

Principy počítačů a operačních systémů

Operační systémy
Procesy, plánování, synchronizace

Zimní semestr 2007/2008

Co je to proces?

Proces

- běžící instance programu (posloupnost instrukcí)
- zapouzdřuje prostředí, ve kterém program běží

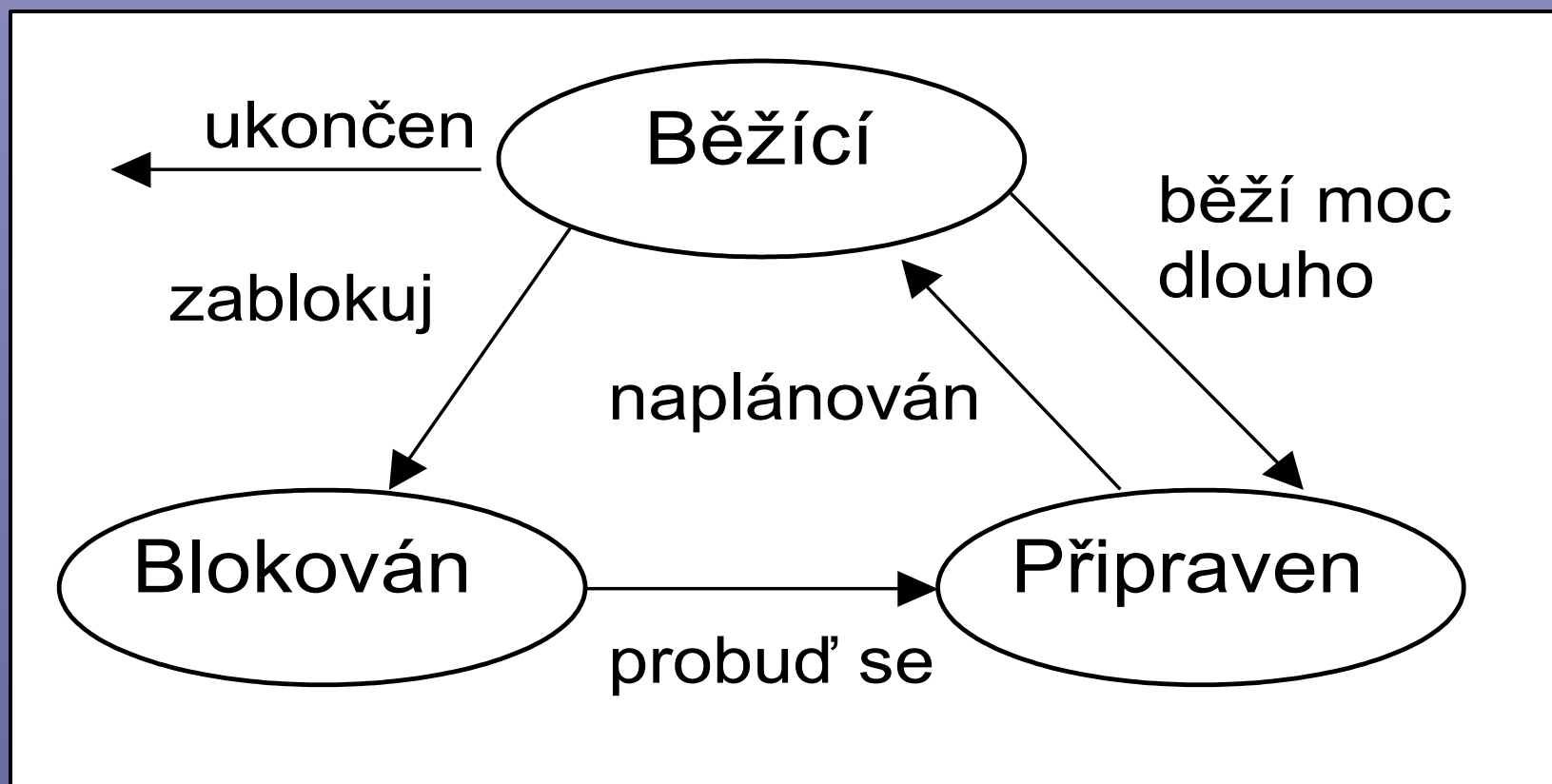
Proces kontext výpočtu

- kontext procesoru (hodnoty registrů)
 - ♦ PSW, IR, PC, SP, general purpose registers
- kontext paměti (adresový prostor, obsah paměti)
 - ♦ text, data, heap, stack
- kontext prostředí (struktury operačního systému)
 - ♦ otevřené soubory a komunikační kanály, uživatelský terminál

Co je to proces?

