KIV/ZOS CVIČENÍ 8

L. Pešička, 2024

OBSAH

- Zámek, monitor v pthread (jazyk C)
- Java
  - Zámek
  - Semafor
  - Vlákna
- TSL obecně
- Monitor
  - Obecně – podmínkové proměnné aj.
  - Příklad monitoru „hospoda“
  - Monitor v Javě
- Opakování

PŘÍKLADY NA CVIČENÍ

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

soubor	obsah
pthreads-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)

1. pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

2. pthread_**mutex_lock**(&lock); -- zamkni
3. Kritická Sekce
4. pthread_**mutex_unlock**(&lock); -- odemkni

Bude inicializovaný a odemčený

Makro pro inicializaci

Také je možné volat fci pthread_mutex_init

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

MUTEX - CVIČENÍ

- Modifikujte příklad z minulého cvičení, aby místo semaforu používal k ošetření kritické sekce mutex

1. wget http://home.zcu.cz/~pesicka/zos/sem1.c
2. cp sem1.c mutex1.c
3. Použití mutexu místo semaforu – úpravy viz předchozí slide
4. gcc –l pthread –o mutex1 mutex1.c

SEMAFOR - C

```
#include <semaphore.h>
sem_t s;                        /* semafor          */
int x = 0;
sem_init(&s, 0, 1);             /* inicializace na 1*/
…
sem_wait(&s);                   /* P(s)             */
x = x - 7;                      /* Kritická sekce   */
sem_post(&s);                   /* V(s)             */
…
sem_destroy(&s);
```

MONITOR - C

- mutex + podmínková proměnná = monitor
- https://cs.wikipedia.org/wiki/Monitor_(synchronizace) přečíst!

```
1.  pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
2.  pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

3.  pthread_mutex_lock(&lock);
4.      pthread_cond_wait(&cond, &lock);
5.      pthread_cond_signal(&cond);
6.      pthread_cond_broadcast(&cond);
7.  pthread_mutex_unlock(&lock);
```

BARIÉRA

- bariéra nastavená na N vláken
- vlákna volají bariera()
- N-1 vláken se zde zablokuje
- když přijde N-té vlákno, všichni najednou projdou bariérou

- použití: např. iterační výpočty

BARIÉRA POMOCÍ MONITORU - C

```
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

pthread_cond_t  cond  = PTHREAD_COND_INITIALIZER;
```

BARIÉRA POMOCÍ MONITORU - C

```
int barieraCount = 0;
int BAR_PRAH = 5;

void  bariera() {
  pthread_mutex_lock(&lock);
     barieraCount++;
     if (barieraCount < BAR_PRAH) {
          pthread_cond_wait(&cond, &lock);
     } else {
          pthread_cond_broadcast(&cond);
          barieraCount = 0;
    }
    pthread_mutex_unlock(&lock);
}
```

BARIERA - CVICENI

- wget http://home.zcu.cz/~pesicka/zos/bariera.c
- Vyzkoušet nastavit práh na 3, co se stane?

POZNÁMKA

- https://en.wikipedia.org/wiki/Spurious_wakeup
  - Lépe po probuzení na čekání zkontrolovat stav podmínky
  - Důvěřuj ale prověřuj

JAVA – ZÁMEK

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

1. Lock lock = new ReentrantLock();

2. lock.lock(); -- zamkni
3. Kritická Sekce -- kritická sekce
4. lock.unlock(); -- odemkni

JAVA – SEMAFOR

- import java.util.concurrent.Semaphore;

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

2. sem.acquire(); .. operace P()
3. Kritická Sekce
4. sem.release(); .. operace V()

JAVA – VLÁKNA

1. Vytvoření třídy, která bude potomkem třídy **Thread**

2. Vytvořením třídy, která implementuje rozhraní **Runnable**

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

```
class Counter implements Runnable {
// Kód, který bude vykonáván v našem vlákně
  public void run() {
        for(int i = 0; i < 10; i++) { System.out.println(i); }
  }
}
...
public class Vlakna {
  public static void main(String[] args) {
  Runnable counter = new Counter();
  //Vytvoříme nové vlákno, jako parameter  konstruktoru předáme
  // referenci na naši implementaci rozhraní Runnable
  Thread vl1 = new Thread(counter);
  // spustíme vlákno, kód metody Counter.run()
  // se od této chvíle začne vykonávat v novém vlákně
  vl1.start();
}}
```

JAVA - SYNCHRONIZACE

- 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++;}
  }
}
```

lock1
a
lock2

Slouží pouze k synchronizaci

BARIÉRA - JAVA

```
private synchronized void barrier() {
   cnt++;
   if ( cnt < N)
        try {  wait(); } catch (InterruptedException ex) {}
  else {
            cnt = 0;
            notifyAll();
   }
}
```

PRODUCENT – KONZUMENT JAVA

- wget http://home.zcu.cz/~pesicka/zos/ProdKonzum.java

- Jeden producent (10 položek)
- Dva konzumenti (každý čte 5 položek)

- Pool (úložiště zde na jednu položku)
  - Synchronizované metody
  - Buffer na jednu položku je buď plný nebo prázdný

INSTRUKCE TSL (OBECNĚ)

- Tato nebo obdobná poskytována HW počítače
  TSL R, zamek

- Provede atomicky dvě operace
  - Nastaví proměnou zámek na hodnotu ZAMCENO (1)
  - Vrátí původní hodnotu zámku v registru
  - Nikdo jiný nemůže manipulovat s proměnnou zamek v daném okamžiku
  - Je to ošetřeno i na CPU s více jádry

Význam TSL

- Chceme získat zámek
- Provedeme TSL R, zámek
- Otestujeme:
  - R 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, získali jsme např. přístup do kritické sekce jako jediní
  - R je rovno ZAMCENO (tj. 1)
    - Zámek už byl zamknutý někým
    - Instrukce TSL jej sice opět nastavila na ZAMCENO
    - Ale zámek není stejně náš, musíme zkusit znovu ☹

POUŽITÍ TSL PRO SPIN_LOCK

