KIV/ZOS 2003/2004
Přednáška 6

Problém čtenářů a písařů
---

* Courtois et al. 1971, angl. ”Readers and Writers Problem”
* modeluje přístup do databáze

* představme si velkou databázi (např. rezervace místenek), množina procesů se pokouší souběžně číst a zapisovat (čtenáři a písaři)
 - více požadavků čtení - akceptovatelné
 - při zápisu nesmí nikdo jiný přistupovat, ani žádní čtenáři
 - jak naprogramujeme čtenáře a písaře?

```
var
   m=1, w=1: semaphore;  { semafory mutex, přístup pro zápis }
   rc = 0: integer;      { počet čtenářů }

procedure reader;
begin
   P(m);
   rc := rc + 1;
   if rc = 1 then P(w);
   V(m);
   // čti
   P(m);
   rc := rc - 1;
   if rc=0 then V(w);
   V(m)
end;

procedure writer;
begin
   P(w);
   // zapisuj
   V(w)
end;
```

Jak to pracuje:

 - první čtenář, který dostane přístup do databáze, provede P(w), další pouze zvětšují čítač rc
 - po skončení čtenáři zmenšují rc, poslední provede V(w)
 - semafor w zabrání vstupu písaře, pokud jsou čtenáři
 - semafor w také zabrání vstupu čtenářům, je-li písař (P(w) zabrání prvnímu, P(m) ostatním)

* v tomto řešení mají čtenáři přednost před písaři; pokud je nějaký čtenář, písaři musejí čekat až všichni čtenáři skončí
* existuje řešení, kde přednost mají písaři - je ale složitější, proto ho uvedu pouze pro zajímavost o něco níže

* řešení problému čtenářů a písařů se používají pro implementaci zámků v operačních a databázových systémech
 - potřebuje-li proces přistupovat k souboru nebo záznamu v databázi, může soubor nebo záznam uzamknout
 - zámky jsou dvou typů:
  . zámek pro zápis neboli výhradní zámek - nikdo další nesmí přistupovat
  . zámek pro čtení neboli sdílený zámek - o sdílený zámek mohou žádat další procesy)
 - operační systémy - např. v systémech Linux a UNIX je možné zamykat části souboru fcí fcntl(fd, F_SETLK, struct flock) atd.
 - databázové systémy: např. s každým záznamem databáze sdružen zámek, systém poskytuje fce typu:
  . db_lock_r(x)   - uzamčení záznamu x pro čtení,
  . db_lock_w(x)   - uzamčení záznamu x pro zápis,
  . db_unclock_r(x) a db_unclock_w(x) - odemčení záznamu x

```
Nyní ještě pro zajímavost uvedu řešení problému čtenářů a písařů s předností
písařů (jakmile alespoň jeden písař požaduje přístup k položce, čtenáři
nebudou moci vstupovat); řešení je přestrukturované do podoby výše zmíněných
procedur.

type zámek = 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; // je-li písař, blokuje 1. čtenáře
                rdel: semaphore := 1; // blokování ostatních čtenářů
             end;

procedure db_lock_w(var x: zámek); // uzamčení záznamu pro zápis
begin
    P(x.mutw);
    x.wc:=x.wc+1;
    if x.wc=1 then P(x.rsem);
    V(x.mutw);
    P(x.wsem);
end;

procedure db_unlock_w(var x: zámek); // odemčení záznamu uzamčeného pro zápis
begin
    V(x.wsem);
    P(x.mutw);
    x.wc:=x.wc-1;
    if x.wc=0 then V(x.rsem);
    V(x.mutw)
end;

procedure db_lock_r(var x: zámek); // uzamčení záznamu pro čtení
begin
    P(x.rdel);
    P(x.rsem);
    P(x.mutr);
    x.rc:=x.rc+1;
    if x.rc=1 then P(x.wsem);
    V(x.mutr);
    V(x.rsem);
    V(x.rdel)
end;

procedure db_unlock_r(var x: zámek); // odemčení záznamu uzamčeného pro čtení
begin
    P(x.mutr);
    x.rc:=x.rc-1;
    if x.rc=0 then V(x.wsem);
    V(x.mutr)
end;
```

```
Poznámka pro zajímavost (další problémy meziprocesové komunikace)

* existuje mnoho dalších zajímavých problémů, kterými se na
  přednáškách nebudeme zabývat z časových důvodů, např.:

 - problém spícího holiče (autor neznámý)
 - problém populárního pekaře (Lamport 1974)
 - plánovač hlavičky disku (uvedeme si v příslušné části přednášek)

* některé z těchto úloh možná ponecháme na cvičení.

[]

Plánování procesů
=================

Opakování:

* základní stavy procesu:
 - běžící - běží na CPU
 - připraven - čeká na procesor
 - blokován - nemůže pokračovat dokud nedostane zdroj nebo zprávu.
```

