

06.
Čtenáři – písaři
Plánování procesů

ZOS 2014

1. zápočtový test

- 5. a 6. listopadu 2014
 - v čase cvičení (na poloviny dle domluvy na cvičení)
 - u školního PC pod speciálním účtem
 - 30 minut čistého času na test
 - otázka z 1.-4. prezentace přednášek
 - různé varianty testů
- Hodnocení
 - Každý úkol ANO/NE získá bodů
 - nadpoloviční většina bodů



1. zápočtový test

- Pomůcky k dispozici
 - manuálové stránky (man)
 - dostanete vytištěný seznam základních příkazů (viz portál předmětu)
 - nic víc

tipy:

- syntaxe: help if, help case, help test, man test
- umět poznat, že je skript spuštěný s 0, 1, 2, .. parametry
- umět iterovat přes soubory (včetně adresářů) v daném adresáři

Problém čtenářů a písarů

- modeluje přístup do databáze
- rezervační systém (místenky, letenky)
- množina procesů, souběžné čtení a zápis
 - souběžné čtení lze
 - výhradní zápis (žádný další čtenář ani písar)

Častá praktická úloha, lze realizovat s předností čtenářů, nebo s předností písarů.

Pro komerční aplikace je samozřejmě vhodnější přednost písarů.

var

m=1 : semaphore;	{mutex, chrání čítač}
w=1: semaphore;	{přístup pro zápis }
rc = 0: integer;	{ počet čtenářů }

procedure **writer**;

begin

P(w);

// zapisuj

V(w)

end;

```
procedure reader;  
begin  
    P(m);  
    rc := rc + 1;  
    if rc = 1 then P(w);    //1. čtenář blok. píše  
    V(m);  
    // čti  
    P(m);  
    rc := rc - 1;  
    if rc=0 then V(w);    // poslední čtenář odblok. pí.  
    V(m)  
end;
```

Čtenáři – písaři popis

■ čtenáři

- první čtenář provede $P(w)$
- další zvětšují čítač rc
- po “přečtení” čtenáři zmenšují rc
- poslední čtenář provede $V(w)$

■ semafor w

- zabrání vstupu písaře, jsou-li čtenáři
- zabrání vstupu čtenářům při běhu písaře:
 - prvnímu zabrání $P(w)$
 - ostatním brání $P(m)$

■ toto řešení je s předností čtenářů

- písaři musí čekat, až všichni čtenáři skončí

Implementace zámků v operačních a databázových systémech

- přístup procesu k souboru nebo záznamu databázi
- **výhradní zámek (pro zápis)**
 - nikdo další nesmí přistupovat
- **sdílený zámek (pro čtení)**
 - mohou o něj žádat další procesy
- **granularita zamykání**
 - celý soubor x část souboru
 - tabulka x řádka v tabulce

Implementace zámek v OS

Linux, UNIX lze zamknout část souboru funkcí

`fcntl (fd, F_SETLK, struct flock)`

```
int fd;  
struct flock fl;  
fd = open("testfile", O_RDWR);  
fl.l_type = F_WRLCK;  
fl.l_whence = SEEK_SET;  
fl.l_start = 100; fl.l_len = 10;  
fcntl (fd,F_SETLK, &fl);  
// vrátí -1 pokud se nepovede
```

- zámek pro zápis
- pozice od začátku souboru
- pozice, kolik
- `zamkneme pro zápis`

Implementace zámků v OS

odemknutí

```
fl.l_type = F_UNLCK;  
fl.l_whence = SEEK_SET;  
fl.l_start = 100; fl.l_len = 10;  
fcntl (fd, F_SETLK, &fl);
```

- odemknutí
- pozice od začátku souboru
- pozice, kolik
- nastavíme

operace

```
F_SETLK  
F_GETLK  
F_SETLKW
```

- set / clear lock, nečeká
- info o zámku
- nastavení zámku, čeká
když je zamčený

Implementace zámků v OS

- zámky jsou odstraněny, když proces skončí (teoreticky)
- zámky **poradní** (advisory)
 - nejsou vynucené
 - pro kooperující procesy
 - defaultní chování
- zámky **mandatory**
- různé způsoby zamykání:
fcntl, flock, lockf

Poznámky

■ mandatory vs. advisory locks:

<http://stackoverflow.com/questions/575328/fcntl-lockf-which-is-better-to-use-for-file-locking>

Locking in **unix/linux** is by default **advisory**, meaning other processes don't need to follow the locking rules that are set. So it doesn't matter which way you lock, as long as your co-operating processes also use the same convention.

Linux does support **mandatory** locking, but only if your **file system is mounted with the option on** and the file special attributes set. You can use **mount -o mand** to mount the file system and set the file attributes **g-x,g+s** to enable mandatory locks, then use **fcntl** or **lockf**. For more information on how mandatory locks work see [here](#).

Note that locks are applied not to the individual file, but to the **inode**. This means that 2 filenames that point to the same file data will **share the same lock status**.

