INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA STROJNÍ

OPERAČNÍ SYSTÉMY

PŘIDĚLOVÁNÍ PROCESORU

doc. Dr. Ing. Oldřich Kodym

Ostrava 2013

© doc. Dr. Ing. Oldřich Kodym

© Vysoká škola báňská – Technická univerzita Ostrava

ISBN 978-80-248-3053-7

Tento studijní materiál vznikl za finanční podpory Evropského sociálního fondu (ESF) a rozpočtu České republiky v rámci řešení projektu: CZ.1.07/2.2.00/15.0463, MODERNIZACE VÝUKOVÝCH MATERIÁLŮ A DIDAKTICKÝCH METOD

OBSAH

4. PŘIDĚLOVÁNÍ PROCESORU

OBSAH KAPITOLY:

Základní pojmy, principy.

Kritéria přidělování procesoru.

Algoritmy plánování v jednoprocesorových systémech.

Algoritmy plánování ve víceprocesorových systémech.

Plánování v reálném case.

MOTIVACE:

Chod operačního systému využívá mnoha standardních mechanizmů známých z jiných oblastí řízení i běžného života. Přidělování procesoru ve víceúlohovém systému je založeno na obecné teorii front a hromadné obsluhy. Plánování je základem fungování moderního OS. V takovém systému jsou plánovány veškeré jeho zdroje a CPU je pochopitelně zdroj nejdůležitější. Proto je plánování CPU ústředním bodem návrhu OS. Různé přidělovací algoritmy mají různé vlastnosti a mohou upřednostňovat jednu třídu procesů před jinými. Vybíráme-li, který algoritmus užít v určité konkrétní situaci, musíme uvažovat právě ony vlastnosti různých algoritmů.

CÍL:

Přidělování procesoru – základní principy, CPU cyklus, I/O cyklus, preemptivní, nepreemptivní plánování.

Kritéria přidělování CPU, plánování FCFS, SJF.

Plánování podle priority, cyklická obsluha.

Plánování pomocí více frontStavy procesu, Process Control Block, přepínání kontextu.

Plánování procesů, plánovací fronty.

Operace s procesy – vytvoření, zrušení.

Komunikace procesů – základní principy, přímá a nepřímá komunikace, buffery.

1. ÚVOD

Přidělování CPU je základem multiprogramového OS. Pomocí přidělování CPU různým procesům OS zvyšuje výkon výpočetního systému.

Popíšeme základní principy přidělování CPU a ukážeme různé algoritmy přidělování.

2. ZÁKLADNÍ PRINCIPY

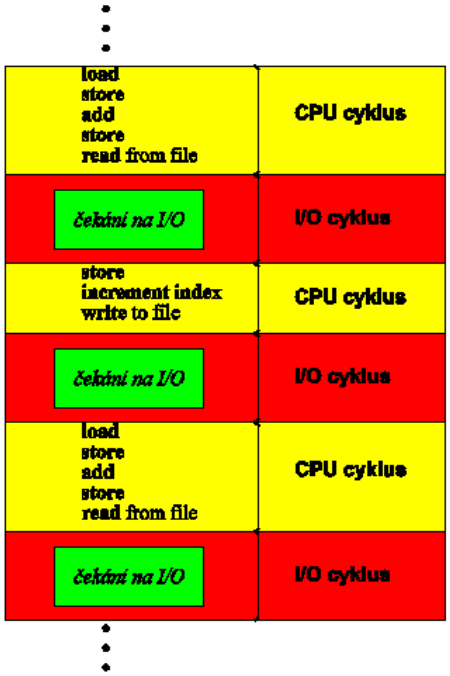
Základní myšlenka multiprogramování je relativně jednoduchá. V jednoduchém OS, pokud jediná spuštěná úloha musí čekat (nejčastěji na I/O operaci), je blokováno CPU. V multiprogramovém OS je možno tento čas využít produktivně. V paměti je současně uchováváno více úloh. V okamžiku, kdy musí běžící úloha čekat, je CPU přiděleno úloze jiné.

Plánování je základem fungování moderního OS. V takovém systému jsou plánovány veškeré jeho zdroje a CPU je pochopitelně zdroj nejdůležitější. Proto je plánování CPU ústředním bodem návrhu OS.

2.1 CPU cyklus – I/O cyklus (CPU – I/O Burst Cycle)

Běh procesu obvykle sestává z *cyklu* spuštění na CPU a čekání na I/O. Probíhající proces přechází mezi těmito dvěma *cykly*.

Spuštění procesu začíná CPU cyklem, následuje I/O cyklus, pak opět CPU cyklus atd. Nakonec při posledním CPU cyklu dojde k systémovému volání, které vede k ukončení procesu (při korektním průběhu programu).



