ÚVOD DO OPERAČNÍCH SYSTÉMŮ

Ver.1.00

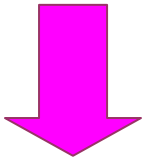
Procesy a vlákna – IPC

Komunikace mezi procesy

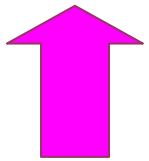
(*IPC = Inter-Process Communication*)

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.

Komunikace mezi procesy (IPC)

Meziprocesová komunikace (IPC) = techniky výměny dat mezi dvěma nebo více procesy/vlákny

- zahrnuje metody synchronizace (semafor, mutex, monitor – viz přednáška č.5, zasílání zpráv, sdílení paměti, vzdáleného volání procedur, aj.
- při programování se využívá API dané obecným standardem (POSIX=IEEE 1003, SUS Single Unix Specification), příslušnou platformou (Win32 API) nebo jednoúčelovou implementací
- může probíhat na jednom příp. více počítačů propojených počítačovou sítí

Problémy při synchronizaci procesů/vláken
Uváznutí (*Deadlock*) a vyhladovění (*Starvation*)

Uváznutí (= zablokování):

množina procesů/vláken čeká na událost, kterou může generovat pouze jiný proces/vlákno z této množiny

Vyhladovění:

procesu/vláknu jsou trvale odpírány požadované sdílené prostředky bez kterých nemůže dokončit úlohu.

IPC: zasílání zpráv (Message Passing)

Základní operace:

send(destination,&message)

receive(source,&message)

send() – vysílá zprávu na dané cílové místo;

receive() – přijímá zprávu z určeného zdrojového místa;

pokud není zpráva dostupná, je proces blokován (příp. pokračuje s chybovým kódem)

Klasické problémy synchronizace mezi procesy (IPC)

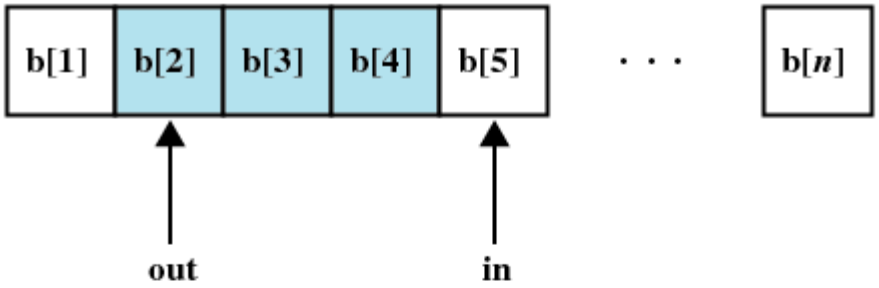
Problém producent-konzument (*Producer-Consumer Problem*)

- jeden proces (=producent) poskytuje data a ukládá je do sdílené paměti
- druhý proces (=konzument) data ze sdílené paměti odebírá

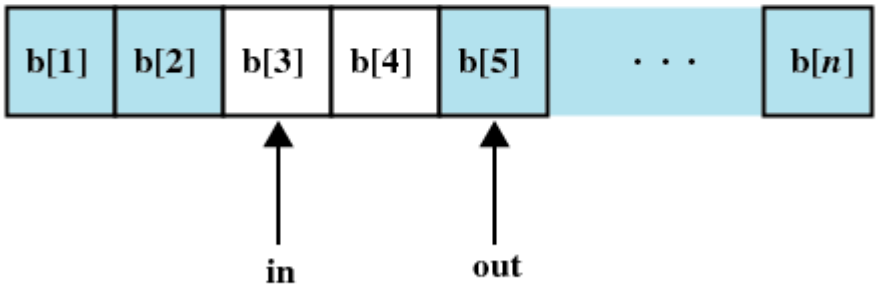
Blokování pokud:

- producent vkládá do plného bufferu
- konzument odebírá z prázdného bufferu

Problém producent – konzument 1



(a)



(b)

Figure 5.12 Finite Circular Buffer for the Producer/Consumer Problem

Viz [1]

Problém producent – konzument 2 (viz [3])

```
define N 100                    /* number of slots in buffer */
int count = 0;                  /* number of items in buffer */

void producer(void)
{   int item;
    while(TRUE) {
      item = produce_item();
      if (count == N) sleep();          /* if buffer is full, go to sleep */
        insert_item(item);
        count = count + 1;
        if (count == 1) wakeup(consumer);  /* was buffer empty? */
    }
}
```

Problém producent – konzument 3

```
void consumer(void)
{   int item;
    while (TRUE) {
      if (count == 0)                    /* critical point for context switching */
        sleep();                         /* if buffer is empty, go to sleep */
      item=remove_item()
      count = count – 1;
      if (count == N - 1) wakeup(producer);      /* was buffer full? */
      consume_item(item);
    }
}
```

Toto je špatné řešení !!! Může dojít k chybě souběhu.