In **Windows** on the other hand, you can actively **exclusively open a file**, and that will block other processes from opening it completely. Even if they want to. I.e. The locks are **mandatory**. The same goes for Windows and file locks. Any process with an open file handle with appropriate access can lock a portion of the file and no other process will be able to access that portion.

Zámky v DB systémech

např. s každým záznamem databáze sdružen zámeček
funkce:

db_lock_r(x)	zámeček pro čtení
db_lock_w(x)	uzamkne záznam x pro zápis
db_unlock_r(x)	odemčení záznamu x
db_unlock_w(x)	dtto

čtenáři – písaři s předností písařů

```
type zamek = record
  wc, rc: integer := 0;           // počet písařů a čtenářů
  mutw: semaphore := 1;          // chrání přístup k čítači wc
  mutr: semaphore := 1;          // chrání přístup k čítači rc
  wsem: semaphore := 1;          // blokování písařů
  rsem: semaphore := 1;          // blokuje 1. čtenáře (písař)
  rdel: semaphore := 1;          // blokování ostatních čtenářů
end;
```

Algoritmus je složitější, ale praktičtější
uveden jen na ukázkou
:= symbol přiřazení, = symbol porovnání

```
procedure db_lock_w(var x: zamek);
```

```
    // uzamčení záznamu pro zápis
```

```
begin
```

```
    P(x.mutw);
```

```
    x.wc:=x.wc+1;
```

```
    if x.wc=1 then P(x.rsem); // 1.písař zablokuje 1. čtenáře
```

```
    V(x.mutw);
```

```
    P(x.wsem); // blokování písařů
```

```
end;
```

```
procedure db_unlock_w(var x: zamek);
```

```
// odemčení zápisů pro zápis
```

```
// sníží počet písařů, poslední písař odblokuje čtenáře
```

```
begin
```

```
    V(x.wsem);                // odblokování písařů
```

```
    P(x.mutw);
```

```
    x.wc:=x.wc-1;
```

```
    if x.wc=0 then V(x.rsem); // poslední písař pustí 1.čten.
```

```
    V(x.mutw)
```

```
end;
```



```
procedure db_lock_r(var x: zamek);  
begin  
    P(x.rdel);           // nejsou blokováni ostatní čtenáři  
    P(x.rsem);           // není blokován 1. čtenář  
    P(x.mutr);  
    x.rc:=x.rc+1;  
    if x.rc=1 then P(x.wsem); // 1. čtenář zablokuje pisaře  
    V(x.mutr);  
    V(x.rsem);  
    V(x.rdel)  
end;
```

```
procedure db_unlock_r(var x: zamek);  
begin  
    P(x.mutr);  
    x.rc:=x.rc-1;  
    if x.rc=0 then V(x.wsem); // poslední čtenář odblokuje pisaře  
    V(x.mutr)  
end;
```

Další problémy meziprocesové komunikace

- problém spícího holiče
- problém populárního pekaře (Lamport 1974)
 - Google: Lamport baker
 - Každý zákazník dostane unikátní číslo
- plánovač hlavičky disku
- další probrané
 - problém večeřících filozofů
 - producent – konzument
 - čtenáři – písaři
- Knížka *The Little Book of Semaphores* (zdarma pdf)

Plánování procesů

Základní stavy procesu

- **běžící**
- **připraven** – čeká na CPU
- **blokován** – čeká na zdroj nebo zprávu
- **nový (new)** – proces byl právě vytvořen
- **ukončený (terminated)** – proces byl ukončen

Správce procesů – udržuje **tabulku procesů**

Záznam o konkrétním procesu – **PCB** (Process Control Block) – souhrn dat potřebných k řízení procesů

opakování (!!)

v Linuxu je datová struktura `task_struct`, která obsahuje informace o procesu (tj. představuje PCB)

- každý proces má záznam (řádku) v **tabulce procesů**
- tomuto záznamu se říká **PCB** (process control block)
- PCB obsahuje všechny potřebné informace (tzv. **kontext** procesu) k tomu, abychom mohli proces **kdykoliv pozastavit** (odejmout mu procesor) a znovu jej od tohoto místa přerušení spustit (Program Counter: CS:EIP)
- proces po opětovném přidělení CPU pokračuje ve své činnosti, jako by k žádnému přerušení vykonávání jeho kódu nedošlo, je to z jeho pohledu transparentní

opakování (!!)

- kde leží tabulka procesů?
v paměti RAM, je to datová struktura jádra OS
- kde leží informace o PIDu procesu?
v tabulce procesů -> v PCB (řádce tabulky) tohoto procesu
- jak procesor ví, kterou instrukci procesu (vlákna) má vykonávat?
podle program counteru (PC, typicky CS:EIP), ukazuje na oblast v paměti, kde leží vykonávaná instrukce;
obsah CS:EIP, stejně jako dalších registrů je součástí PCB

opakování (!!)