Obrázek 1 - CPU a I/O cykly programu

2.2 Plánovač CPU

V okamžiku, kdy by mělo být CPU spuštěným procesem blokováno, OS ho přidělí jinému procesu z *fronty připravených*. Tento výběr provádí *plánovač CPU (short-term scheduler* nebo *CPU scheduler).*

Jak již bylo uvedeno, fronta připravených nemusí být a také většinou není prostou strukturou First In First Out. Fronta připravených může být implementována jako FIFO, fronta s prioritou, strom nebo i jako neuspořádaný seznam. Každý proces je ve frontě reprezentován zejména svým ID, resp. PCB.

2.3 Preemptivní plánování

K předělení (změně přidělení) CPU může dojít kvůli některému z následujících důvodů:

1. Jestliže proces přechází ze stavu *probíhající* do stavu *čekající* (např. kvůli I/O požadavku nebo kvůli čekání na dokončení synovského procesu).
2. Jestliže proces přechází ze stavu *probíhající* do stavu *připraven* (např. kvůli přerušení).
3. Jestliže proces přechází ze stavu *čekající* do stavu *připraven* (např. díky dokončení I/O operace).
4. Jestliže je proces ukončen.

Jestliže k plánování dochází pouze v případě 1 a 4, nazývá se toto plánování *nepreemptivní*, v opačném případě *preemptivní*.

Je-li v případě nepreemptivního plánování CPU přidělen procesu, ten ho opouští, pouze pokud je ukončen nebo čeká (např. na I/O operaci). Tato metoda plánování je užita v prostředí Microsoft Windows, nepotřebuje totiž speciální hardware jako preemptivní plánování (časovač apod.).

Preemptivní plánování však zvyšuje cenu výpočetního systému. Uvažujme případ dvou procesů sdílejících data. První proces právě provádí změnu dat, a je odstaven od procesoru, který je přidělen druhému procesu. Ten by pak mohl zkusit číst data, která jsou momentálně nekonzistentní. Aby se podobným situacím předešlo, je třeba důsledně synchronizovat běh procesu v preemptivním systému. Blíže viz kap. 6.

Preemptivní plánování musí být důkladně ošetřeno zejména vzhledem k jádru OS. Během vykonávání systémového volání může jádro pracovat ve prospěch procesu (např. měnit nějakou I/O frontu). Během tohoto měnění může být CPU přidělen jinému procesu, který se na tuto frontu může chtít odvolat => CHAOS.

Mnoho OS včetně většiny UNIXů řeší tento problém tak, že během vykonávání volání jádra není možno přidělit CPU jinému procesu. Volání jádra se vykonávají v *privilegovaném režimu*. Samozřejmě, že tento mechanismus znesnadňuje multiprocessing a výpočty v reálném čase.

2.4 Dispečer

Další části OS, která zajišťuje přidělování CPU je *dispečer*. Dispečer je modul, který má kontrolu nad CPU a procesem vybraným pro spuštění plánovačem CPU.

Funkce dispečera jsou:

- Přepínání kontextu;
- Přepínání mezi uživatelskými módy;
- Nalezení místa, kde byl uživatelský program přerušen a znovu ho spustit.

Dispečer musí být tak rychlý jak jen je to možné, protože je volán při každém přepínání kontextu. Čas potřebný k zastavení jednoho procesu a spuštění jiného je označován jako *čekací doba pro přidělení (dispatch latency)*.

3. KRITÉRIA PŘIDĚLOVÁNÍ CPU

Různé přidělovací algoritmy mají různé vlastnosti a mohou upřednostňovat jednu třídu procesů před jinými. Vybíráme-li, který algoritmus užít v určité konkrétní situaci, musíme uvažovat právě ony vlastnosti různých algoritmů.

Kritéria užitá pro posouzení plánování CPU jsou:

- **Využití procesoru (CPU utilization)** – Necháme CPU tak zaměstnaný jak jen to bude možné. Celkové využití procesoru je pak hodnota od 0 do 100 %. V reálném systému je využití CPU přibližně od 40 % při malém zatížení do 90 % při velkém zatížení.
- **Propustnost (Throughput)** – počet procesů, které jsou vykonány za jednotku času.
- **Doba obrátky (Turnaround time)** – doba, jak dlouho je spuštěn daný proces. Suma času, kdy proces čeká na zavedení do paměti, čeká ve frontě připravených, je spuštěn na CPU a provádí I/O operace.
- **Doba čekání (Waiting time)** – celková doba, jak dlouho čekal proces ve frontě připravených. Jak dlouho čekal na dokončení I/O či jak dlouho byl spuštěn není pro plánování CPU významná.
- **Doba odezvy (Response time)** – doba od odeslání dotazu po první reakci systému. Nezahrnuje v sobě dobu cele odpovědi, protože ta může být závislá např. na rychlosti výstupního zařízení.

