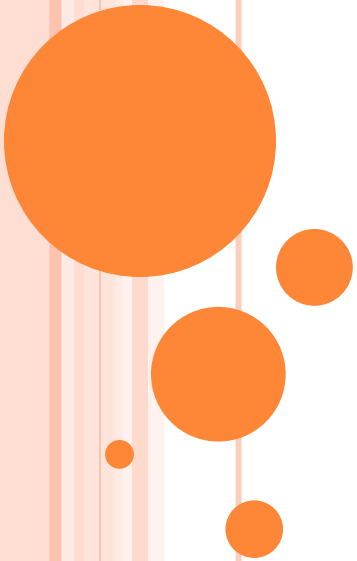


KIV/ZOS cvičení 10

L. Pešička, verze 2014

Obsah

- Producent konzument obecným semaforem
- Zámek v pthread (jazyk C)
- Java
  - Zámek
  - Semafor
  - Vlákna
- TSL obecně
  - Ukázky C, Java
- Monitor
  - Obecně – podmínkové proměnné aj.
  - Příklad monitoru „hospoda“
  - Monitor v Javě

Producent – konzument obecně semaforem

```
semaphore  ...  =  ... ;
semaphore  ...  =  ... ;
semaphore  ...  =  ... ;
Buffer velikosti N;

producent() {  while(1) { ... }   }
konzument() { while(1) { ... } }

cobegin
  producent()  ||  konzument()
coend
```

Dopište kód producenta a konzumenta:

Vloz_poloz_do bufferu()
Vyber_poloz_z bufferu()
P()
V()
produkuj_polozku()
Konzumuj_polozku()

Příklady na cvičení

Přihlášení do portálu – Courseware – KIV/ZOS – Cvičení – Materiály ke cvičením – C, Java příklady

soubor	obsah
Pthread-semafor	C: vlákna, semafory
Příklady synchronizace	C, Java: • Mutex • Semafor • Monitor • spinlock_TSL • CAS
monitorJavanew	Java Bariéra vytvořená monitorem bez synch. metod
pr_zavoznik	C, Java Monitor hospoda

MUTEX – PTHREAD (C)

- pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

- pthread_**mutex_lock**(&lock); -- zamkni
- KS
- pthread_**mutex_unlock**(&lock); -- odemkni

- pthread_**mutex_trylock**(&lock);
  - zkusí zamknout, pokud má zámek někdo jiný vrátí se hned
- pthread_**mutex_destroy**(&lock);
  - zruší zámek

Semafor - C - opakování

```
#include <semaphore.h>
sem_t s;                    /* semafor          */
int x = 0;
sem_init(&s, 0, 1);         /* inicializace na 1*/
…
sem_wait(&s);               /* P(s)             */
x++;                        /* Kritická sekce   */
sem_post(&s);               /* V(s)             */
…
sem_destroy(&s);
```

TSL - C

```
void tsl_spinlock() {
    while (__sync_lock_test_and_set(&lockInt, 1) == 1) {
        // místo aktivního čekání se vzdáme
        // přiděleného  procesorového času
        sched_yield();
    }
}

void tsl_spinunlock() {
    __sync_lock_release(&lockInt);
}
```

MONITOR_BARIERA - C

- Mutex + podmínková proměnná = monitor
- Ukázka vytvoření bariéry pomocí monitoru

```
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

pthread_mutex_lock(&lock);
    pthread_cond_wait(&cond, &lock);
    pthread_cond_broadcast(&cond);
pthread_mutex_unlock(&lock);
```

JAVA – SEMAFOR

- import java.util.concurrent.Semaphore;

- Semaphore sem = **new Semaphore(1);**

- sem.acquire(); .. operace P()
- KS
- sem.release(); .. operace V()

JAVA – ZÁMEK

- import java.util.concurrent.locks.Lock;
- import java.util.concurrent.locks.ReentrantLock;

- Lock lock = new ReentrantLock();

- lock.lock(); -- zamkni
- KS -- kritická sekce
- lock.unlock(); -- odemkni

JAVA – VLÁKNA

- Vytvoření třídy, která bude potomkem třídy **Thread**

- Vytvořením třídy, která implementuje rozhraní **Runnable**

- Do metody **run()** – napsat kód vlákna
- Zavolání metody **start()** – spuštění vlákna

```
class CounterThread extends Thread {
// Tato metoda obsahuje kód,
// který bude vykonáván v našem vlákně
public void run() {
    for ( int i = 0; i < 10; i++)
            { System.out.println(i); }
    }
}

// Vytvoříme nové vlákno
Thread counterThread = new CounterThread();
// spustíme vlákno, kód metody CounterThread.run()
// se od této chvíle začne vykonávat v novém  vlákně
counterThread.start();
```