- jak vytvořím nový proces?
systémovým voláním `fork()`
- jak vytvořím nové vlákno?
voláním `pthread_create()`
- jak spustím jiný program?
systémovým voláním `execve()`
začne vykonávat kód jiného programu
v rámci existujícího procesu

Plánovač x dispatcher

- **plánovač** vs. **dispatcher**
- **dispatcher** předává řízení procesu vybranému short time plánovačem:
 - přepnutí kontextu
 - přepnutí do user modu
 - skok na vhodnou instrukci daného programu, aby pokračovalo jeho vykonávání
- více připravených procesů k běhu – plánovač vybere, který spustí jako první
- plánovač procesů (**scheduler**)
používá plánovací algoritmus (**scheduling algorithm**)

Pamatuj

Plánovač určí, který proces (vlákno) by měl běžet nyní.

Dispatcher provede vlastní přepnutí z aktuálního běžícího procesu na nově vybraný proces.

Plánování procesů - vývoj

■ dávkové systémy

- Úkol: spustit další úlohu, nechat ji běžet do konce
- Uživatel s úlohou nekomunikuje, zadá program plus vstupní data např. v souboru
- O výsledku je uživatel informován, např. e-mailem aj.

■ systémy se sdílením času

- Můžeme mít procesy běžící na pozadí
- interaktivní procesy – komunikují s uživatelem

■ kombinace obou systémů (dávky, interaktivní procesy)

■ chceme: přednost interaktivních procesů

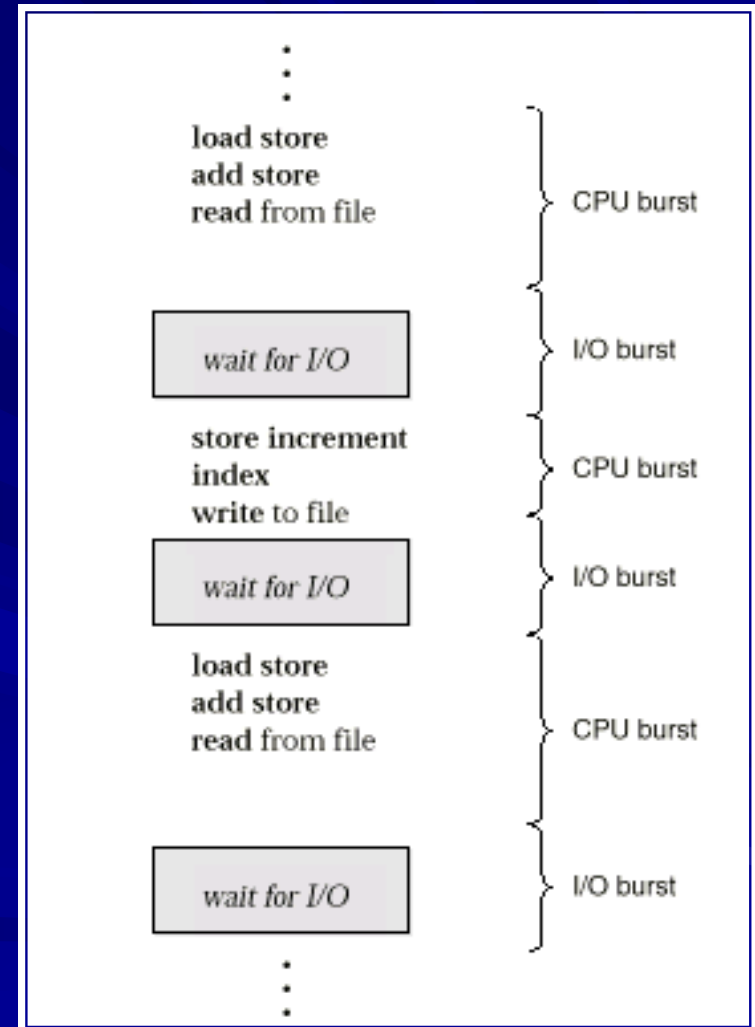
- Srovnejte: odesílání pošty x zavírání okna