Každý OS se snaží maximalizovat využití CPU a propustnost a minimalizovat dobu obrátky, čekání a odezvy. Mnohdy je výhodnější minimalizovat průměr, než minimum či maximum. V interaktivních systémech (jako např. v systémech se sdílením času) je výhodnější minimalizovat rozptyl doby odezvy než její průměr.

Pro zjednodušení porovnání jednotlivých přidělovacích algoritmů budeme uvažovat jen jeden CPU cyklus v procesu a mírou efektivity bude průměrná doba čekání.

4. ALGORITMY PLÁNOVÁNÍ CPU

4.1 Algoritmus FCFS

Algoritmus první přišel, první dostal (First-Come, First-Served Scheduling) je zdaleka nejjednodušším algoritmem plánování CPU. Jedna se o "FIFO" implementaci a ten proces, který prvně požádá o přidělení CPU, ho dostane.

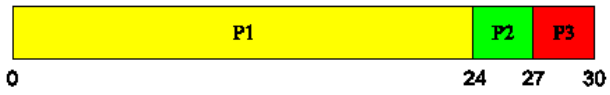
Implementace FCFS algoritmu je jednoduše proveditelná FIFO frontou. Jakmile proces vstoupí do fronty připravených, je jeho PCB zapojen na konec fronty. Je-li CPU volný, alokuje se pro procesor na vrcholu fronty. Spuštěný proces je z fronty odstraněn.

Kód FCFS algoritmu je snadno pochopitelný a naprogramovatelný.

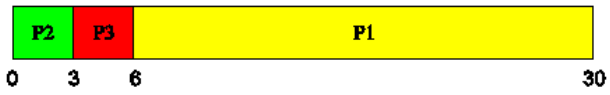
Průměr čekací doby u FCFS algoritmu je vsak dosti velký. Uvažujme sled procesů, který nastal od času 0 s následující délkou CPU cyklu v ms:

Proces	Délka CPU cyklu
P1	24
P2	3
P3	3

Dorazí-li procesy v pořadí P1, P2, P3 a budou obslouženy FCFS algoritmem, výsledkem bude následující diagram:



Čekací doba procesu P1 je 0 ms, procesu P2 je 24 ms a pro P3 je 27 ms. Průměrná čekací doba je (0 + 24 + 27)/3 = 17 ms. Dorazí-li procesy v pořadí P2, P3, P1 výsledek bude následující:



Nyní bude čekací doba procesu P1 je 0 ms, procesu P2 je 3 ms a pro P3 je 6 ms. Průměrná čekací doba je (0 + 3 + 6)/3 = 3 ms. Rozdíl v průměrné čekací době obou případů je značný, doba čekání není minimalizovaná a může silně kolísat v závislosti na velikosti CPU cyklu za sebou jdoucích procesu.

Uvažujme FCFS algoritmus v dynamické situaci. V systému je jeden proces výrazně orientovaný na CPU (CPU-bound process) a několik menších procesů více méně pracujících na periferiích. V okamžiku, kdy se CPU-bound proces dostane k CPU, drží si ho velmi dlouho. Za tu dobu všechny ostatní procesy nejspíše dokončily své I/O operace a přesunuly se do fronty připravených. Tou dobou již jsou I/O fronty vesměs prázdné. Nato CPU-bound proces dokončí svůj CPU cyklus a přesune se do nějaké I/O fronty. Potom všechny ostatní I/O-bound procesy rychle vykonají své krátké CPU cykly opět se rozprchnou do I/O front a CPU je nevyužito. Potom někdy CPU-bound proces dokončí svou I/O operaci, vrátí se do fronty připravených a alokuje opět na značnou dobu CPU. A opět čekají I/O-bound procesy ve frontě připravených a tak stále dokola. Tato situace bývá označována jako *konvoj efekt*, kdy všechny ostatní procesy čekají na jeden velký, který obsadil CPU.

Důsledkem konvoj efektu je pak výrazně nižší využití CPU a I/O zařízení, než by tomu bylo za situace, kdy by krátké procesy dostaly CPU dříve, než ten velký.

FCFS je nepreemptivní algoritmus přidělování CPU. Dostane-li proces CPU, drží si ho do té doby, než je ukončen nebo než začne provádět nějakou I/O operaci. Je nevhodný pro systémy se sdílením času, kde by měl každý uživatel dostat odpověď v přiměřené době.

4.2 Algoritmus SJF

První jde nejkratší úloha (Shortest-Job-First Scheduling) používá jiný přístup k plánování CPU. V tomto algoritmu je s každým procesem asociován čas jeho následujícího CPU cyklu. Je-li CPU volný, je přidělen procesu s nejkratším následujícím CPU cyklem.

Jestliže má více procesů stejnou dobu následujícího CPU cyklu, je mezi nimi rozhodnuto algoritmem FCFS.