Problém producent – konzument: použití semaforů (viz [3])

```
#define N 100                   /* number of slots in buffer */
typedef int semaphore;
semaphore mutex = 1;            /* control access to critical section (CS)*/
semaphore empty = N;            /* counts empty buffer slots*/
semaphore full = 0;             /* counts full buffer slots*/
```

```
void producer(void) {
    itemtype item;
    while (TRUE) {
        item=produce_item();
        down(&empty);
        down(&mutex);   /* enter CS*/
        insert_item(item);
        up(&mutex); /* leave CS*/
        up(&full);
    }
}
```

```
void consumer(void) {
    itemtype item;
    while (TRUE) {
        down(&full);
        down(&mutex); /* enter CS */
        item=remove_item();
        up(&mutex); /* leave CS */
        up(&empty);
        consume_item(item);
    }
}
```

Problém čtenáři – písař

Více procesů/vláken, v roli čtenáře nebo písaře, musí přistupovat ke sdílené paměti, aby do ní zapisovaly nebo z ní četly. Přitom je potřeba zaručit integritu dat, která může být porušena souběhem čtenáře a písaře nebo dvou písařů.

Praktický příklad: rezervační systém letenek

Problém čtenáři – písař (korektní řešení) (viz [3])

```
int rc = 0;                 /* readers counter */
semaphore mutex = 1;        /* controls access to 'rc' */
semaphore db = 1;           /* access to database */
```

```
void reader(void) {
    while(TRUE){
        down(&mutex);
        rc = rc + 1;
        if (rc == 1) down(&db);
        up(&mutex);
        read_data_base();
        down(&mutex);
        rc = rc - 1;
        if (rc == 0) up(&db);
        up(&mutex);
        use_data_read();
    }
}
```

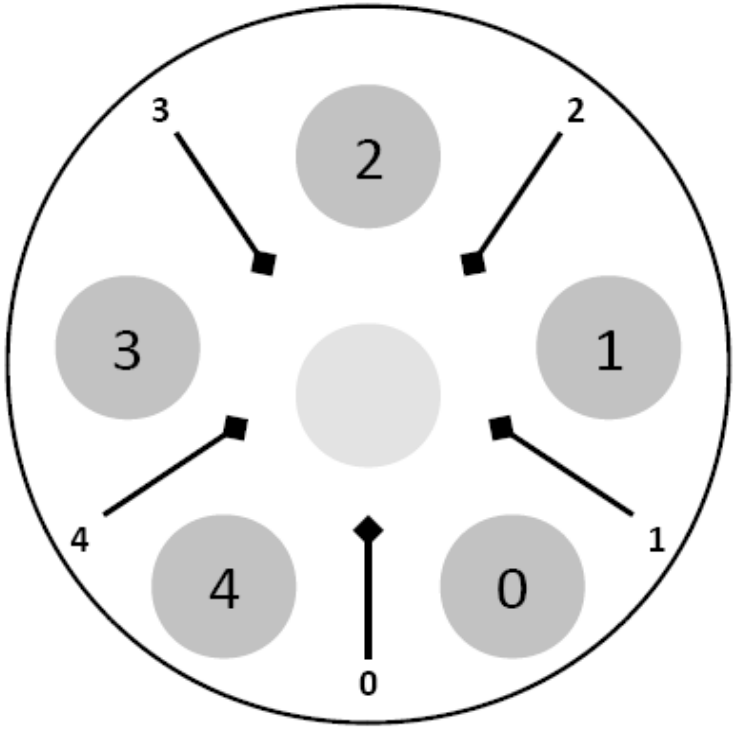
```
void writer(void) {
    while(TRUE) {
        think_up_data();
        down(&db);
        write_data_data();
        up(&db);
    }
}
```

Problém večeřící filosofové

Problém procesů, které soutěží o výlučný přístup k omezenému počtu prostředků.

Zadání:

N filozofů sedí kolem kulatého stolu a každý z nich buď přemýšlí nebo jí.
K jídlu potřebují současně levou a pravou vidličku.



Problém večeřící filosofové – řešení (viz [3])

```
#define N 5
#define LEFT ((i+N-1) % N)
#define RIGHT ((i+1) % N)

enum stat(thinking, hungry, eating);
enum stat state[N];
semaphore mutex=1;
semaphore s[N]; # initially set to 0

void philosopher(int i) {
    while (TRUE) {
        think();
        take_forks(i);
        eat();
        put_forks(i);
    }
}
```

```
void take_forks(int i) {
    down(&mutex);
    state[i] = hungry;
    test(i);
    up(&mutex);
    down(&s[i]);
}
void put_forks(int i)
{
    down(&mutex);
    state[i] = thinking;
    test(LEFT);
    test(RIGHT);
    up(&mutex);
}
void test(int i)
{
    if (state[i] == hungry &&
        state[LEFT] != eating &&
        state[RIGHT] != eating)
    {
        state[i] = eating;
        up(&s[i]);
    }
}
```

Příklady meziprocesové komunikace

Nezávislé na platformě:

- Roury (*pipes*)
- Sokety (*sockets*)
- Semafory

Závislé na platformě:

- Signály
- Fronty zpráv
- Sdílená paměť
- Zasílání zpráv
- Paměťově mapované soubory
- aj.

Příklady meziprocesové komunikace v OS Unix

Signály

Viz další presentace

ÚVOD DO OPERAČNÍCH SYSTÉMŮ

KONEC 6. přednášky

České vysoké učení technické Fakulta elektrotechnická