* v předchozích příkladech často situace, kdy jeden nebo více procesů
  logicky připravených k běhu
* pokud je méně procesorů než připravených procesů, OS musí vybrat
  který spustí jako první
* část OS, která to rozhoduje se nazývá plánovač procesů (scheduler),
  použitý algoritmus se nazývá plánovací algoritmus (scheduling
  algorithm)

* v době dávkových systémů to bylo jednoduché - např. spustit další
  zadanou úlohu, nechat jí běžet do konce
* v době systémů se sdílením času složitější - uživatelské interaktivní
  procesy, procesy běžící na pozadí apod. (např. v NT: můžete se zároveň
  dívat na film a psát dokument)
* některé současné systémy kombinace dávkových systémů a sdílení času,
  plánovač musí určit jestli čas dostane dávka nebo interaktivní proces

* například pokud uživatel současně pošle poštu a zavře okno:
 - pokud zavírání okna trvá 2 sec zatímco se odesílá pošta, uživatel bude
   systém vnímat jako extrémně pomalý
 - pokud se zavře okno ihned a posílání emailu se zdrží o 2 sec,
   uživatel to ani nepostřehne => přednost by měl dostat interaktivní
   proces.

Základní terminologie

Plánovače lze popsat pomocí následujících tří charakteristik:

1. rozhodovací mód
2. prioritní fce
3. rozhodovací pravidla

V okamžicích určených {rozhodovacím módem} plánovač pro všechny aktivní
procesy v systému vyhodnotí {prioritní fci}; výsledkem je určení aktuální
priority procesů. Připravený proces s nejvyšší prioritou dostane CPU; pokud
je více procesů se stejnou prioritou, přijde na řadu {rozhodovací pravidlo},
které rozhodne mezi procesy se stejnou prioritou.

Rozhodovací mód
....

* rozhodovací mód specifikuje časové okamžiky ve kterých jsou
  vyhodnoceny priority procesu a je vybrán proces pro běh na CPU
* dva základní typy:
 - proces běží až do konce nebo dokud se sám nezablokuje = nepreemptivní
   plánování
 - běžící proces může být pozastaven = preemptivní plánování

* nepreemptivní plánování
 - proces může běžet dokud je to logicky možné (plánování se děje pouze pokud
   proces skončí nebo pokud se zablokuje, tj. proces se CPU vzdává
   dobrovolně)
 - jednoduchá implementace, vhodné např. pro dávkové nebo některé
   specializované systémy
 - není vhodné pro interaktivní a RT systémy

* preemptivní plánování = CPU může být procesu odebrán
 - přeplánování přijde-li nový proces (v dávkových systémech)
 - periodické přeplánování - ”kvantově orientované” plánovací algoritmy
   (v interaktivních systémech)
 - jiná strategie - např. preempce kdykoli je priorita připraveného procesu
   větší než priorita běžícího (systémy reálného času)

Preemptivní plánování je dražší o čas přepínání procesů a o přídavnou
logiku plánovače.

Prioritní funkce
.....

* prioritní funkce je funkce parametrů procesů a systémových parametrů, určující prioritu procesu v systému
 - externí priority (rozlišení mezi třídami uživatelů a systémovými procesy)
 - priority odvozené z chování procesu v systému apod.
* priorita se může odvozovat například od:
 - času po který proces používal CPU
 - aktuálního zatížení systému
 - paměťových požadavků procesu
 - času který proces strávil v systému
 - celkové doby provádění úlohy (v některých dávkových systémech se pro programy specifikuje maximální doba provádění, aby nedošlo k nekonečnému provádění chybné úlohy)
 - urgence (v RT systémech).

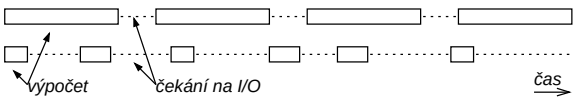
Rozhodovací pravidlo
.....