Stavy procesu

- ke změnám dochází za běhu v důsledku událostí



Co je to proces?

Události v systému

- synchronní – vznikají v důsledku běhu procesu
 - ♦ traps, speciální instrukce (např. pro volání OS)
 - ♦ exceptions, nesprávné chování procesu
- asynchronní – vznikají vně systému
 - ♦ žádost zařízení o přerušování

Obsluha přerušování

- procesor předá řízení OS, uloží se kontext CPU
- analyzuje se příčina přerušování, vyvolá se obsluha
- obsluha přerušování, obnovení kontextu CPU
- návrat do přerušované aplikace

Co je to proces?

Vlákna

- abstrakce toku výpočtu – aktivita
 - ♦ odpovídá vykonávání instrukcí programu
- vícevláknový proces ~ víceprocesorový stroj
 - ♦ sdílený přístup do paměti a k prostředkům
- v rámci procesu vlákna sdílí
 - ♦ kontext paměti, kontext prostředí
- vlákna nesdílí
 - ♦ kontext procesoru, zásobník
 - ♦ zásobník uchovává historii běhu výpočtu (volání funkcí)

Co je to proces?

Použití vláken

- strukturování programu
 - ♦ samostatná vlákna pro zpracování požadavků na serveru
- implementace asynchroních I/O operací
 - ♦ překrytí výpočetních operací s I/O operacemi

Implementace vláken

- přepínání kontextu procesoru
 - ♦ na úrovni operačního systému
 - ♦ na úrovni uživatelského procesu
- pro využití více procesorů nutná podpora v jádře

Podpora více procesů

Terminologie

- multitasking
 - ♦ více procesů sdílí stejný procesor
- multiprogramming
 - ♦ systém spravuje více procesů najednou, spouští je na jednom nebo více procesorech
- multiprocessing
 - ♦ použití více procesorů pro běh procesů

Podpora více procesů

Motivace pro více procesů

- zlepšení odezvy systému
 - ♦ dlouhé odezvy při dávkovém zpracování úloh
 - ♦ při sdílení procesoru by kratší úlohy byly hotovy dříve
- zlepšení využití systému
 - ♦ aplikace typicky něco počítá, nebo čeká na data
 - doba čekání na data z disku v řádu *ms*
 - ♦ během čekání mohou jiné aplikace “něco počítat”
 - ♦ zlepšuje odezvu – jiný proces může postupovat vpřed
- současný běh více procesů
 - ♦ přepínání mezi více aplikacemi, spojování procesů pro práci na stejném problému, ...

Plánování procesů a vláken

Plánování

- rozhodnout, který proces (vlákno) poběží
- rozhodnutí typicky optimalizuje nějakou metriku

Typické metriky

- doba odezvy (response time, turnaround)
 - ♦ do ukončení procesu, do první odezvy, ...
- propustnost (throughput)
 - ♦ počet dokončených úloh za jednotku času
- využití procesoru (utilization)
- spravedlnost (fairness)

Plánování procesů a vláken

Off-line plánování

- předpoklady
 - ♦ všechny procesy jsou k dispozici od začátku a žádné již nepřibydou
 - ♦ o všech procesech je známo jak dlouho poběží
- výsledky
 - ♦ dávkové zpracování s ohledem na cílové metriky
 - ♦ běh procesů není nutné přerušovat, plán je optimální
- problém
 - ♦ předpoklady jsou málo realistické
 - ♦ poskytují teoretické meze, pokud bychom měli požadované informace

Základní off-line algoritmy

FCFS – First Come First Served

- základní algoritmus dávkového zpracování
 - ♦ procesy plánovány v pořadí, v jakém přicházejí
 - ♦ procesy běží dokud neskončí

SJF – Shortest Job First

- kratší úlohy plánovány přednostně
 - ♦ minimalizuje průměrnou dobu odezvy

Plánování procesů a vláken

On-line plánování

- předpoklady
 - ♦ procesy se objevují libovolně a neočekávaně
 - ♦ doba běhu procesů je neznámá
- kritéria plánování
 - ♦ vázanost na CPU nebo I/O, interaktivní/dávkový proces
 - ♦ chování procesu v minulosti, výpadky stránek, priorita
- preemptivní plánování
 - ♦ potřebuje podporu HW (časovač)
 - ♦ možnost měnit plán na základě nových informací
 - ♦ **context switch** – přepnutí na jiný proces / vlákno