```
enter_region:          ; funkce spin_lock

  TSL R, flag          ; Test and Set Lock
                       ; flag je sdílená proměnná
                       ; hodnota flag nakopírovaná do registru
                       ; a pak je automaticky nastavena na 1 (zamčeno)

  CMP R, #0            ;  Měla proměnná flag hodnotu odemčeno, tedy 0?

  JNZ enter_region     ; Pokud nebyla hodnota odemceno (0)
                       ; tak to zkus znova, skok na návěští enter_region
                       ; točení se nad zámkem

  RET                  ; zámek je náš
```

POUŽITÍ TSL PRO SPIN_UNLOCK

```
leave_region:       ; funkce spin unlock
  MOVE flag, #0     ; nastavime na odemceno
  RET
```

TSL - C

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

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

Viz https://gcc.gnu.org/onlinedocs/gcc/_005f_005fsync-Builtins.html

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

```
monitor hospoda {
  int piv;                // počet piv na skladě
  int vin;                // počet vín na skladě
 condition c1;            // čekáme na pivo
 condition c2;            // čekáme na víno

  void get_pivo();   // zákazník chce pivo
  void get_vino();       // zákazník chce víno

  void zavoz(int zpiv, int zvin);
              // závozník dovezl daný počet piv a vín
}
```

MONITOR HOSPODA - ZÁKAZNÍK

```
void get_pivo() {
   if (piv < 1)
      wait(c1);
    piv --;
}
```

Lepší je použití while místo if
„dvakrát měř, jednou řež“
po vzbuzení otestovat, jestli je podmínka opravdu splněna
Pokud závozník používá broadcast, tak je while nutností.

MONITOR HOSPODA – VYLEPŠENÍ – ZÁKAZNÍK

```
void get_pivo() {
  while (piv < 1)
     wait(c1);
  piv --;
}
```

MONITOR HOSPODA - ZÁVOZ

```
void zavoz(int zpiv, int zvin) {
  int i,j;

  piv = piv + zpiv;
  for (i=0;i<zpiv;i++)  signal(c1);

 vin = vin + zvin;
 for (j=0;j<zvin;j++) signal(c2);

}
```

MONITOR HOSPODA – ZÁVOZ - BROADCAST

```
void zavoz(int zpiv, int zvin) {
  int i,j;

  piv = piv + zpiv;
  broadcast(c1);

 vin = vin + zvin;
 broadcast(c2);

}
```

MONITOR HOSPODA

- Funkce zákazníka:
  getpivo(), getvino()

- Funkce závozníka:
  zavoz(int piv, int vin)

- Zákazník volá getpivo()
  - Není-li pivo dostupné, blokuje nad podmínkovou proměnnou

- Závozník kromě zvýšení proměnné počtu piv a vin signalizuje závoz piva – přivezeli 100 piv, 100x signalizuje signal(c1)

OTÁZKY K OPAKOVÁNÍ

- Co je to systémové volání?
- Co je to přerušení?
- Jak se přerušení využije pro systémové volání?
- Co jsou to výjimky (vnitřní přerušení). Uveďte příklad.
- Co to je a k čemu slouží tabulka vektorů přerušení?
- Linux, Windows jaký typ OS (monolit, hybrid, mikrojádro)?
- Co je to souběh?
- V čem je obtížnost odhalení souběhu?
- Jak jej lze ošetřit?
- Jaký je rozdíl mezi binárním semaforem a mutexem?

OTÁZKY K OPAKOVÁNÍ

- Jaká je výhoda a nevýhoda aktivního čekání?
- Jak byste způsobili deadlock pomocí semaforu při ošetření kritické sekce?
- Znázorněte mechanismus dvojího kopírování při předávání zprávy mezi procesy
- Jaké jsou výhody a nevýhody randez-vous při předávání zpráv?
- Co je to RPC?
- Vysvětlete účel tabulky procesů?
- Co znamená PCB?

SYSTÉMOVÉ VOLÁNÍ

Volání služby jádra z uživatelského kódu

fork()

- Uživateli stačí #include<unistd.h> a zavolat fci fork()
- Viz man 2 fork

Realizace systémového volání (2 možnosti):

- SW přerušení
  - EAX <- číslo služby, INT 0x80
- Speciální instrukce
  - sysenter
  - Sama zajistí přepnutí do režimu jádra
  - Na závěr obsluhy přerušení se vykoná instrukce sysexit

PŘERUŠENÍ

- Přerušení = událost, kterou je potřeba obsloužit
- Procesor přeruší vykonávání sledu instrukcí, vykoná obsluhu přerušení a pak pokračuje v předchozí činnosti

1. SW přerušení
   - Instrukcí INT, např. INT 0x80
   - Realizace systémového volání pomocí SW přerušení
2. HW přerušení
   - Např. stisk tlačítka na klávesnici
   - HW si žádá pozornost (obsluhu) – informuje řadiče přerušení – ten informuje procesor
3. Vnitřní přerušení
   - Výjimky (dělení nulou, neplatná instrukce, výpadek stránky paměti)