Poznamenejme, že v rozporu s názvem algoritmu, vhodnější pravidlo je *shortest-next-CPU-burst- first*, tedy plánovat podle délky následujícího CPU cyklu každého procesu, než podle jeho celkové délky. Algoritmus budeme i nadále nazývat SJF, neboť většina lidi i učebnic toto jméno používá.

Pro příklad uvažujme následující množinu procesů s uvedenou délkou následujícího CPU cyklu:

Proces	Délka CPU cyklu
P1	6
P2	8
P3	7
P4	3

Užitím SJF plánování bude procesům přidělen procesor následovně:

Doba čekání pro proces P1 je 3 ms, pro proces P2 je 16 ms, pro proces P3 je 9 ms a pro proces P4 je 0 ms. Průměrná doba čekání je (3 + 16 + 9 + 0)/4 = 7 ms. Při užití FCFS plánování by byla průměrná doba 10,25 ms.

SJF algoritmus vykazuje minimální průměrnou dobu čekání a je v tomto směru *optimální*. Při přesunutí "krátkého" procesu před "dlouhý" se více zkrátí čekací doba "krátkého" procesu než se prodlouží čekací doba procesu "dlouhého" a tím se průměrná čekací doba zmenší.

Reálný problém tohoto algoritmu představuje nutnost vědět dopředu délku následujícího CPU cyklu. U dávkových systémů je možno užít časový limit pro proces, který specifikuje uživatel při předávání úlohy OSu. Uživatelé jsou zde nuceni poměrně přesně tuto hodnotu odhadnout, neboť menší hodnota znamená rychlejší výsledek (příliš malá potom ale vyvolá chybu vypršení časového kvanta a je nutno zadat úlohy znova). U těchto systémů je SJF často užívaným algoritmem plánování CPU.

Ačkoli je SJF optimální algoritmus přidělování CPU, nemůže být implementován v systémech s krátkodobým plánováním (např. v systémech se sdílením času). Tady totiž není možné vědět dopředu přesnou délku následujícího CPU cyklu. Je samozřejmě možnost tento algoritmus aproximovat a onu hodnotu vždy nějak odhadnout. Koncepce je nasnadě – očekávejme, že následující CPU cyklus procesu bude přibližně stejně dlouhý jako předcházející, a potom opět vybrat pro přidělení CPU proces s nejmenší odhadnutou hodnotou.

Délka dalšího CPU cyklu procesu je odhadnuta jako exponenciální průměr všech předcházejících CPU cyklů procesu. Nechť t_n je délka n-tého CPU cyklu procesu a τ_{n+1} nechť je odhadnutá délka následujícího n+1. cyklu, $0 <= \alpha <= 1$. Potom:

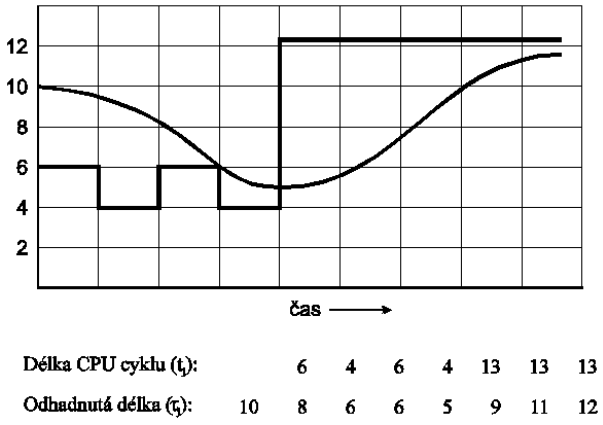
$$\tau_{n+1} = \alpha \cdot t_n + (1 - \alpha) \cdot \tau_n$$

Hodnota t_n obsahuje poslední přesnou informaci, hodnota τ_n je minulá odhadnutá hodnota. α vyjadřuje relativní váhu obou hodnot pro nový odhad. Jestliže $\alpha = 0$, potom $\tau_{n+1} = \tau_n$ a délka posledního CPU cyklu nemá na odhad žádný vliv (poslední cyklus je považován za přechodný). Jestliže $\alpha = 1$, potom $\tau_{n+1} = t_n$ (odhad je vytvořen pouze na základě posledního CPU cyklu, historie je považována za starou a nerelevantní). Velmi často $\alpha = 1/2$, kdy význam doby posledního cyklu je stejný jako význam předchozí odhadnuté hodnoty – uplynulé historie.

Celkový význam uvedeného vzorce bude jasnější z jeho nerekurzivní podoby:

$$\tau_{n+1} = \alpha \cdot t_n + (1 - \alpha)\alpha \cdot t_{n-1} + \ldots + (1 - \alpha)^j \alpha \cdot t_{n-j} + \ldots + (1 - \alpha)^{n+1} \tau_0$$

Protože obě hodnoty α_i, $(1 - \alpha)$ jsou menší nebo rovny jedné má každý následující výraz v historii menší váhu než jeho předchůdce. τ_0 je systémová konstanta nebo celkový systémový průměr.