* řeší konflikty mezi procesy se stejnou prioritou
* při malé pravděpodobnosti stejné priority (díky prioritní fci) je možný náhodný výběr
* při vysoké pravděpodobnosti
 - v kvantově orientovaných preemptivních plánovacích algoritmech cyklické přidělování kvanta
 - ve většině ostatních případů chronologický výběr (FIFO)

Výpočetně vázané a vstupně/výstupně vázané procesy
.....

* typický proces prokládá výpočet (využití CPU) s čekáním na vstup nebo diskovou operací (tj. čekáním na dokončení vstupu/výstupu)
* pro některé I/O - CPU zadá požadavek I/O zařízení, do dokončení operace se se nemusí účastnit (vlastní přenos provádí DMA)
* někdy je výhodně rozlišit 2 kategorie procesů:
 - výpočetně vázané (compute-bound, CPU-bound) - téměř veškerý čas spotřebovávají výpočtem, tj. využitím CPU
 - vstupně/výstupně vázané (I/O-bound) - většinu času čekají na dokončení I/O
* výpočetně vázané procesy typicky vyžadují CPU na delší dobu, zatímco vstupně/výstupně vázané využívají CPU krátkou dobu a pak zase zadají I/O požadavek

a) výpočetně vázaná úloha
b) vstupně/výstupně vázaná úloha



Cíle plánovacích algoritmů
---

* problém - každý plánovací algoritmus nutně upřednostňuje nějakou třídu úloh na úkor ostatních (koneckonců, čas CPU je omezený zdroj, pokud ho dám jednomu procesu, nemůžu ho dát jinému)

Např. pro minimalizaci doby odpovědi pro interaktivní uživatele
uživatele by systém neměl spouštět dávkové úlohy (v době kdy CPU
potřebují interaktivní uživatelé), což je v rozporu s požadavky na dobu
obrátky a průchodnost (které jsou důležité pro zadavatele dávkových
úloh).

* co má plánovač optimalizovat se bude lišit podle typu OS, rozlišíme:
 1. dávkové systémy
 2. interaktivní systémy
 3. systémy reálného času

* dávkové systémy:
 - nejsou interaktivní uživatelé => každý proces může dostat dlouhý čas
 - omezí se přepínání úloh, vyšší efektivita
* interaktivní systémy:
 - nutné přepínání mezi procesy
* systémy reálného času:
 - někdy není nutné procesům odebírat CPU, protože procesy vědí, že nemohou běžet dlouhou dobu - procesy po spuštění rychle obslouží požadavek a zablokují se.

* společné cíle plánovacích algoritmů:
 - spravedlivost: srovnatelné procesy by měly být srovnatelně obslouženy (měly by obdržet srovnatelný díl času CPU apod.)
 - vynucovat stanovená pravidla (např. pokud mají mít interaktivní procesy přednost před dávkovými úlohami)
 - zaměstnávat všechny části systému: pokud je možný I/O bez účasti CPU, měl by systém po zadání I/O požadavku např. přepnout na jiný proces
 - efektivita: režie plánování by měla být co nejnižší.

Ostatní cíle se liší podle kategorie OS:

* dávkové systémy - obvykle se používají 3 metriky výkonnosti systému:
 - průchodnost (throughput) - počet úloh za časovou jednotku, které je systém dokončit
 - průměrná doba obrátky (turnaround time) - průměrná doba od zadání úlohy do systému do dokončení úlohy
 - využití CPU - kolik procent času je CPU zaměstnána
* možné cíle
 - maximální průchodnost - počet zpracovaných úloh za hodinu
 - minimální průměrná doba obrátky - snaha minimalizovat dobu, po kterou dávkoví uživatelé čekají na výstup
 - využití CPU - není prakticky dobrá metrika, důležitý je výsledek, tj. zpracované úlohy

Poznámka:

  Plánovací algoritmus maximalizující průchodnost nemusí nutně minimalizovat dobu obrátky.

  Např. pokud budeme upřednostňovat krátké úlohy, a do systému budou zadány dlouhé úlohy následované krátkými, můžeme obdržet dobrou průchodnost (za časovou jednotku se vykoná mnoho krátkých úloh), průměrná doba obrátky bude ale nekonečná (dlouhé úlohy nebudou nikdy spuštěny).