Základní on-line algoritmy

FCFS – First Come First Served

- procesy plánovány v pořadí, v jakém se objevují
- nepreemptivní

Připravené procesy

C

B

A



CPU

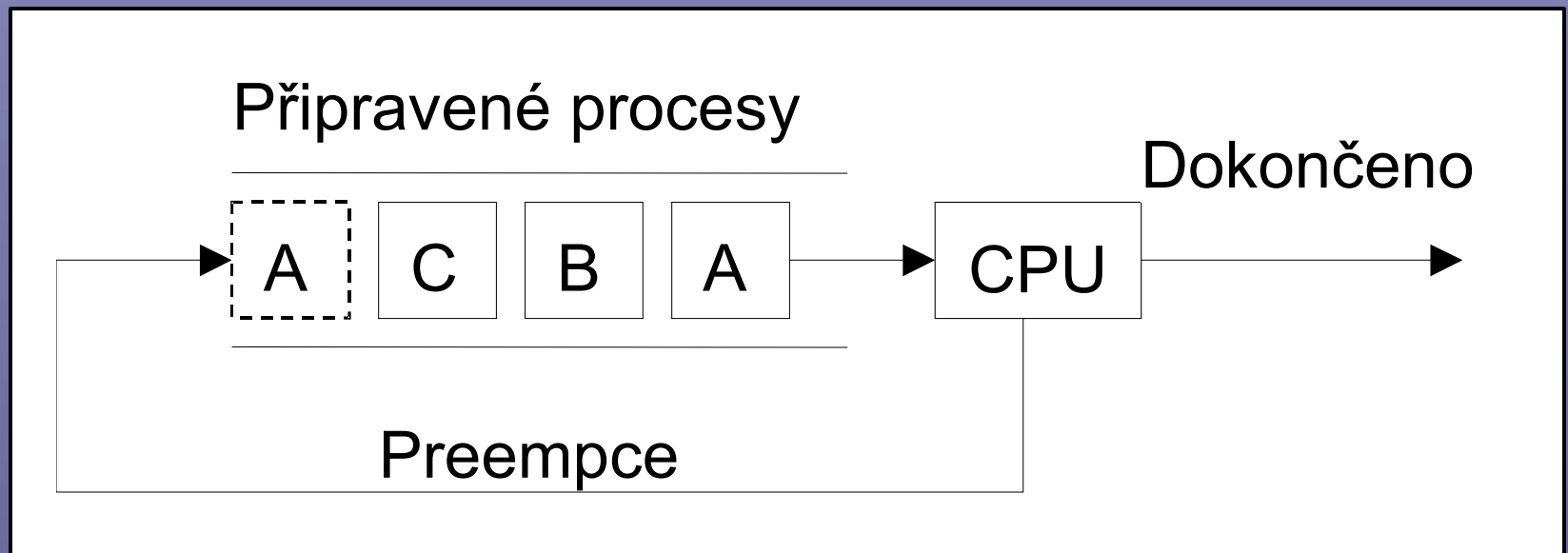
Dokončeno



Základní on-line algoritmy

RR – Round Robin

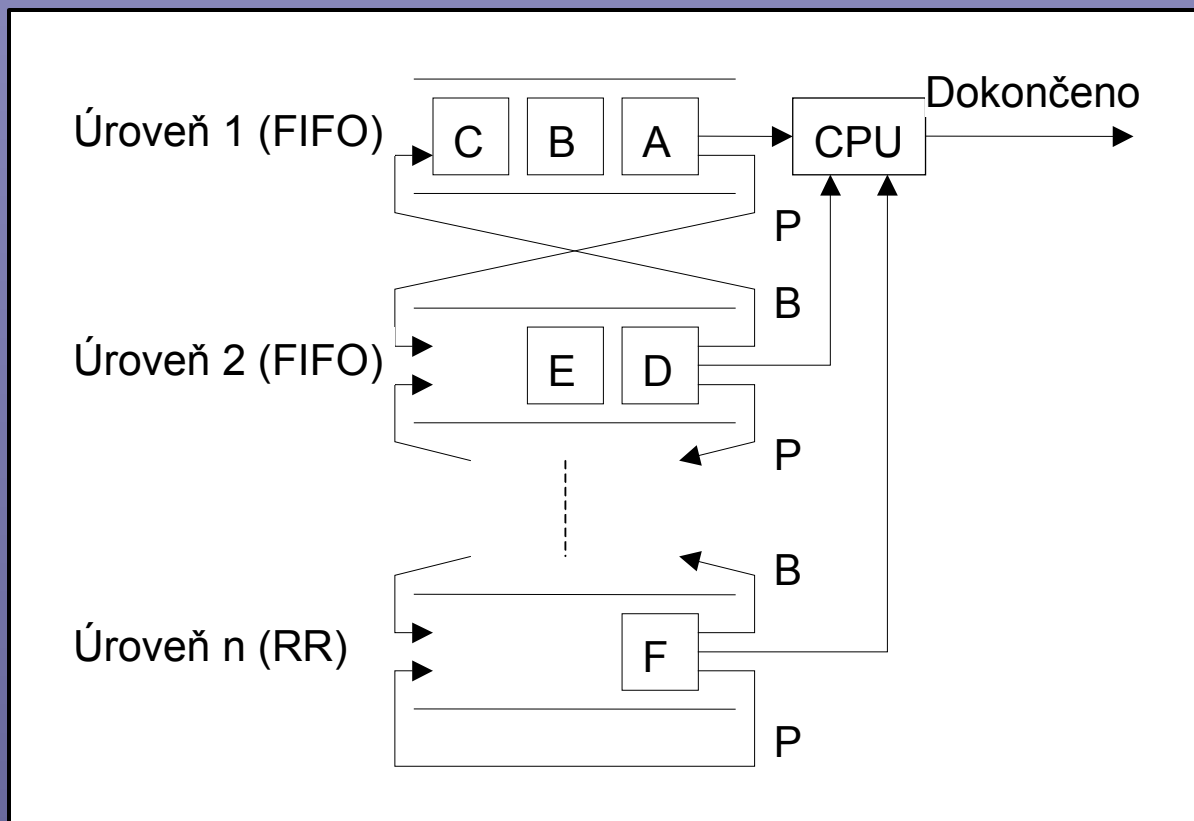
- časové kvantum (time slice)
 - ♦ aproximace sdílení procesoru
- preemptivní plánování, fairness



Základní on-line algoritmy

Prioritní s více frontami a zpětnou vazbou

- reaguje na chování procesů (kvantum, priorita)
 - ♦ rozlišuje interaktivní a dávkové procesy



Plánování procesů a vláken

Plánování pro více procesorů

- každý procesor má vlastní “ready queue”
- vyvažování zátěže, zohlednění afinity

Plánování pro real-time systémy

- aplikace řízené událostmi
 - ♦ procesy ~ obsluha událostí, příjem a zpracování dat, generování výstupu
- běh omezen reálným časem dokončení – deadline
- hard real-time, soft real-time
- často off-line plánování

Interakce mezi procesy

Přístup ke sdíleným datům

- datové struktury OS nebo vícevláknového programu

Zablokování na sdílených prostředcích

- přístup k prostředkům spravovaným OS
- vyplývá z funkce OS jako správce prostředků

Komunikace mezi procesy

- spolupráce procesů v rámci paralelní/distribuované aplikace

Přístup ke sdíleným datům

Souběžné vs. paralelní zpracování

- paralelní provádění (parallel execution)
 - ♦ více činností na různých místech současně
 - ♦ v daném okamžiku více než 1 aktivní vlákno
