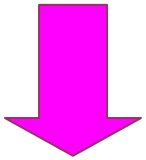
ÚVOD DO OPERAČNÍCH SYSTÉMŮ

Ver.1.00

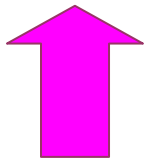
Procesy a vlákna - synchronizace

České vysoké učení technické Fakulta elektrotechnická

Studijní materiály a informace o předmětu



- http://measure.feld.cvut.cz/vyuka/predmety/bakalarske/navody
- http://aldebaran.feld.cvut.cz/vyuka/uvod_do_os



Použitá literatura

[1] Stallings, W.: Operating Systems. Internals and Design Principles. 4th Edition. Prentice Hall, New Jersey, 2001.

[2] Silberschatz, A. – Galvin, P. B. - Gagne, G. : Operating System Concepts. 6th Edition. John Wiley & Sons, 2003.

[3] Tanenbaum, A.: Modern Operating Systems. Modern Operating Systems. Prentice Hall, New Jersey, 2008.

Synchronizace procesů/vláken

Cíle synchronizace:

- zabránit konfliktům mezi vlákny při přístupu ke sdíleným prostředkům (paměti, souborům, periferním zařízením, ap.)
- pozastavení běhu vláken, dokud nejsou splněny specifikované podmínky nebo nenastane určená událost;
- zajistit vykonání určitých programových sekvencí v požadovaném pořadí.

Synchronizace procesů/vláken – pokračování

Optimální mechanismus synchronizace:

- založen na možnosti převedení procesu/vlákna do **blokovaného stavu** (*blocked, waiting*, *sleeping*), kdy mu není přidělován čas procesoru (viz 5-ti stavový model procesu/vlákna)
- v tomto stavu proces/vlákno setrvá tak dlouho, dokud nebudou splněny požadované podmínky (např. ukončení jiného vlákna nebo procesu, nenulový stav semaforu, uvolněný mutex, uplynutí nastaveného časového intervalu apod.)
- jakmile podmínky nastanou, systém proces/vlákno probudí, tzn. přesune je (v souladu s plánovací strategií) do příslušné fronty aktivních procesů připravených na přidělení procesoru.

Stavový model procesu/vlákna

Základní pětistavový model

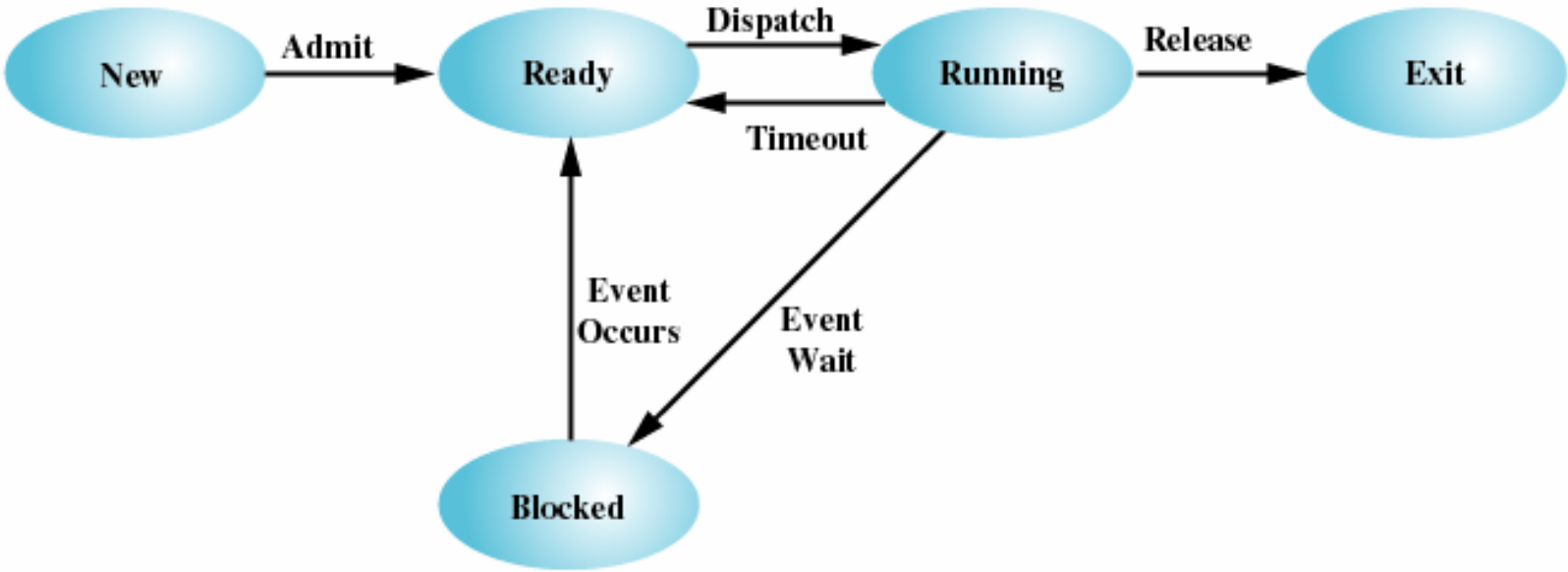


Figure 3.6 Five-State Process Model

Viz [1]

Kritická sekce (*Critical Section*)

Kritická sekce:

část programu, kde dochází k přístupu procesu/vlákna ke sdílenému prostředku (např. sdílená proměnná).

Procesy/vlákna vstupující do kritické sekce musí být synchronizovány!!!

