Übungsstunde 6

Programm für heute

- Assignment6 besprechen
- Semaphoren repetieren
  - Semaphoren
  - Semaphoren und Monitore
- Semaphore als Monitor (bsp. Vorlesung)
- Classroom Exercise
- Assignment7 vorbesprechen

Assignment6 nachbesprechen

Semaphoren

Semaphoren

- Wurden 1968 von Dijkstra eingeführt
- Nicht-negative Integer Variable mit 2 atomischen Operationen
  - P() (*Passeren*, wait/up)
  - V() (*Vrijgeven/Verhogen*, signal/down*)*

Klasse Semaphore

```
public class Semaphore {
   private int value;
   public Semaphore() {
            value = 0;
    }
   public Semaphore(int k) {
            value = k;
    }
   public synchronized void P() { /* see next slide */ }
   public synchronized void V() { /* see next slide */ }
}
```

P() Operation

```
public synchronized void P() {
    while (value == 0) {
      try {
        wait();
      }
      catch (InterruptedException e) { }
    }
    value --;
  }
```

V() Operation

```
public synchronized void V() {
    ++value;
    notifyAll();
  }
```

Wichtige Eigenschaften

- Nur p() und v() verwenden um Wert von Semaphore zu verändern (abgesehen von Konstruktor)
  - P() kann blockieren
  - V() blockiert NIE
- Verwendung:
  - Mutual Exclusion
  - Conditional Synchronization

2 Typen von Semaphoren

- Binäres Semaphore
  - Ist entweder 0 oder 1
  - Verwendung: mutual exclusion

    Semaphore s = new Semaphore(1);

    s.p()

    //critical section

    s.v()
- Zählendes Semaphore
  - Kann jeden nicht-negativen Wert annehmen

Fairness

- Ein Semaphor wird als fair bezeichnet, wenn es sicherstellt, dass kein wartender Prozess/Thread verhungern kann (Starvation).

Semaphoren in Java

- *java.util.concurrent.Semaphore*
- *aquire()* statt P()
- *release()* statt V()
- Konstruktor: *Semaphore(int permits, boolean fair);*
  - erster Parameter gibt Anzahl erlaubter paralleler Zugriffe an
  - zweiter Parameter gibt an, ob das Semaphor fair ist

Semaphoren und Monitore

- Monitor kann als Semaphore verwendet werden
  - Haben wir bereits gesehen
- Semaphore kann als Monitor verwendet werden

→Sind gleich mächtig/äquivalent

Semaphore als Monitor

Buffer using condition queues

```
class BoundedBuffer extends Buffer {
  public BoundedBuffer(int size) {
       super(size);
  }
  public synchronized void insert(Object o)
         throws InterruptedException {
    while (isFull())
        wait();
   doInsert(o);
     notifyAll();
  }// insert
```

Buffer using condition queues, continued

```
// in class BoundedBuffer
public synchronized Object extract()
            throws InterruptedException {
  while (isEmpty())
            wait();
  Object o = doExtract();
  notifyAll();
  return o;
 }// extract

} // BoundedBuffer
```

Emulation of monitor w/ semaphores

- We need 2 semaphores, access (S) and cond (SCond)
  - Counter a_c to count number of waiting threads

- Two issues
  - Frame all synchronized methods with S.p() and S.v()
  - Translate wait() and notifyAll()

Buffer with auxiliary fields

```
class BoundedBuffer extends Buffer {
  public BoundedBuffer(int size) {
      super(size);
    access = new Semaphore(1);
    cond = new Semaphore(0);
  }
  private Semaphore access;
                              // with p() and v()
  private Semaphore cond;
  private int a_c = 0;
  // continued
```

Framing all methods

```
public void insert(Object o) throws
   InterruptedException {
    access.p(); //ensure mutual exclusion
    while (isFull())
          wait();
      doInsert(o);
      notifyAll();
    access.v();
  }// insert
```

Translate “wait()”

```
public void insert(Object o)    throws InterruptedException {
    access.p();
    while (isFull()) {
          a_c ++;
          access.v(); // let other thread access object
          cond.p(); // wait for change of state
          a_c --;
     }
    doInsert(o);
    notifyAll();
     access.v()
 }// insert
```

Translate “notifyAll()”

```
public void insert(Object o) throws InterruptedException {
    access.p();
    while (isFull()) {
          a_c ++;
          access.v(); // let other thread access object
          cond.p(); // wait for change of state
          a_c --; }
    doInsert(o);
    if (a_c > 0) {
       cond.v(); // resuming thread “keeps” semaphore
                          // must later increment
    }
    else access.v(); //nobody waiting so increment
  }// insert
```

Exercise

- How would the method “extract” look like (if we used semaphores to emulate monitors)?

Classroom Exercise

Assignment7

Um was geht es?

- Implementierung eines Read/Write Lock
- Max. 4 Threads
- Max. 2 Reader Threads (Zugriff auf Objekt geteilt) und 1 Writer Thread
- Reader durch Operation read() bestimmt, kann später auch Writer werden

Einschränkungen

- Keine Starvation
- Effiziente Implementierung
  - Wenn weniger als 2 Readers aktiv sind muss dem nächsten Reader der Zugriff erlaubt sein
- Reihenfolge der Anfragen muss eingehalten werden (verwende FIFOQueue.java)
- In Abwesenheit einer Konkurrenzsituation (Contention) muss sofortiger Zugriff gewährt sein

Wie implementieren?

- So einfach wie möglich
  - Mit synchronized Methoden
  - Mit Semaphoren
    - http://java.sun.com/j2se/1.5.0/docs/api/

    →Semaphore
- Bitte verändert nur die Klasse Monitor.java
  - Falls ihr in einer anderen Klasse etwas ändert, bitte in der Monitor Klasse oben vermerken! Danke ☺
- Bitte kommentiert euren Code!

Evtl. nützlich

- Thread.currentThread().getId()
  - wait_list.enq(Thread.*currentThread().getId())*
  - wait_list.getFirstItem() == Thread.*currentThread().getId()*

So sollte es etwa aussehen…

READ LOCK ACQUIRED 1
READ LOCK RELEASED 0
WRITE LOCK ACQUIRED 1
WRITE LOCK RELEASED 0
READ LOCK ACQUIRED 1
READ LOCK ACQUIRED 2
READ LOCK RELEASED 1
READ LOCK ACQUIRED 2
READ LOCK RELEASED 1
READ LOCK ACQUIRED 2