 - ♦ nastává pouze na víceprocesorovém stroji
- souběžné provádění (concurrent execution)
 - ♦ více činností na různých místech prokládaně
 - ♦ v daném okamžiku pouze 1 aktivní vlákno
 - ♦ může nastat i na jednoprocessorovém stroji

Přístup ke sdíleným datům

Implicitní sdílení data v rámci OS

- synchronní události: služby poskytované procesům
 - ♦ více souběžně/paralelně prováděných procesů volá služby operačního systému
- asynchronní události: reakce na vnější přerušení
 - ♦ příjem dat ze sítě, klávesnice, myši, atd.
- modifikace datových struktur v reakci na události
 - ♦ v kontextu přerušení nebo systémového volání
- reentrantní operační systém
 - ♦ podpora souběžného zpracování více událostí
 - ♦ nutná podmínka pro paralelní zpracování

Přístup ke sdíleným datům

Explicitní sdílení dat v rámci procesu

- souběžný přístup k datovým strukturám
 - ♦ v kontextu různých vláken
- explicitní sdílení – vlákna používají stejné adresy
- situace stejná jako u implicitního sdílení v OS
 - ♦ důsledky chyb jsou typicky méně fatální

Přístup ke sdíleným datům

Problém při přístupu ke sdíleným datům

- operace nad daty sestávají z více kroků
 - ♦ při souběžném přístupu může dojít k přerušení
 - ♦ proces/vláknno přeplánován uprostřed operace
- datová struktura může být v nekonzistentním stavu
 - ♦ problém pokud s ní chce pracovat jiný proces
 - ♦ výsledek závisí na pořadí provádění procesů
- ***race condition (chyba souběhu)***