[]

* interaktivní systémy - důležitá je doba odpovědi, tj. čas od požadavku (zadání příkazu, příchod události) do obdržení výsledku
* plánovač by měl minimalizovat dobu odpovědi pro interaktivní procesy (ale pozor: přepínání mezi procesy je drahé - tj. je třeba brát v úvahu i efektivitu)

* systémy reálného času:
 - termíny (deadlines) které musejí nebo alespoň mají být dodrženy
 - v některých RT systémech důležitá předvídatelnost - např. plánování procesu generujícího zvuk musí být předvídatelné a pravidelné.

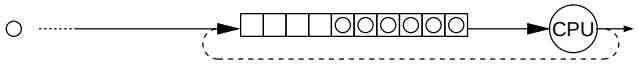
Od obecných charakteristik přejdeme ke konkrétním plánovacím algoritmům.

Plánování úloh v dávkových systémech

First-Come First-Served (FCFS)
...

* někdy používán také název FIFO (First-in First-out)
* zpracování podle času příchodu úlohy, ”kdo dřív přijde ten dřív mele”

* nepreemptivní FCFS je nejjednodušší plánovací algoritmus
* jedna fronta příchozích úloh, nově příchozí úlohy se řadí na konec fronty
* úloha může běžet dokud neskončí, po dokončení je vybrána a spuštěna další úloha ve frontě



* výhody:
 - lze ho jednoduše naprogramovat i snadno mu porozumět
 - je spravedlivý

* základní verze algoritmu předpokládá, že po dobu I/O je úloha zablokována a CPU se nevyužívá
* možné řešení: pokud úloha provádí I/O, zablokuje se; po dokončení I/O je zařazena na konec fronty (jako nově příchozí)
* nevýhoda (za cenu jednoduchosti) - vstupně/výstupně vázané úlohy jsou silně znevýhodněny před výpočetně vázanými

Shortest Job First (SJF)
.....

* strategie SJF také nazývána Shortest Job Next (SJN)
* v češtině ”nejkratší úloha první”

* předpoklad, že je předem známá doba vykonávání úlohy (u dávkových úloh může být známo, např. pokud se počítají výplaty každý měsíc, je známo že to trvá 2 hod.)
* nepreemptivní, jedna fronta příchozích úloh, plánovač vybere vždy úlohu s nejkratší dobou běhu
* dokazatelně optimální vzhledem k době obrátky

Příklad:

 Do systému přijdou úlohy A, B, C a D s dobou běhu 8, 4, 4, 4 minuty.

 Pokud spustíme v pořadí A, B, C, D (podle strategie FCFS) doba obrátky bude:
   A ... 8 minut
   B ... 8+4 = 12 min
   C ... 8+4+4 = 16 min
   D ... 8+4+4+4 = 20 min => průměrná doba obrátky (8+12+16+20)/4=14 min

 Pokud spustíme v pořadí SJF, tj. B, C, D, A:
   B ... 4 min
   C ... 4+4 = 8 min
   D ... 4+4+4 = 12 min
   A ... 4+4+4+8 = 20 min => průměrná doba obrátky (4+8+12+20)/4 = 11 min

 (Tj. průměrná doba obrátky je lepší při použití strategie SJF.)

Poznámka (optimalita strategie SJF vzhledem k průměrné době obrátky)

* představme si, že máme 4 úlohy s dobou běhu a, b, c a d
* první úloha skončí v čase a, druhá a+b, třetí a+b+c ...
* průměrná doba obrátky bude (4a + 3b + 2c + d)/4
* tj. první úloha ”přispívá” k době obrátky nejvíce, poslední nejméně
* proto je SJF optimální vzhledem k době obrátky

[]

Poznámka (předpoklad optimality strategie SJF)

 Pokud mohou úlohy přicházet kdykoli, nemusí být nepreemptivní SJF optimální.

- předpokládejme, že úlohy A, B, C, D, E mají dobu běhu 2, 4, 1, 1, 1
- doba příchodu do systému bude 0, 0, 3, 3, 3
- algoritmus SJF by volil pořadí A, B, C, D, E - prům. doba 4.6 min
- při běhu v pořadí B, C, D, E, A - průměrná doba obrátky 4.4

[]

Shortest Remaining Time (SRT)
.....