Střídání CPU a I/O aktivit procesu

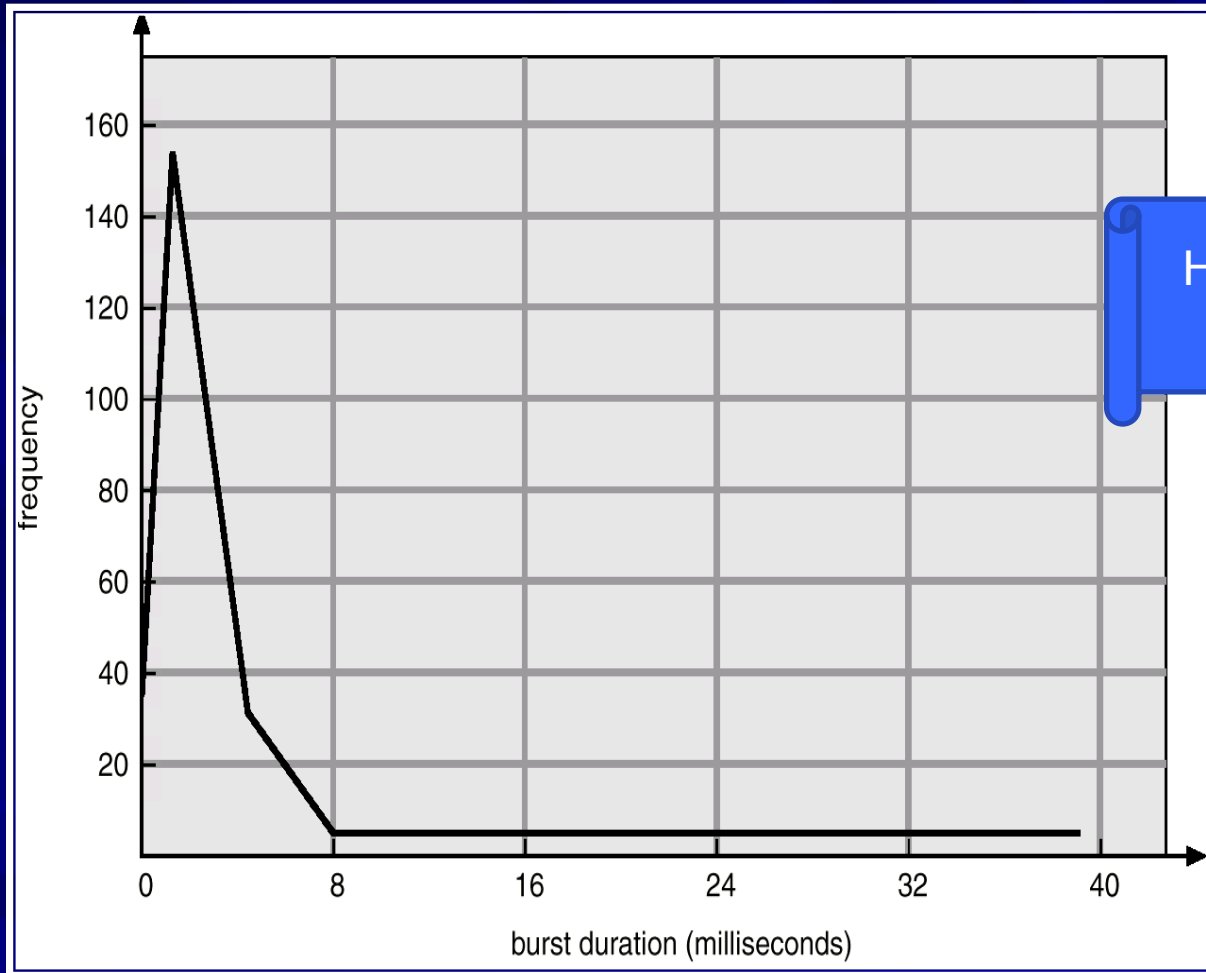
Během vykonávání procesu

- CPU burst (vykonávání kódu)
- I/O burst (čekání)
- střídání těchto fází
- končí CPU burstem