Přístup ke sdíleným datům

Příklad race condition

- přidání prvku do spojového seznamu
`new_item->next = current1->next;`
`current1->next = new_item;`
- přesun prvku mezi dvěma seznamy
`mov_item = current1->next;`
`current1->next = mov_item->next;`
`mov_item->next = current2->next;`
`current2->next = mov_item;`
- a výsledek... ?
 - ♦ v horším případě viz. Therac-25

Přístup ke sdíleným datům

Řešení problémů s race condition

- atomická změna stavu datové struktury
 - ♦ bez ohledu na počet kroků nutných k realizaci
 - ♦ dokud není operace dokončena, není možné začít jinou
- nutno identifikovat ***kritické sekce*** programu

```
begin_critical_section;  
new_item->next = current1->next;  
current1->next = new_item;  
end_critical_section;
```
- systém zajistí ***vzájemné vyloučení (mutual exclusion)***
 - ♦ pouze 1 aktivita smí provádět kód kritické sekce

Přístup ke sdíleným datům

Realizace vzájemného vyloučení

- přidání sdílené proměnné pro řízení přístupu...

```
while (locked);
```

```
locked = TRUE;
```

```
critical section
```

```
locked = FALSE;
```

- ... nefunguje!
 - ♦ do programu jsme přidali novou race condition
 - ♦ původní problém zůstal nevyřešen

Přístup ke sdíleným datům

Realizace vzájemného vyloučení

- na podruhé trochu lépe...

Proces 1

```
p1_locked = TRUE;  
while (p2_locked);  
critical section  
p1_locked = FALSE;
```

Proces 2

```
p2_locked = TRUE;  
while (p1_locked);  
critical section  
p2_locked = FALSE;
```

- ... pořád nefunguje! Ale už jsme blízko...
 - ♦ funguje, když procesy vstupují do KS postupně
 - ♦ co když do KS vstupují oba procesy *najednou*?

Přístup ke sdíleným datům

Realizace vzájemného vyloučení

- do třetice všeho dobrého...

Proces 1

```
p1_locked = TRUE;  
turn = P2;  
while (p2_locked && turn == P2);  
critical section  
p1_locked = FALSE;
```

Proces 2

```
p2_locked = TRUE;  
turn = P1;  
while (p1_locked && turn == P1);  
critical section  
p2_locked = FALSE;
```

- ... funguje? Ano!
 - ♦ procesy si dávají přednost, což řeší předchozí deadlock
 - ♦ Petersonův algoritmus, funguje pro N procesů
 - ♦ v praxi se používají jiná řešení (s podporou HW)

Přístup ke sdíleným datům

Realizace vzájemného vyloučení

- zámek s podporou HW...
while (test_and_set (locked));
critical section
locked = FALSE;
- ... funguje? Tentokrát ano!
 - ♦ funkci **test_and_set** odpovídá instrukce procesoru
 - ♦ přečte proměnnou, nastaví ji na **TRUE** a vrátí **původní hodnotu**
 - ♦ operace je atomická, čtení a zápis jsou neoddělitelné
- spin-lock: proměnná + operace lock/unlock

Přístup ke sdíleným datům

Problém spin-locků

- *aktivní čekání (busy waiting)* při zamčeném zámku
- procesor nedělá nic užitečného
 - ♦ obzvláště markarní na jednoprocessorovém systému

Nešlo by to jinak?

- *pasivní čekání*
 - ♦ pokud je zamčeno, vlákno se uspí (ready → blocked)
 - ♦ procesor může dělat něco jiného (užitečného)
 - ♦ ten kdo odemyká zámek vzbudí uspané vlákno
- uspání/vzbuzení procesu vyžaduje podporu OS
 - ♦ změna stavu vlákna, přeplánování

Přístup ke sdíleným datům

Realizace pasivního čekání

- zámek + sleep/wakeup...
while (test_and_set (locked)) sleep (queue);
critical section
locked = FALSE;
wakeup (queue);
- ... pochopitelně nefunguje!
 - ♦ test a uspání musí být atomické
 - ♦ je potřeba zámek k frontě, který se při sleep() uvolní
- OS poskytuje synchronizační primitiva
 - ♦ datová struktura + synchronizační operace