Obrázek 2 - Odhad délky následujícího CPU cyklu a skutečnost pro α = ½

SJF algoritmus může být jak *preemptivní* tak i *nepreemptivní*. K výběru dochází, když nový proces dorazí do fronty připravených, zatímco předcházející je ještě spuštěný. Nový proces může mít kratší CPU cyklus. Při preemptivním plánování bude probíhající proces odstaven a CPU přidělen novému procesu, zatímco nepreemptivní SJF algoritmus nechá původní proces dokončit jeho CPU cyklus.

Preemptivní SJF algoritmus se někdy nazývá *shortest-remaining-time-first* scheduling.

Jako příklad uvažujme následující 4 procesy s uvedenou délkou jejich následujícího CPU cyklu:

Proces	Čas přijetí	Délka CPU cyklu
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Jestliže procesy dorazí do fronty připravených tak, jak je uvedeno, výsledek preemptivního SJF plánování je vidět na následujícím obrázku:

Proces P1 startoval v čase 0, jako jediný proces ve frontě. Proces P2 dorazil do fronty 1 ms potom, a protože délka CPU cyklu u P2 je 4, zatímco u P1 je 8, byl procesor procesu P1 odebrán a přidělen procesu P2. Průměrná doba čekání je v tomto případě: ((10 – 1) + (1 – 1) + (17 – 2) + (5 - 3))/4 = 6,5 ms. Při nepreemptivním plánování by průměrná doba čekání činila 5,75 ms.

4.3 Plánování podle priority

Algoritmus SJF je speciálním případem plánování podle *priority*. Principiálně toto plánování znamená, že každý proces je asociován se svou prioritou a CPU je přidělen procesu s nejvyšší prioritou. Mezi procesy se stejnou prioritou se plánuje FCFS algoritmem.

V případě SJF plánování je prioritou *p* převrácená hodnota odhadnuté doby τ. Platí tedy $p = \tau^{-1}$. Čím větší je odhadnutá doba, tím menší je priorita daného procesu a naopak.

Priorita může být definována buď *interní* nebo *externí*. Interní priorita užívá měřitelné nebo počitatelné veličiny, jako např.: časový limit, paměťové nároky, počet otevřených souborů nebo poměr průměrné délky CPU a I/O cyklu pro výpočet priority.

Externí priorita je nastavována podle vnějších kritérií OS, jako např. důležitost procesu a další často politická kritéria.

Plánování dle priority může být *preemptivní* i *nepreemptivní*. V okamžiku, kdy proces dorazí do fronty připravených, je jeho priorita porovnána s procesem právě spuštěným a je-li vyšší, u preemptivního plánování je procesor přidělen ihned novému procesu, u nepreemptivního je nový proces zařazen na vrchol fronty.

Hlavním problémem plánování dle priority je *neomezené zablokování (indefinite blocking)* nebo *umoření (starvation)* procesu. Procesy s velmi nízkou prioritou by totiž mohly na CPU čekat až neomezeně dlouho. V silně zatíženém výpočetním systému může soustavný tok procesů s vyšší prioritou zabránit procesu s prioritou nižší jakkoli se dostat k CPU. Globálně nastane jedna ze dvou možností: buď se proces spustí v neděli ve 14.00 hod, kdy je nízké zatížení nebo při možném výpadku dojde ke ztrátě tohoto procesu. (Když v roce 1973 provedly na MIT (Massachussets Institute of Technology) shut down systému IBM 7094, našli proces s nízkou prioritou zadaný v roce 1967, a který do té doby nebyl ani jednou spuštěn).

Řešením problému neomezeného zablokování procesu je *aging*. Aging je technika, která významně zvýší prioritu procesu, který je v systému dlouhou dobu. Např. je-li priorita definována na intervalu 0 – 127, je možné inkrementovat prioritu o 1 každých 15 minut. Takto každý proces, i ten s nejnižší prioritou dosáhne časem priority vyšší a bude spuštěn. I v nejhorším případě to nebude trvat déle než 32 hodin. Důležitou součástí algoritmu je návrat priority takového vystárnutého procesu na původní hodnotu poté, co opustí stav *probíhající*. Pokud by k tomu totiž nedošlo, zůstal by tento proces na začátku fronty připravených, ostatní procesy by postupně prošly procesem stárnutí také a nakonec by všechny procesy v systému měly stejnou (nejvyšší) prioritu. Celý systém plánování by degradoval na plánování FCFS.