Typicky máme:
hodně krátkých CPU burstů
málo dlouhých



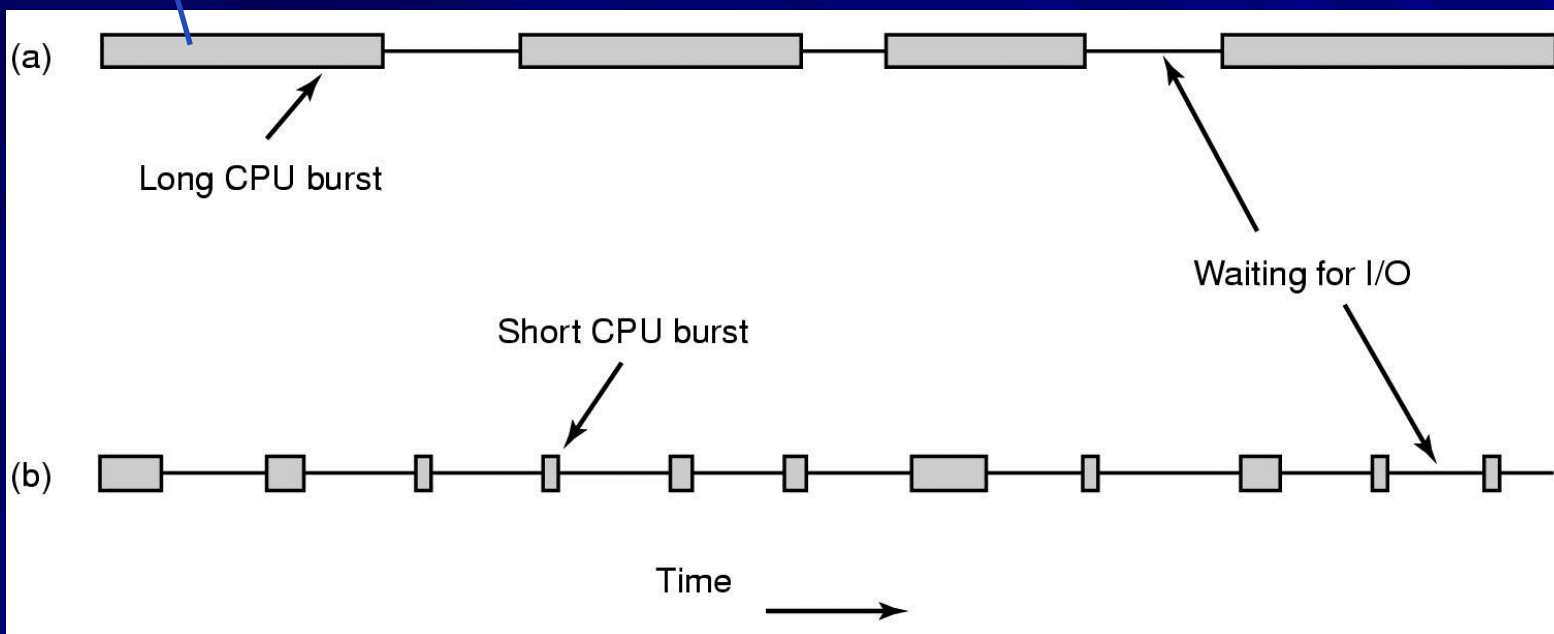
Histogram CPU burstů



Hodně krátkých
burstů

počítám

Plánování



- a) CPU-vázaný proces („hodně času tráví výpočtem“)
- b) I/O vázaný proces („hodně času tráví čekáním na I/O“)

Uveďte příklady CPU vázaného a I/O vázaného procesu

Preemptivní vs. non-preemptivní plánování

■ Non-preemptivní

- každý proces dokončí svůj CPU burst (!!!!)
- proces si podrží kontrolu nad CPU, dokud se jí nevzdá (I/O čekání, ukončení)
- lze v dávkových systémech, není příliš vhodné pro time sharingové (se sdílením času)
- Win 3.x non-preemptivní (kooperativní) plánování
- od Win95 preemptivní
- od Mac OS 8 preemptivní
- na některých platformách je stále

Jaký má vliv non-preemptivnost systému na obsluhu kritické sekce u jednojádrového CPU?

Preemptivní vs. non-preemptivní plánování

■ Preemptivní plánování

- proces lze přerušit **KDYKOLIV** během CPU burstu a naplánovat jiný (-> problém kritických sekcí !!!)
- dražší implementace kvůli přepínání procesů (režie)
- Vyžaduje speciální hardware – **timer (časovač)**
časovač je na základní desce počítače, pravidelně generuje hardwarová přerušení systému od časovače

Část výkonu systému spotřebuje režie nutná na přepínání procesů. K přepnutí na jiný proces také může dojít v nevhodný čas (ošetření KS).

Preemptivnost je ale u současných systémů důležitá, pokud potřebujeme interaktivní odezvu systému.

Časovač tiká (generuje přerušení), a po určitém množství tiků se určí, zda procesu nevypršelo jeho časové kvantum.

Otázky preemptivní plánování

- Nutnost koordinovat přístup ke sdíleným datům
- preempce jádra OS
 - přeplánování ve chvíli, kdy se manipuluje s daty (I/O fronty) používanými jinými funkcemi jádra..
 - UNIX (když nepreemptivní)
 - čekání na dokončení systémového volání
 - nebo na dokončení I/O
 - výhodou jednoduchost jádra
 - nevýhodou výkon v RT a na multiprocsoch

Preempce se může týkat nejen **uživatelských procesů**, ale i **jádra OS**. Linux umožňuje zkompileovat preemptivní jádro.

Cíle plánování - společné

■ Spravedlivost (Fairness)

- Srovnatelné procesy srovnatelně obsloužené

■ Vynucení politiky (Policy enforcement)

- Bude vyžadováno dodržení stanovených pravidel

■ Balance (Balance)

- Snaha, aby všechny části systému (CPU, RAM, periferie) byly vytížené

■ Nízká režie plánování

Cíle plánování

– dávkové systémy

■ Propustnost (Throughput)

- maximalizovat počet jobů za hodinu

■ Doba obrátky (Turnaround time)

- minimalizovat čas mezi přijetím úlohy do systému a jejím dokončením

■ CPU využití

- snaha mít CPU pořád vytížené

Cíle plánování

All systems

- Fairness - giving each process a fair share of the CPU
- Policy enforcement - seeing that stated policy is carried out
- Balance - keeping all parts of the system busy

Batch systems

- Throughput - maximize jobs per hour
- Turnaround time - minimize time between submission and termination
- CPU utilization - keep the CPU busy all the time

Interactive systems

- Response time - respond to requests quickly
- Proportionality - meet users' expectations

Real-time systems

- Meeting deadlines - avoid losing data
- Predictability - avoid quality degradation in multimedia systems

Některé cíle jsou společné, jiné se liší dle typu systému

Zajímavosti

V roce 1973 provedli na MITu shut-down systému IBM 7094 a našli low priority proces, který nebyl dosud spuštěný a přitom byl založený

Zajímavosti

..v roce 1967 ..

Plánovač (!)