Přístup ke sdíleným datům

Pasivní čekání pomocí semaforu

- operace **down** (původně P)
 - ♦ zabere semafor pokud je volný, jinak čeká na uvolnění
- operace **up** (původně V)
 - ♦ uvolní semafor, vzbudí čekající proces (pokud existuje)
- realizace semaforu
 - ♦ celočíselná proměnná + fronta čekajících procesů
 - ♦ semafor je zabraný při hodnotě < 1 , jinak je volný
 - ♦ zabrání semaforu snižuje hodnotu o 1, uvolnění zvyšuje
- další použití semaforu
 - ♦ reprezentace volných/přidělených prostředků

Přístup ke sdíleným datům

Příklad použití semaforu

- problém producent/konzument

```
#define N 100
semaphore mutex = 1;
semaphore empty = N, full = 0;
```

Producent

```
while (1) {
    // produce item
    down (empty);
    down (mutex);
    // put item in buffer
    up (mutex);
    up (full);
}
```

Konzument

```
while (1) {
    down (full);
    down (mutex);
    // get item from buffer
    up (mutex);
    up (empty);
    // consume item
}
```

Přístup ke sdíleným datům

Realizace semaforu

- operace **down**
 - if (`--value < 0`)
 - `block_this_process()`
- nutno vyřešit realizaci vnitřní kritické sekce
 - ♦ kontrola hodnoty + zablokování
- spinlock na strukturu + odemknutí při uspání
 - ♦ zavádí aktivní čekání, kterému jsme se chtěli vyhnout
- zákaz přerušení na procesoru
 - ♦ zajistí atomicitu na velmi hrubé úrovni
 - ♦ nefunguje na více procesorech

Přístup ke sdíleným datům

Pasivní čekání pomocí monitoru

- datová struktura + operace pro čtení/změnu stavu
- operace ve stejné instanci se vzájemně vylučují
- speciální operace *wait*
 - ♦ zablokuje volající proces uvnitř monitoru
 - ♦ uvolní monitor pro jiný proces
- speciální operace *signal*
 - ♦ odblokuje zablokováné procesy, ale *neuvolňuje* monitor
 - ♦ odblokováné procesy musí nejprve získat monitor
- realizace
 - ♦ zámek + fronta spících procesů

Přístup ke sdíleným datům

Další synchronizační primitiva

- read/write zámky, reentrantní zámky
- podmínkové proměnné, bariéry

...

Ekvivalence synchronizačních primitiv

- implementace jednoho primitiva pomocí druhého
 - ♦ implementace semaforu pomocí monitoru, ...
- obecně funguje pouze u sdílené paměti
 - ♦ v případě distribuovaných systémů se situace komplikuje
 - ♦ místo datové struktury se ze semaforu stane server

Zablokování na sdílených prostředcích

Prostředky

- výpočetní – nutné k běhu programu
- synchronizační – nutné ke koordinaci konfliktů

Přidělování prostředků

- OS jako centrální správce prostředků, přiděluje právo používat nějaký prostředek (nebo jeho část, pokud je prostředek dělitelný)
- k zablokování může dojít v situaci, kdy procesy žádají současné přidělení více prostředků

Zablokování na sdílených prostředcích

Práce s prostředky

- žádost o prostředek (blokující)
- použití přiděleného prostředku
- odevzdání prostředku (dobrovolné, při skončení)

Zablokování (Deadlock)

- Množina procesů je zablokována, jestliže každý proces z této množiny čeká na událost, kterou může způsobit pouze jiný proces z této množiny.