* preemptivní varianta SJF, řeší výše zmíněný problém
* plánovač vždy vybere úlohu, jejíž zbývající doba běhu je nejkratší

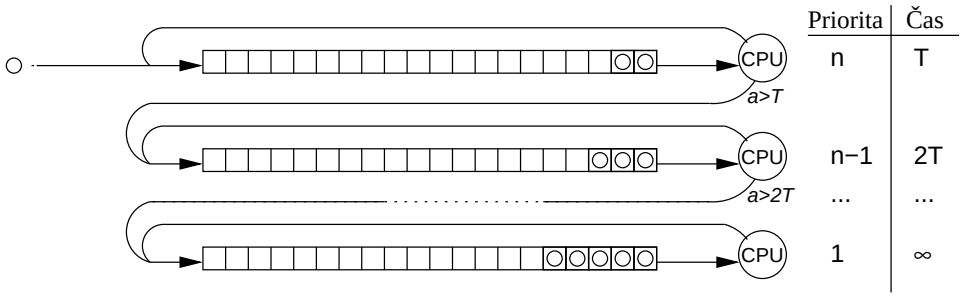
Např. právě prováděné úloze to bude trvat 10 min, do systému přijde úloha trvající 1 minutu - systém prováděnou úlohu pozastaví a nechá běžet novou úlohu.

* dobrá služba pro nové krátké úlohy
* problém: možnost vyhladovění dlouhých úloh (řešení si ukážeme později)

Multilevel Feedback
.....

* n rozdílných prioritních úrovní, každá úroveň má svou frontu úloh
* úloha vstoupí do systému s nejvyšší prioritou
* na každé prioritní úrovni je stanoveno maximum času CPU, které může úloha na této úrovni obdržet
  - např. T na prioritní úrovni n, 2T na prioritní úrovni n-1, atd.
  - pokud úloha překročí limit spotřebu času CPU, její priorita se sníží
  - na nejnižší prioritní úrovni může proces běžet neustále nebo může být překročení času považováno za chybu
* procesor obsluhuje nejvyšší neprázdnou frontu

* výhoda - rozlišuje mezi I/O vázanými a CPU-vázanými úlohami; upřednostňuje I/O vázané



Poznámka (souhrnná charakteristika uvedených algoritmů)

Výše uvedené algoritmy pro plánování v dávkových systémech bychom mohli popsat podle rozhodovacího módu, prioritní fce a rozhodovacího pravidla následující tabulkou:

algoritmus	rozhodovací mód	prioritní fce	rozhodovací pravidlo
FCFS	nepreemptivní	P(r)=r	náhodně
SJF	nepreemptivní	P(t)=-t	náhodně
SRT	preemptivní (při příchodu)	P(a, t)=a-t	FIFO nebo náhodně
MLF	nepreemptivní	viz popis	FIFO v rámci fronty

V prioritní fci je r ... celkový čas strávený úlohou v systému
t ... předpokládaná délka běhu úlohy
a ... čas strávený během úlohy v systému.

[]

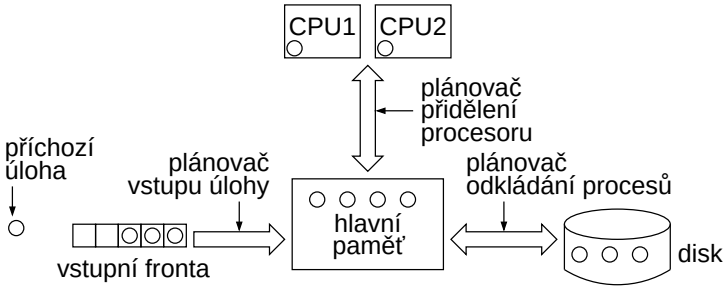
Poznámka (plánování probíhá na více úrovních)
.....

Dávkové systémy dovolují plánování na 3 různých úrovních:

* úlohy jsou ve vstupní frontě dokud je plánovací algoritmus nevybere - pak vstoupí do systému
* po příchodu do systému může být pro úlohu vytvořen proces; pokud je procesů příliš, bude třeba některé odložit na disk
 - přesun procesů mezi diskem a pamětí je drahý, proto procesy musejí být nějakou dobu odloženy před opětovným zavedením do paměti
* jeden z připravených procesů v paměti dostane CPU

=> ve skutečných dávkových OS můžeme najít tři různé typy plánovače:

* plánovač příchodu úlohy do systému (admission scheduler) - určuje, které procesy vstoupí do systému ze vstupní fronty
* plánovač odkládání procesů (memory scheduler) - určuje, které procesy budou v paměti a které budou odloženy na disku
* plánovač přidělení procesoru (CPU scheduler) - přiděluje procesor procesům v paměti



[]

❇