Zdroj: http://kore.fi.muni.cz:5080/wiki/index.php/Java:Vl%C3%A1kna

```
class Counter implements Runnable {
// Tato metoda obsahuje kód,
// který bude vykonáván v našem vlákně
    public void run() {
            for(int i = 0; i < 10; i++) { System.out.println(i); }
    }
}

 Runnable counter = new Counter();
// Vytvoříme nové vlákno, jako parametr /
/ konstruktoru předáme referenci na
// naši implementaci rozhraní Runnable

Thread counterThread = new Thread(counter);
// spustíme vlákno, kód metody Counter.run()
// se od této chvíle začne vykonávat v novém vlákně
counterThread.start();
```

INSTRUKCE TSL (OBECNĚ)

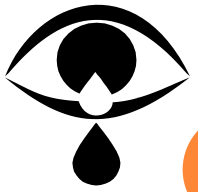
- Tato nebo obdobná poskytována HW počítače
- Provede atomicky dvě operace
  - Nastaví zámek na hodnotu ZAMCENO
  - Vrátí původní hodnotu zámku

```
#define ZAMCENO 1
atomic int TSL(int *zamek) {
   int oldValue;
   oldValue = zamek;
   zamek = ZAMCENO;
    return oldValue }
```

Funkce by musela být atomická, tj. provést se bez přeplánování

Význam TSL

- Chceme získat zámek
- Zavoláme
  i = TSL(&zamek);
- Otestujeme:
  - i je rovno ODEMCENO (tj. 0)
    - Původní hodnota zámku byla ODEMCENO
    - Instrukce TSL zámek nastavila na ZAMCENO
    - Jelikož TSL je atomické, zámek se podařilo zamknout nám
    - Zámek je náš ☺
  - i je rovno ZAMCENO (tj. 1)
    - Zámek už byl zamknutý
    - Instrukce TSL jej sice opět nastavila na ZAMCENO
    - Ale zámek není stejně náš, musíme zkusit znovu ☹



Využití TSL pro implementaci zámku

Je dáno:

```
#define ZAMCENO 1
#define ODEMCENO  0
int zamek;
int TSL(&zamek)                    //atomickou TSL
```

Napište kód metod:

```
void TSL_spinlock()                // zamkni
void TSL_spinunlock()              // odemkni
```

(použití: *TSL_spinlock(); KS ; TSL_spinunlock()*)

Napište kód metod TSL_spinlock() a TSL_spinunlock()

Ukázka TSL

- Ukázkové příklady na portále
  - V jazyce C
  - V Javě

Monitor

- Výhody oproti semaforu
  - Ošetření kritických sekcí, synchronizace v jednom modulu (není roztroušené po celém kódu programu) – přehlednost
- Obecný monitor
  - V monitoru může být N procesů (vláken)
  - Z nich 1 je aktivní a N-1 je blokovaných
- Podmínková proměnná
  - Představuje frontu procesů blokovaných nad podmínkou (!!)
- Monitor hospoda
  - Viz příklad v Coursewaru-Cvičení (část BACI)

Monitor hospoda

- Zákazník volá getpivo()
  - Není-li pivo dostupné, blokuje nad podmínkovou proměnnou
- Závozník kromě zvýšení proměnné počtu piv signalizuje závoz piva – přivezeli 100 piv, 100x signalizuje signal(c1)

Java - synchronizace

- Monitory – hlavní mechanismus
- Každý objekt – svůj monitor

- Metoda označená synchronized
- synchronized (objekt) { …. }

Při synchronizaci zvažte i použití mechanismů z java.util.concurent – i kvůli výkonnosti

Synchronizovaná metoda

```
class Counter{
   // sdílená proměnná
private int currentValue = 0;

   public synchronized int next() {
         // kritická sekce
         // musí proběhnout atomicky
         return  (++currentValue);
   }
}
```

Pouze část kódu je synchronizovaná

```
class Counter {
   // sdílená proměnná
   private int currentValue = 0;

   public int next() {
         synchronized(this) {
               //  kritická sekce
               // musí proběhnout atomicky
               return  (++currentValue);
         }
   }
}
```

Objekt, vůči jehož monitoru synchronizujeme

Ukázka viz
http://download.oracle.com/javase/tutorial/essential/concurrency/locksync.html

```
public class MsLunch {
    private long c1 = 0;
    private long c2 = 0;
    private Object lock1 = new Object();
    private Object lock2 = new Object();

    public void inc1() {
        synchronized(lock1) {  c1++; }
    }

    public void inc2() {
        synchronized(lock2) { c2++; }
    }
}
```