Zablokování na sdílených prostředcích

Ilustrační příklad – večeřící filozofové

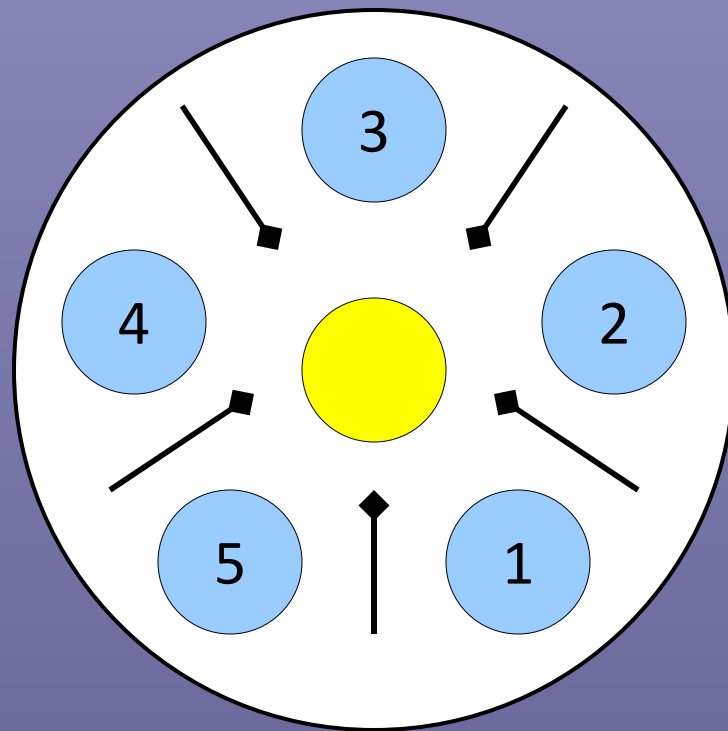
- 5 filozofů žije pohromadě, čas tráví přemýšlením
 - ♦ pokud má filozof hlad, jde se do jídelny najíst
 - ♦ v jídelně je prostřeno pro 5 osob, na stole mísa špaget
 - ♦ k jídlu je potřeba použít 2 vidličky...
- ... na stole je ovšem pouze 5 vidliček!
 - ♦ 1 vidlička mezi dvěma talíři
- v kontextu operačního systému...
 - ♦ 5 procesů soupeřících o prostředky
 - ♦ každý prostředek sdílen pouze dvěma procesy

Zablokování na sdílených prostředcích

Ilustrační příklad – večeřící filozofové

- co když dostanou hlad všichni filozofové najednou?
- a navíc si všichni vezmou nejprve vidličku nalevo?

```
#define N 5
void phil (int i) {
    for (;;) {
        think ();
        take_fork (i);
        take_fork ((i + 1) % N);
        eat ();
        put_fork (i);
        put_fork ((i + 1) % N);
    }
}
```



Zablokování na sdílených prostředcích

Ilustrační příklad – večeřící filozofové

- funkce `take_fork` je blokovací
 - ♦ všichni najednou zvednou svoji levou a čekají na pravou
- funkce `take_fork` je opatrná
 - ♦ pokud nemohu vzít druhou vidličku, položím tu první
 - ♦ všichni zvednou levou, podívají se doprava, položí levou
 - ♦ filozofové pracují ale nenají se – livelock + vyhladovění

Možná řešení

- jeden z filozofů vezme vidličky v jiném pořadí
- do jídelny vpustíme pouze 4 filozofy najednou
- randomizace časů

Zablokování na sdílených prostředcích

Formální model zablokování

- stav reprezentován orientovaným grafem

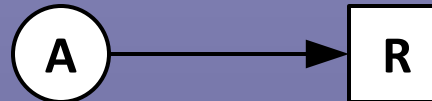
- ♦ prostředky



- ♦ procesy



- ♦ žádost o prostředek



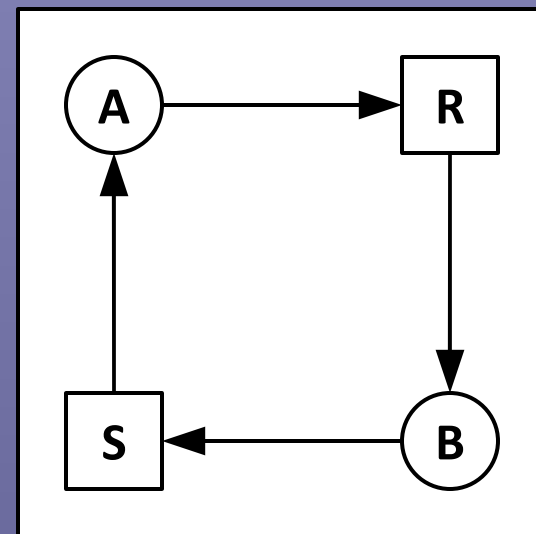
- ♦ vlastnění prostředku



Zablokování na sdílených prostředcích

Formální model zablokování

- cyklus v grafu indikuje potenciální deadlock
 - ♦ požadavky vyjadřují skutečné potřeby procesu
 - ♦ pokud nejsou požadavky uspokojeny, proces je zablokován a čeká na uvolnění prostředku
 - ♦ pokud prostředek vlastní jiný proces, který je také zablokován, nemůže dojít k uvolnění prostředku
 - ♦ původní proces zůstane zablokován