■ rozhodovací mód

- okamžik, kdy jsou vyhodnoceny priority procesu a vybrán proces pro běh

■ prioritní funkce

- určí prioritu procesu v systému

■ rozhodovací pravidla

- jak rozhodnout při stejné prioritě

Tři zásadní údaje, které charakterizují plánovač

Plánovač – Rozhodovací mód

■ nepreemptivní

- Proces využívá CPU, dokud se jej sám nevzdá (např. I/O)
- jednoduchá implementace
- vhodné pro dávkové systémy
- nevhodné pro interaktivní a RT systémy

■ preemptivní

- kdy ?
 - přijde nový proces (dávkové systémy)
 - periodicky – kvantum (interaktivní systémy)
 - jindy – priorita připraveného > běžícího (RT)
- náklady
 - přepínání procesů, logika plánovače



Plánovač – Prioritní funkce

- Funkce, bere v úvahu parametry procesu a systémové parametry
- určuje prioritu procesu v systému
- externí priority
 - třídy uživatelů, systémové procesy
- priority odvozené z chování procesu (dlouho neběžel, čekal ...)
- Většinou **dvě složky** – statická a dynamická priorita
 - **Statická** – přiřazena při startu procesu
 - **Dynamická** – dle chování procesu (dlouho čekal, aj.)

Prioritní funkce (!)

priorita = statická + dynamická

proč 2 složky?

pokud by chyběla:

- statická – nemohl by uživatel např. při startu označit proces jako důležitější než jiný
- dynamická – proces by mohl vyhladovět, mohl by být neustále předbíhán v plánování jinými procesy s větší prioritou

Plánovač – Prioritní funkce

Co všechno může vzít v úvahu prioritní funkce:

- čas, jak dlouho proces využíval CPU
- aktuální zatížení systému
- paměťové požadavky procesu
- čas, který strávil v systému
- celková doba provádění úlohy (limit)
- urgency (RT systémy)

Plánovač – Rozhodovací pravidlo

- malá pravděpodobnost stejné priority
 - náhodný výběr
- velká pravděpodobnost stejné priority
 - cyklické přidělování kvanta
 - chronologický výběr (FIFO)

Prioritní funkce může být navržena tak, že málokdy vygeneruje stejné priority, nebo naopak může být taková, že často (nebo když se nepoužívá vždy) určí stejnou hodnotu. Pak nastupuje rozhodovací pravidlo.

Cíle plánovacích algoritmů

Každý algoritmus nutně upřednostňuje nějakou třídu úloh na úkor ostatních.

- dávkové systémy
 - dlouhý čas na CPU, omezí se přepínání úloh
- interaktivní systémy
 - Interakci s uživatelem, tj. I/O úlohy
- systémy reálného času
 - Dodržení deadlines

Společné cíle

- spravedlivost
 - srovnatelné procesy srovnatelně obsloužené
- vynucovat stanovená pravidla
- efektivně využít všechny části systému
- nízká režie plánování

Dávkové systémy (!)

■ průchodnost (throughput)

- počet úloh dokončených za časovou jednotku

■ průměrná doba obrátky (turnaround time)

- průměrná doba od zadání úlohy do systému do dokončení úlohy

■ využití CPU

Průchodnost a průměrná doba obrátky jsou různé údaje ! Někdy snaha vylepšit jednu hodnotu může zhoršit druhou z nich.

Dávkové systémy

- maximalizace průchodnosti nemusí nutně minimalizovat dobu obrátky
- modelový příklad:
 - dlouhé úlohy následované krátkými
 - upřednostňování krátkých
 - bude tedy dobrá průchodnost
 - dlouhé úlohy se nevykonají
 - doba obrátky bude nekonečná

Interaktivní systémy

Chceme:

Minimalizaci doby odpovědi

ale je třeba dbát na:

efektivitu – drahé přepínání mezi procesy

Realtimové systémy

■ Dodržení deadlines

- termín, do kdy musí být daný proces obsloužen

■ Předvídatelnost

- Některé akce pravidelné, periodické (např. generování zvuku)

Př. Obsluha GSM části telefonu

Plánování úloh v dávkových systémech

- FCFS (First Come First Served)
- SJF (Shortest Job First)
- SRT (Shortest Remaining Time)
 - Preemptivní varianta SJF
- Multilevel Feedback

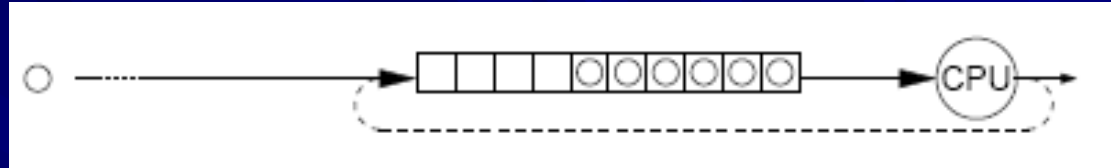
FCFS (First Come First Served)

■ FIFO

■ Nepreemptivní FIFO

■ Základní varianta

- Nově příchozí na konec fronty připravených
- Úloha běží dokud neskončí, poté vybrána další ve frontě (viz 1.)



■ Co když úloha provádí I/O operaci?