4.4 Algoritmus RR

Plánování cyklickou obsluhou (Round–Robin Scheduling) je vhodné zejména pro OS se sdílením času. Je podobný algoritmu FCFS, obsahuje však navíc možnost preemptivního plánování. V systému je definována malá časová jednotka – *časové kvantum* – většinou 10–100 ms. Fronta připravených je definována jako cyklická fronta a plánovač CPU touto frontou prochází stále dokola a přiděluje CPU jednotlivým procesům vždy na jedno časové kvantum.

Při RR plánování je fronta připravených implementována formou běžné FIFO fronty procesů a nový proces je vždy zařazen na její konec. Plánovač CPU vezme proces na vrcholu, nastaví časovač na přerušení po uplynuti 1 časového kvanta a přidělí procesu CPU.

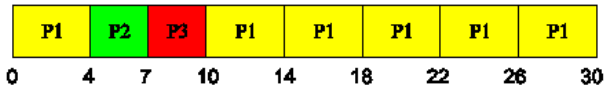
Nastane jedna ze dvou možností:

- CPU cyklus procesu může být kratší než časové kvantum. V tom případě proces sám uvolní CPU a plánovač ho může přidělit dalšímu procesu z fronty připravených.
- CPU cyklus aktuálně spuštěného procesu je delší než jedno časové kvantum a během vykonávání procesu vyvolá časovač přerušení. Nastává přepnutí kontextu a stávající proces je umístěn na konec fronty připravených. Plánovač CPU potom z fronty připravených vybere další proces pro přidělení CPU.

Průměrná čekací doba při RR plánování je ovšem dosti značná. Uvažujme následující množinu procesů, jež dorazily do fronty připravených v čase 0 a délka jejich CPU cyklu je uvedena:

Proces	Délka CPU cyklu
P1	24
P2	3
P3	3

Užijeme-li jako délku časového kvanta 4 ms, potom proces P1 dostane přidělen procesor na 4 ms, ovšem pro dokončení svého CPU cyklu ho potřebuje ještě dalších 20 ms. Jeho běh je vsak preemptivně ukončen a CPU přidělen procesu P2. Ten ani cele 4 ms nevyčerpá a po 3 ms je jeho CPU cyklus ukončen a CPU se přidělí procesu P3. Protože každý proces ve frontě připravených již měl CPU přidělen na 1 časové kvantum, je CPU opět přidělen procesu P1, fronta připravených je prázdná, takže je mu procesor přidělen 6x za sebou:

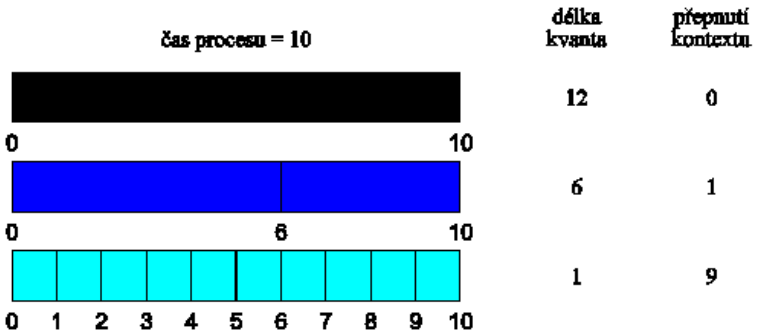


Čekací doba procesu P1 je 6 ms, procesu P2 je 4 ms a procesu P3 je 7 ms. Průměrná čekací doba je tedy 17/3 = 5,66 ms.

V případě RR plánování nedostane žádný proces najednou CPU na dobu delší než 1 časové kvantum. Pokud je CPU cyklus procesu delší než 1 kvantum je proces preemptivně odloučen od CPU a zařazen na konec fronty připravených. RR algoritmus je *preemptivní*.

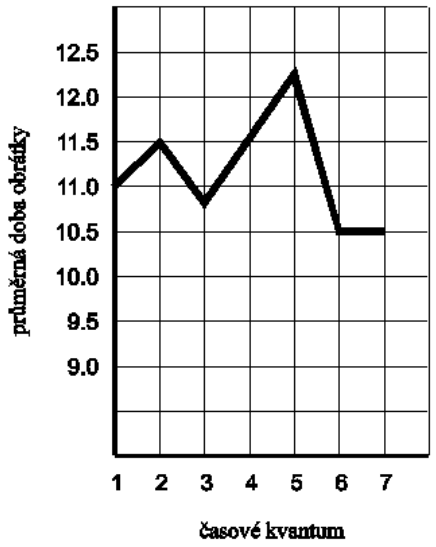
Efektivita RR plánování značně závisí na délce časového kvanta. Je-li časové kvantum příliš velké, RR algoritmus se přeměňuje na FCFS algoritmus se všemi jeho nedostatky. Je-li časové kvantum příliš malé (řekněme 1 ms), stává se RR přístup *sdílením procesoru* a uživatelsky se (teoreticky) zdá, že každý z *n* procesů je spuštěn na vlastním procesoru s výkonem 1/*n* oproti skutečnému procesoru systému. Tento přístup využívá CDC hardware (Control Data Corporation), tak že implementuje 10 virtuálních procesorů jedním fyzickým a 10 sadami registrů. Tento hardware spustí jednu instrukci na jedné sadě registrů a jde na následující. Celkově je program vykonáván jakoby deseti pomalejšími procesory namísto jedním rychlým. (Každá instrukce referuje paměť, která je mnohem pomalejší než rychlý procesor, takže skutečné zpomalení běhu programu je minimální).