Zablokování na sdílených prostředcích

Coffmanovy podmínky

- při splnění všech podmínek dojde k zablokování
- **Výlučný přístup (*Exclusive use*)**
 - ♦ prostředek je přidělen výhradně jednomu procesu
- **Neodnímatelnost (*No preemption*)**
 - ♦ přidělené prostředky nemohou být odebrány
- **Drž a čekej (*Hold and wait*)**
 - ♦ proces může zároveň držet prostředek a čekat na další
- **Kruhová závislost (*Cyclic dependency*)**
 - ♦ procesy čekají na prostředky v kruhu

Zablokování na sdílených prostředcích

Řešení problému zablokování

- deadlock prevention
 - ♦ napadení jedné z Coffmanových podmínek
- deadlock avoidance
 - ♦ zabránit realizaci všech Coffmanových podmínek
- deadlock detection & recovery
 - ♦ řešení problému až když nastane
- pštrosí algoritmus
 - ♦ předstíráme, že problém neexistuje
 - ♦ problém typicky vyřeší uživatel (kill -9)

Zablokování na sdílených prostředcích

Deadlock prevention

- napadení některé z Coffmanových podmínek
 - ♦ nelze aplikovat obecně, závisí na typu prostředku
- výlučný přístup (exclusive use)
 - ♦ spooling – iluze výlučného přístupu
- neodnímatelnost (no preemption)
 - ♦ odnímatelné prostředky lze odejmout bez následků
 - procesor (přeplánování), paměť (swapping)
 - ♦ neodnímatelné prostředky nelze odejmout bez nebezpečí selhání výpočtu
 - ♦ obecně nevhodné z pohledu programátora

Zablokování na sdílených prostředcích

Deadlock prevention

- drž a čekej (hold and wait)
 - ♦ OS vrátí chybu místo zablokování procesu
 - ♦ nutno žádat o všechny prostředky najednou
 - ♦ před žádostí nutno všechny prostředky uvolnit
- kruhová závislost (cyclic dependency)
 - ♦ očíslování prostředků, možno žádat pouze o prostředky, s vyšším číslem
 - ♦ pořadí nemusí být globální, ale pouze v rámci množiny prostředků sdílených současně v nějakém kontextu
 - zámky v subsystémech operačního systému

Zablokování na sdílených prostředcích

Deadlock avoidance

- výchozí stav
 - ♦ počet dostupných a přidělených prostředků
 - ♦ procesy nejsou zablokovány
- následující stav
 - ♦ při přidělení dalších prostředků
 - ♦ přechod pouze pokud je následující stav bezpečný
- bezpečný stav
 - ♦ existuje pořadí, v jakém uspokojit všechny procesy
- nebezpečný stav
 - ♦ uvedené pořadí přidělování prostředků neexistuje

Zablokování na sdílených prostředcích

Deadlock avoidance – Bankéřův algoritmus

- bankéř disponuje jistou částkou a slíbil různým zákazníkům různé úvěry, v součtu přesahující disponibilní částku
 - ♦ předpokládá, že zákazníci nebudou potřebovat úvěr v plné výši najednou
- dodatečné informace
 - ♦ max. počet, o který bude každý proces žádat
- bezpečný stav systému
 - ♦ je možné plně uspokojit alespoň 1 proces
 - ♦ takový proces časem prostředky vrátí

Zablokování na sdílených prostředcích

Deadlock avoidance – Bankéřův algoritmus

Má Max

A	0	6
B	0	5
C	0	4
D	0	7

Volné: 10

bezpečný

Má Max

A	1	6
B	1	5
C	2	4
D	4	7

Volné: 2

bezpečný

Má Max

A	1	6
B	2	5
C	2	4
D	4	7

Volné: 1

nebezpečný

Zablokování na sdílených prostředcích

Deadlock avoidance – Bankéřův algoritmus

- složité rozhodování o přidělení prostředků
 - ♦ algoritmus má navíc složitost $O(N^2)$
- požadované informace jsou typicky nedostupné
- efektivnější řešení řešit až vzniklé problémy
 - ♦ typicky používané v databázových systémech

Zablokování na sdílených prostředcích

Deadlock detection

- model závislostí mezi procesy ve formě grafu
- test na přítomnost kruhových závislostí
 - ♦ hledání cyklu v orientovaném grafu

Deadlock recovery

- odebrání prostředku
 - ♦ na přechodnou dobu, pod dohledem operátora
- odstranění nepohodlných procesů
 - ♦ proces z cyklu závislostí
 - ♦ proces mimo cyklus vlastníčí identický prostředek

Zablokování na sdílených prostředcích

Deadlock recovery

- checkpointing/rollback
 - ♦ OS ukládá stav procesů
 - ♦ restart procesu v předchozím stavu
- transakční zpracování
 - ♦ typické pro databázové systémy