační registr: hodnota nejmenší fyzické adresy paměti procesu
  - mezní registr: rozpětí logických adres, logická adresa musí být menší nebo rovna meznímu registru

HW podpora


limit
register
relocation
register
logical
address
CPU
<
yes
+
physical
address
memory
no
trap; addressing error

Souvislé oblasti

- Přidělování několika částí paměti
  - *díra* – blok dostupné paměti
    - Bloky jsou roztroušeny po FAP
  - evidenci o přidělených a volných sekcí udržuje OS



Přidělování paměti

- Kterou oblast délky n přidělit, když volná paměť je rozmístěna ve více souvislých nesousedních sekcích?
  - First-fit:
    - přiděluje se první dostatečně dlouhá volná oblast resp. její počátek
  - Best-fit:
    - přiděluje se nejmenší dostatečně dlouhá volná oblast resp. její počátek
    - generují se velmi malé (nejmenší) možné volné díry
  - Worst-fit:
    - přiděluje se největší dostatečně dlouhá volná oblast resp. její počátek
    - generují se největší možné volné díry
- Z hlediska rychlosti a kvality využití paměti jsou First-fit a Best-fit lepší techniky než technika Worst-fit

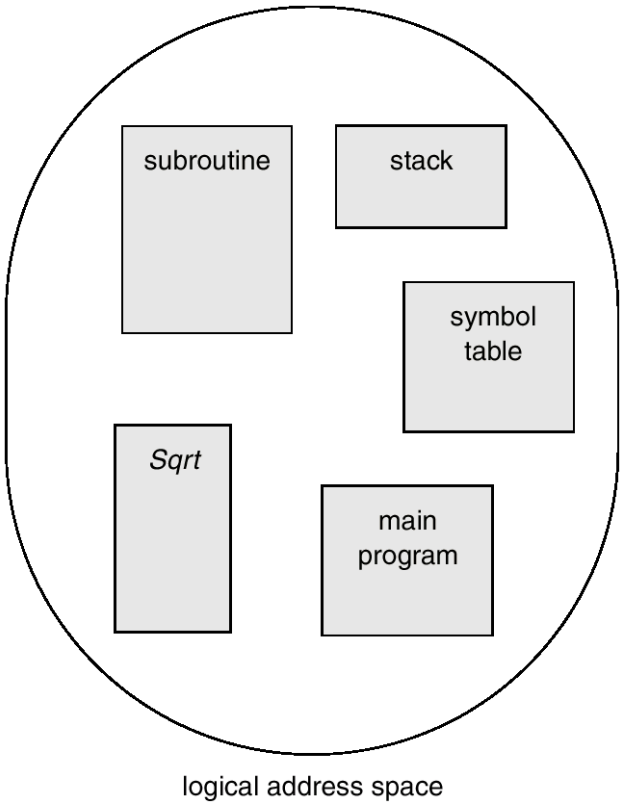
Problém fragmentace

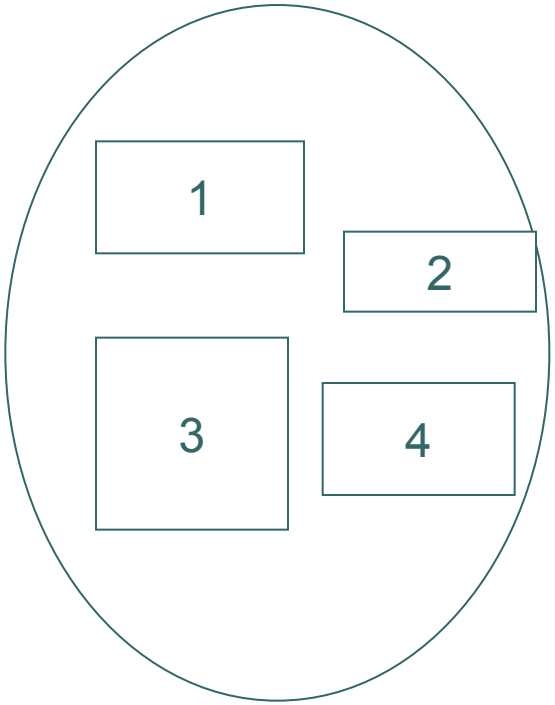
- Vnější fragmentace
  - souhrn volné paměti je dostatečný, ale ne v dostatečné souvislé oblasti
- vnitřní fragmentace
  - přidělená oblast paměti je větší než požadovaná velikost, tj. část přidělené paměti je nevyužitá
- Snižování vnější fragmentace setřásáním
  - přesouvají se obsahy paměti s cílem vytvořit (jeden) velký volný blok
  - použitelné jen když je možná dynamická relokace (viz MMU)
  - provádí se v době běhu
  - problém I/O
    - s vyrovnávacími paměťmi plněnými z periférií autonomně nelze hýbat – umisťují se proto do prostoru OS