Při definici časového kvanta je třeba brát v úvahu počet přepínání kontextu, který se zvyšuje se snižující se délkou časového kvanta. Uvažujme proces, který potřebuje 10 časových jednotek CPU. Je-li délka časového kvanta 12 jednotek, procesu stačí jedno kvantum, které ani celé nevyužije. Byla-li by délka kvanta 6 jednotek, spotřebuje proces kvanta 2 a dojde k přepnutí kontextu. Při délce kvanta 1 jednotka by došlo k 9 přepínáním kontextu, což by už značně zpomalilo běh procesu:



Obrázek 3 - Čím menší časové kvantum, tím více přepínání kontextu

Časové kvantum by tedy mělo být rozumně velké vzhledem k délce přepínání kontextu. Je-li doba nutná k přepnutí kontextu 10 % délky časového kvanta, potom 10 % CPU času padne pouze na přepínání kontextu. Patří mezi režijní činnosti.



proces	čas
P1	6
P2	3
P3	1
P4	7

Obrázek 4 - Vztah průměrné doby obrátky a délky časového kvanta

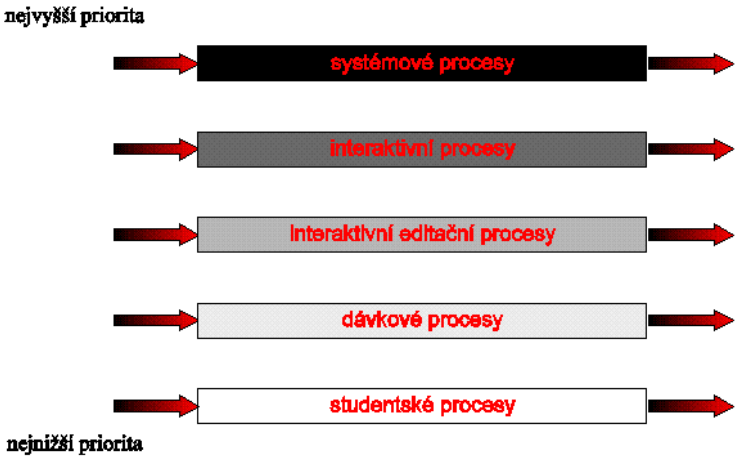
Délka časového kvanta přímo podmiňuje dobu obrátky (turnaround time) procesu. Jak je vidět z obr 4, průměrná doba obrátky množiny procesů se nemusí nutně zlepšovat s prodlužováním časového kvanta. Globálně se průměrná doba obrátky zlepšuje, jestliže mnoho procesů vykoná svůj CPU cyklus během jednoho časového kvanta.

Na druhé straně, je-li časové kvantum příliš velké, RR plánování degeneruje na FCFS.

4.5 Algoritmus MQF

Plánování pomocí více front (Multilevel Queue Scheduling) bylo sestaveno pro situace, kdy procesy jsou rozděleny do skupin. Jednoduchým příkladem budiž interaktivní procesy a procesy na pozadí (dávkové procesy). Obě skupiny procesu mají různé nároky na doby odezvy a měly by tedy být odlišně plánovány. Procesy na popředí mají vyšší prioritu (externí) než procesy na pozadí.

Algoritmus plánování pomocí více front dělí frontu připravených na několik samostatných front, viz obr. 5. Každý proces je stále spojen s jednou frontou na základě nějaké vlastnosti procesu – požadavku na paměť, priority nebo typu procesu.



Obrázek 5 - Plánování systémem více front

Na obr. 5 je víceúrovňový plánovací systém využívající 5 front:

Každá fronta má svůj vlastní plánovací algoritmus. Fronta interaktivních procesů může být např. plánována pomocí RR algoritmu a fronta dávkových procesů může být plánována FCFS algoritmem.

Musí existovat plánování mezi jednotlivými frontami, většinou implementované jako preemptivní plánování s pevnou prioritou. Interaktivní procesy mají absolutní přednost před dávkovými.