1. Zablokována, CPU se nevyužívá (základní varianta)
2. Do stavu blokováný, běží jiná úloha po dokončení I/O zařazena na **konec** fronty připravených (častá varianta !!!)
- I/O vázané úlohy znevýhodněny před výpočetně vázanými
3. Další možná modifikace → po dokončení I/O na začátek fronty připravených

Poznámka

V následujících příkladech předpokládáme, že se uvažovaná úloha skládá jen z **1 dlouhého CPU burstu**, tj. nečeká na I/O, tj. jen počítá

- aby šlo lépe nakreslit diagramy

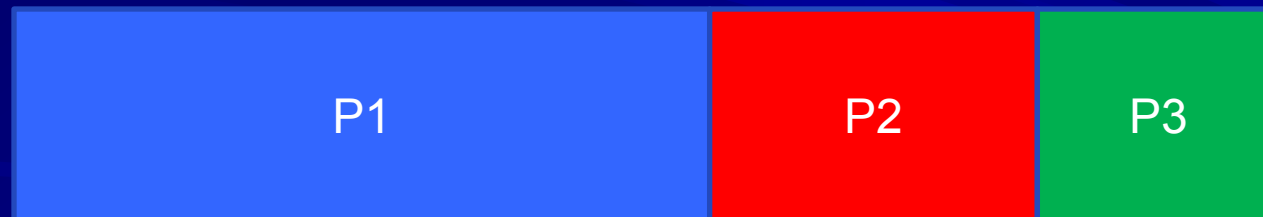
U dávkových úloh můžeme odhadnout dobu provádění úlohy (třeba z dřívějších běhů) a tento čas může být velmi významný pro rozhodnutí plánovače.

FCFS příklad

V čase nula budou v systému procesy P1, P2, P3 přišlé v tomto pořadí.

proces	Doba trvání (s)
P1	15
P2	5
P3	4

doba obrátky:
odešel
-
přišel



0

15

20

24

tedy:

P1: $15 - 0 = 15$

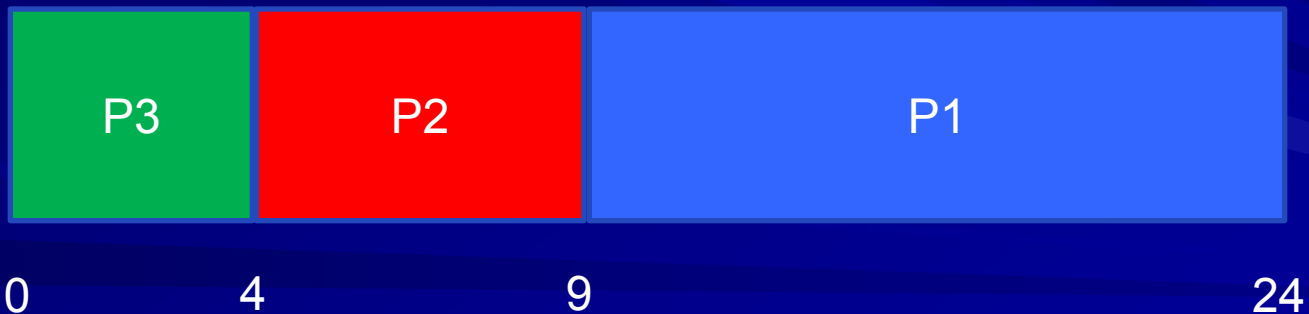
P2: $20 - 0 = 20$

P3: $24 - 0 = 24$

průměrná doba obrátky: $(15+20+24) / 3 = 19,666$

SJF (Shortest Job First)

- Nejkratší úloha jako první
- Předpoklad – známe přibližně dobu trvání úloh
- **Nepreemptivní**
 - Jedna fronta příchozích úloh
 - Plánovač vybere vždy úlohu s nejkratší dobou běhu
- Optimalizuje dobu obrátky



průměrná doba obrátky: $(4+9+24) / 3 = 12,3$

Výpočet průměrné doba obrátky

Do systému přijdou v čase nula úlohy A,B,C,D s dobou běhu: 8, 4, 4, 4 minut. Úlohy budou tvořeny jedním CPU burstem.

FCFS

- Spustí v pořadí A, B, C, D dle strategie FCFS
- Doba obrátky:
 - A 8 minut
 - B $8+4 = 12$ minut
 - C $8+4+4 = 16$ minut
 - D $8+4+4 +4 = 20$ minut
- Průměrná doba obrátky:
 $(8+12+16+20) / 4 = 14$ minut

Výpočet průměrné doby obrátky

■ strategie plánovače **SJF**

■ V pořadí B, C, D, A – od nejkratší

- B 4 minuty
- C $4+4 = 8$ minut
- D $4+4+4 = 12$ minut
- A $4+4+4+8 = 20$ minut

■ Průměrná doba obrátky

$$(4+8+12+20) / 4 = 11 \text{ minut}$$

■ Průměrná doba obrátky je v tomto případě lepší

SRT (Shortest Remaining Time)