Problém kritické sekce 1

Podmínky přístupu do kritické sekce:

- výhradní přístup – vstup do kritické sekce je povolen nejvýše jednomu procesu
- žádné předpoklady nesmí být kladeny na rychlost a počet procesorů
- proces/vlákno běžící mimo kritickou sekci nesmí blokovat ostatní procesy/vlákna
- omezené čekání – rozhodnutí o vstupu nesmí být pro některého čekajícího odkládáno do nekonečna

Problém kritické sekce 2

Přístup do kritické sekce:

```
do {
    entry section
      critical section
    exit section
      reminder section
} while (1);
```

Přístup nelze řešit pomocí jednoduché sdílené proměnné (např. locked=FALSE vstup volný, locked=TRUE vstup blokovaný) !!!

Problém kritické sekce 3

Přístup do kritické sekce: mějme 2 procesy P0, P1 a sdílenou proměnnou pro řízení přístupu *locked*

```
while (locked != 0) ;   /*čeká na vstup do KS */
locked = TRUE;
/* critical section */
locked = FALSE;
/* reminder section */
```

Toto je špatné řešení !!! Proč?

Problém kritické sekce 4

Řízení přístupu do KS podle sdílené proměnné *lock:*

- jednotlivé operace jsou prováděny ve více krocích
- při souběžném přístupu může dojít k jejich přerušení
- vlákno je přeplánováno uprostřed operace
- výsledek závisí na pořadí provádění vláken
- nastává tzv. chyba souběhu (*race condition*)

Problém kritické sekce 5

Petersonův algoritmus:

```
p1_locked = FALSE;
p2_locked = FALSE;
```

```
/* 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;
```

Synchronizace využívající HW počítače 1

Petersonův algoritmus se v praxi nepoužívá, místo něho se aplikuje zámek s podporou HW řešení procesoru.

Instrukce TSL (*Test and Set Lock*)

```
boolean TestAndSet(boolean *target) {
    boolean rv;
    rv = *target;
    *target = true;
    return rv;
}
```

Funkce TSL přečte proměnnou *target,* nastaví ji na TRUE a vrátí původní hodnotu.

Funkce je atomická, čtení a zápis se provádí jako jediná, nedělitelná, nepřerušitelná operace!!!

Synchronizace využívající HW počítače 2

Použití funkce **TestAndSet()**:

```
boolean TestAndSet(boolean *target);
boolean lock = false;

  while (TestAndSet(&lock)) ;
       critical section
  lock = false;
       remainder section
```

Aktivní a pasivní čekání

Aktivní čekání -> Spin-lock: proměnná + operace lock/unlock

- výsledek operace TestAndSet() se ukládá do proměnné
- aktivní čekání (*busy waiting*) při zamčeném zámku (procesor je „zbytečně“ vytěžován)

Pasivní čekání

- pokud je zámek uzamčen, proces/vlákno se „uspí“, přesune do blokovaného stavu (*blocked, sleeping, waiting*)
- změnu stavu (uspání/probuzení) řeší OS
- OS poskytuje tzv. synchronizační primitiva = datová struktura + synchronizační operace (semafor, mutex, monitor)

Semafor

- synchronizační objekt obsahující čítač a frontu čekajících procesů/vláken
- hodnota semaforu se pohybuje od nuly do specifikované maximální hodnoty

Poskytuje 3 atomicky prováděné operace:

- **Init()**: Čítač se nastaví na zadané číslo (většinou 1) a fronta se vyprázdní.
- **Down()**: Pokud je čítač větší než nula, hodnota se sníží o 1, pokud je hodnota nulová, proces/vlákno se zablokuje a uloží do fronty.
- **Up()**: Pokud nějaké procesy/vlákna čekají ve frontě, potom se první z nich probudí. Pokud je fronta prázdná, čítač se inkrementuje o 1.

Mutex, monitor

Mutex (***Mut**ual **Ex**clusion*)

- synchronizační objekt typu binární semafor (hodnota čítače 1 nebo 0)

Monitor = datová struktura + operace pro čtení/změnu stavu

- synchronizační objekt realizovaný jako speciální konstrukce programovacího jazyka (musí ho tedy implementovat překladač), typicky implementovaná pomocí jiného synchronizačního primitiva
- výhodou monitoru oproti jiným objektům je jeho vysokoúrovňovost – snadněji se používá a je bezpečnější
- operace **wait** (zablokuje volající proces uvnitř monitoru,uvolní monitor pro jiný proces)
- Operace **signal** (odblokuje zablokované procesy, ale ***neuvolňuje*** monitor, odblokované procesy musí nejprve získat monitor)

ÚVOD DO OPERAČNÍCH SYSTÉMŮ

KONEC 5. přednášky

České vysoké učení technické Fakulta elektrotechnická

Vznik nového procesu v Unixu

```
int main(...)
{
        ...
        if ((pid = fork()) == 0)                    // vytvoření nového procesu
        {
                fprintf(stdout, "Child pid: %i\n", getpid());
                err = execvp(command, arguments);   // toto je nový proces („dítě“)
                fprintf(stderr, "Child error: %i\n", errno);
                exit(err);
        }
        else if (pid > 0)                           // toto je rodičovský proces
        {
                fprintf(stdout, "Parent pid: %i\n", getpid());
                pid2 = waitpid(pid, &status, 0);    // čekání na ukončení procesu
        }
        ...
        return 0;
}
```

Příklad: implementace příkazu shellu v Unixu pomocí volání jádra a knihovních funkcí jazyka C