- Systémové procesy
- Interaktivní procesy
- Interaktivní editační procesy
- Dávkové procesy
- Studentské procesy

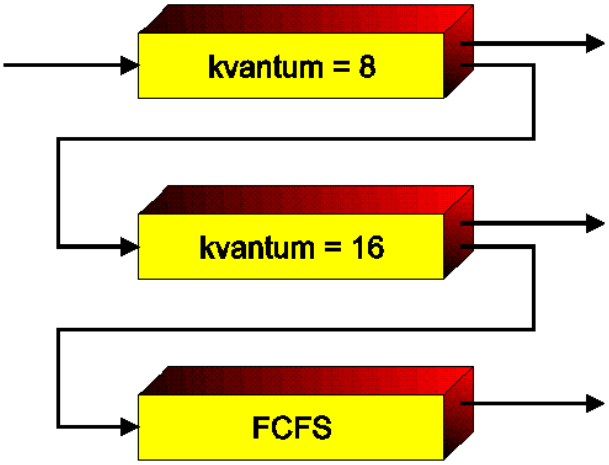
Každá předcházející fronta má absolutní prioritu před každou frontou následující. Žádný proces např. z fronty dávkových procesů nemůže dostat procesor, dokud fronta systémových a obou interaktivních procesů nebude prázdná. Jestliže nějaký interaktivní proces vstoupí do fronty připravených, zatímco procesor je přidělen dávkovému procesu, je dávkovému procesu procesor preemptivně odebrán.

Další možností je definovat časové intervaly přidělení CPU jednotlivým frontám. Každá fronta dostane určitou část času CPU, během kterého může plánovat procesy v ní uložené. Např. interaktivní procesy mohou dostat 80 % času CPU a během nich jsou plánovány RR algoritmem a dávkové procesy dostanou 20 % času CPU a během nich jsou plánovány FCFS algoritmem.

4.6 Algoritmus MFQS

Plánování pomocí MFQS (Multilevel Feedback Queue Scheduling) využívá více front se zpětnou vazbou. V běžném systému plánování pomocí více front je každému procesu přidělena fronta, ve které bude vždy čekat na CPU již v okamžiku vytvoření procesu. Procesy se mezi frontami nemohou přesouvat. Tento systém snižuje režii plánování, ale není příliš flexibilní.

Plánování pomocí více front se zpětnou vazbou umožňuje procesům přesouvat se mezi frontami. Hlavni myšlenka je oddělit procesy s různou charakteristikou CPU cyklu. Jestliže nějaký proces využívá CPU příliš mnoho, bude přesunut do fronty s nižší prioritou, aby tolik nezatěžoval OS. A naopak, proces, který již velmi dlouho čeká ve frontě s nižší prioritou, může být přesunut do fronty s prioritou vyšší – prevence před neomezeným zablokováním procesu.



Obrázek 6 - Plánování pomocí více front se zpětnou vazbou

Toto schéma ponechává vysokou prioritu interaktivním a I/O-bound procesům.

Uvažujme systém plánování pomocí více front se zpětnou vazbou se 3 frontami 0 až 2 (viz obr. 6). Plánovač CPU nejprve obslouží všechny procesy ve frontě 0, pouze v případě, že je fronta 0 prázdná, dostane se na procesy ve frontě 1 a stejně tak procesům z fronty 2 může být CPU přidělen jedině v případě, že jak fronta 0 tak i 1 jsou prázdné. Proces z fronty 1 může preemptivně ukončit běh procesu z fronty 2, proces z fronty 0 může preemptivně ukončit běh procesu z fronty 1 i 2.

Proces, který vstoupí do fronty připravených je zařazen do fronty 0 a časové kvantum v této frontě je 8 ms. Jestliže za tuto dobu nestihne vykonat celý svůj CPU cyklus, je přesunut do fronty 1, kde je časové kvantum 16 ms a jestliže ani za tuto dobu nestihne dokončit svůj CPU cyklus je přesunut do fronty 2, kde je plánováno modelem FCFS.

Tento způsob plánování přiděluje nejvyšší prioritu procesům s délkou CPU cyklu 8 ms a menší. Procesy s délkou CPU cyklu mezi 8 a 16 ms jsou obsluhovány také rychle, ale s nižší prioritou než kratší procesy. Dlouhé procesy automaticky klesnou do fronty 2.

Plánování pomocí více front se zpětnou vazbou je v globále definováno:

- Počtem front.
- Plánovacím algoritmem v každé frontě.
- Metodou, která je užita k identifikaci procesu, který je třeba přesunout do fronty s vyšší prioritou.
- Metodou, která je užita k identifikaci procesu, který je třeba přesunout do fronty s nižší prioritou.
- Metodou, která je užita k identifikaci fronty, do které bude zařazen proces, když bude potřebovat CPU.

Plánování pomocí více front se zpětnou vazbou je jedním z nejvýznamnějších algoritmů plánování CPU. Může být konfigurován pro libovolný specifický systém, je velmi komplexní.