- Úlohy můžou přicházet **kdykoliv** (*nejen v čase nula*)
- **Preemptivní** (!! možný přechod běžící - připravený)
 - Plánovač vždy vybere úlohu, jejíž **zbývající** doba běhu je nejkratší (!!!)
- Př. KDY dojde k preempci:
Právě prováděné úloze zbývá 10 minut, do systému právě teď přijde úloha s dobou běhu 1 minutu – systém **prováděnou** úlohu **pozastaví** a nechá běžet novou úlohu
 - i když by byla tvořena jen CPU burstem
- Možnost vyhladovění dlouhých úloh (!) => neustále předbíhány krátkými

SRT příklad

Čas příchodu	Název úlohy	Doba úlohy (s)
0	P1	7
0	P2	5
3	P3	1

V čase 0 máme na výběr P1, P2. Naplánujeme P2 s kratší dobou běhu

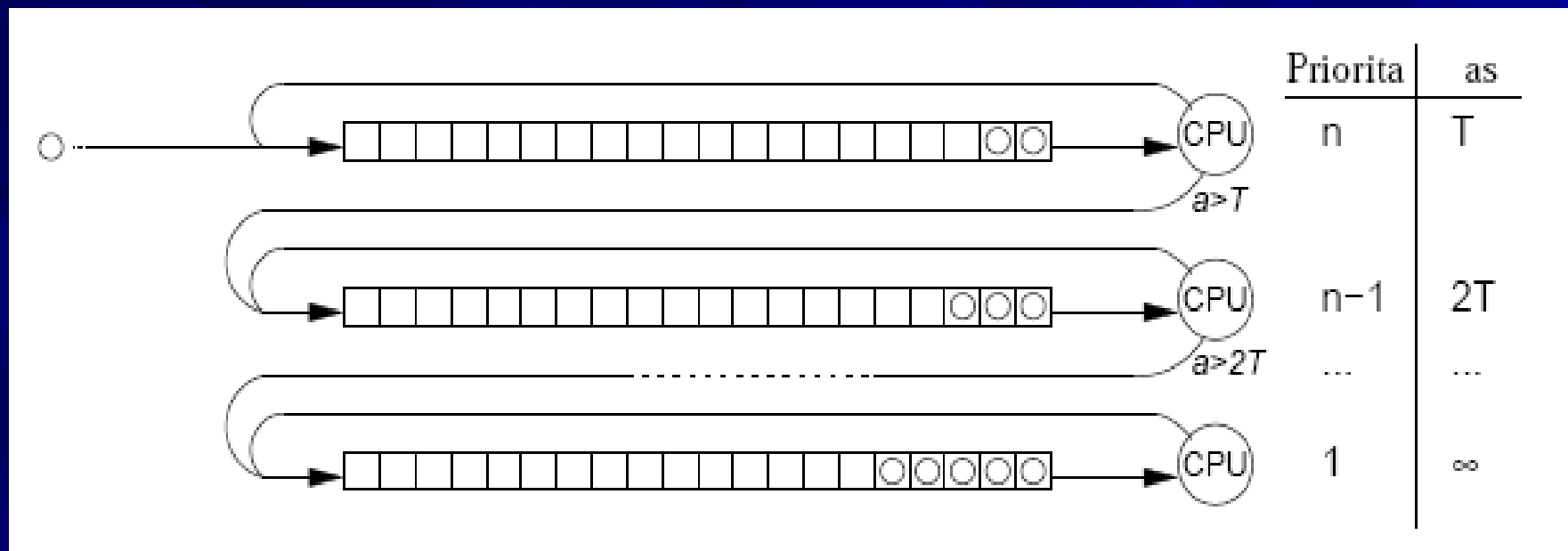
V čase 3 **přijde** do systému nová úloha. Zkontrolujeme zbývající doby běhu úloh: P1(7), P2 (2), P3(1). Naplánujeme P3.

Jakmile **skončí** P3, naplánujeme P2, až doběhne, naplánujeme P1, ...

Multilevel feedback

- **N** prioritních úrovní
- Každá úroveň má svojí frontu úloh
- Úloha vstoupí do systému do fronty s **nejvyšší** prioritou (!)
- Na každé prioritní úrovni
 - Stanovení maximum času CPU, který může úloha obdržet
 - Např.: T na úrovni n , $2T$ na úrovni $n-1$ atd.
 - Pokud úloha překročí tento limit, její priorita se sníží
 - Na nejnižší prioritní úrovni může úloha běžet neustále nebo lze překročení určitého času považovat za chybu
- Proces obsluhuje nejvyšší neprázdnou frontu (!!)

Multilevel feedback



Výhoda – rozlišuje mezi I/O-vázanými a CPU-vázanými úlohami

Upřednostňuje I/O vázané – déle se drží ve vysokých frontách

Shrnutí – dávkové systémy

algoritmus	Rozh. mód	Prioritní funkce	Rozh. pravidlo
FCFS	Nepreemptivní	$P(r) = r$	Náhodně
SJF	Nepreemptivní	$P(t) = -t$	Náhodně
SRT	Preemptivní (při příchodu úlohy)	$P(a,t) = a-t$	FIFO nebo náhodně
MLF	nepreemptivní	Viz popis ☺	FIFO v rámci fronty

r celkový čas strávený úlohou v systému
t předpokládaná délka běhu úlohy
a čas strávený během úlohy v systému