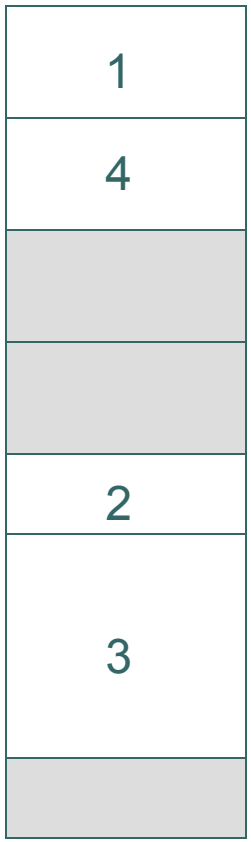
Stránkování

- LAP procesu nemusí být jedinou souvislou sekcí FAP, LAP se zobrazuje do (po částech volných) sekcí FAP
- FAP se dělí na sekce zvané rámce (frames)
  - pevná délka, délka v násobcích mocnin 2 (obvykle mezi 512 až 8192 bajty)
- LAP se dělí na sekce zvané stránky (pages)
  - pevná délka, shodná s délkou rámců
- Udržujeme seznam volných rámců
- Program délky *n* stránek se umístí (zavede) do **n** rámců
- Překlad logická adresa → fyzická adresa − pomocí překladové tabulky nastavované OS a interpretované MMU
- Vniká vnitřní fragmentace, neboť paměť je procesu přidělována v násobcích velikosti rámce

Segmentování



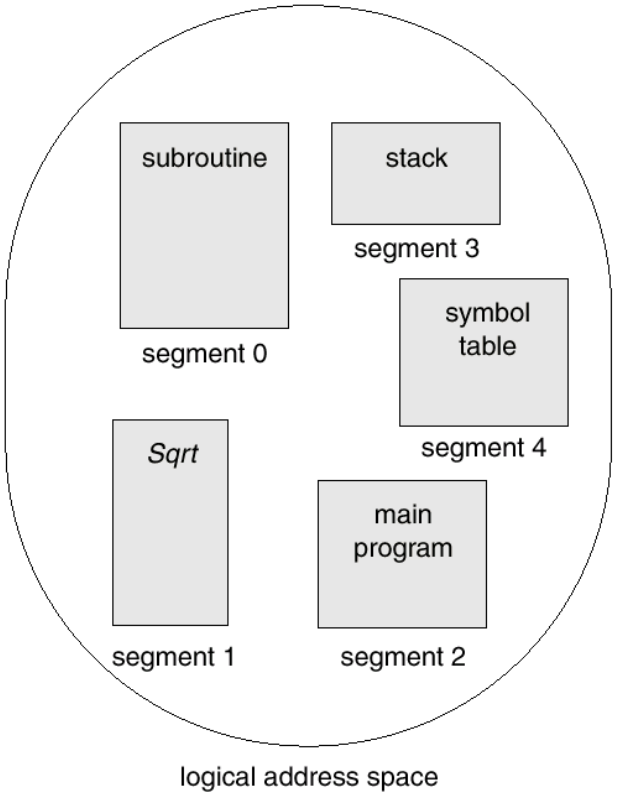




Segmentování

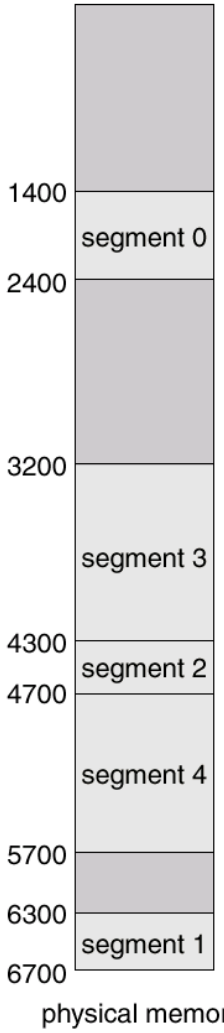
- Logická adresa je dvojice (segment s, offset d)
- *Tabulka segmentů, Segment table*, ST
  - *base* – počáteční adresa umístění segmentu ve FAP
  - *limit* – délka segmentu
- *Segment-table base register (STBR)*
  - odkaz na umístění ST v paměti
- *Segment-table length register (STLR)*
  - počet segmentů, s je legální když *s* < STLR
- Relokace – dynamická, pomocí ST

Příklad segmentace



	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table



Stránkování a segmentování (Intel 386)