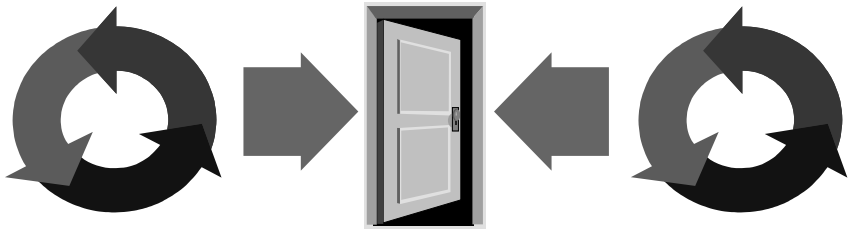
(Chapter 5.2)

Monitors
to Implement Semaphores



Monitors & Condition Synchronization

Concepts: monitors:
encapsulated data + access procedures
mutual exclusion of access procedure
-> single access procedure active in monitor
+ condition synchronization

Practice: private data and synchronized methods (exclusion).
wait(), notify() and notifyAll() for condition synch.
single thread active in the monitor at a time

Monitors, Basic Principles

Active entities (that initiate actions) -> **threads**.
Passive entities (that respond to actions) -> **monitors**.

Mutual exclusion of access procedures
For each monitor exists an exclusion lock.

To ***enter*** the monitor,
a thread acquires the mutual exclusion lock

To ***exit*** the monitor,
a thread releases the lock, and therefore the monitor,
for other threads.

Semaphores

Semaphores are widely used for dealing with inter-process synchronization in operating systems. Semaphore ***s*** is an integer variable that can take only non-negative values.

The only operations permitted on ***s*** are *up(s)* and *down(s)*. Blocked processes are held in a FIFO queue.

***down(s)*: if** *s* >0
then decrement *s*
else block execution of calling process
endif

***up(s)*:** **if** processes blocked on *s*
then awaken one of them
else increment *s*
endif

Semaphores

How to model semaphores ?

using Petri nets

How to implement semaphores ?

using Java's condition synchronization by

wait(), notifyAll

notify()

Condition Synchronization in Java

Java provides a thread **wait queue** per monitor object with the following methods:

`public final void notify()`

Wakes up a single thread that is waiting on this object's queue.

`public final void notifyAll()`

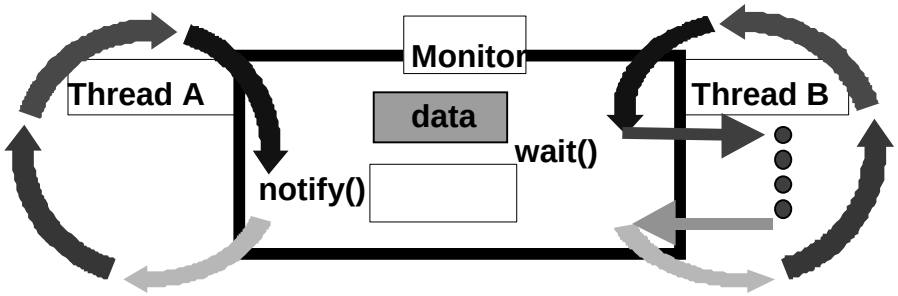
Wakes up all threads that are waiting on this object's queue.

`public final void wait()`
`throws InterruptedException`

Waits to be notified by another thread. The waiting thread releases the synchronization lock associated with the monitor. When notified, the thread must wait to reacquire the monitor before resuming execution.

condition synchronization in Java

Wait() - causes the thread to exit the monitor, permitting other threads to enter the monitor.



Semaphores in Java

Semaphores are passive objects, therefore implemented as **monitors**.

(In practice, semaphores are a low-level mechanism often used in implementing the higher-level monitor construct.)

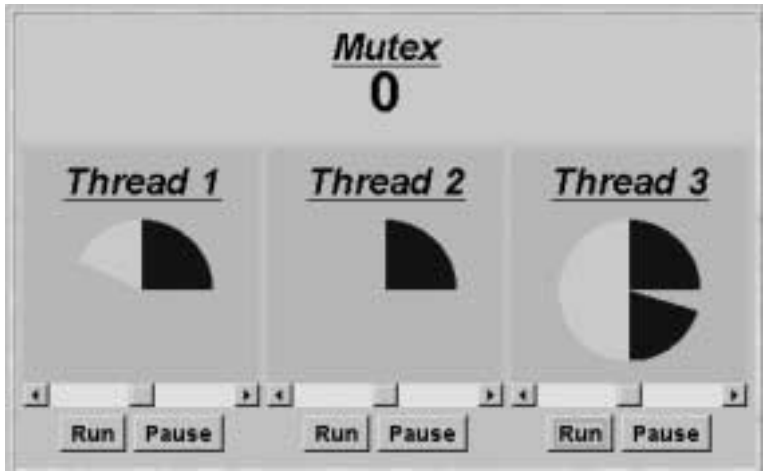
```
public class Semaphore {
  private int value;

  public Semaphore (int initial)
    {value = initial;}

  synchronized public void up() {
     ++value;
     notify();
  }

  synchronized public void down()
      throws InterruptedException {
    while (value == 0) wait();
    --value;
  }
} // Semaphore
```

SEMADEMO display



current semaphore value

thread 1 is executing critical actions.

thread 2 is blocked waiting.

thread 3 is executing non-critical actions.

SEMADEMO program - MutexLoop

```
class MutexLoop implements Runnable {
  Semaphore mutex;

  MutexLoop (Semaphore sema) {mutex=sema;}

  public void run() {
    try {
      while(true)  {
        while(!ThreadPanel.rotate());
        mutex.down();          // get mutual exclusion
        while(ThreadPanel.rotate()); //critical actions
        mutex.up();            //release mutual exclusion
      }
    } catch(InterruptedException e){}
  }
}
```

Threads and semaphore are created by the applet **`start()`** method.

`ThreadPanel.rotate()` returns **`false`** while executing non-critical actions (dark color) and **`true`** otherwise.

Monitors, Summary

Active entities (that initiate actions) are implemented as **threads**.
Passive entities (that respond to actions) are implemented as **monitors**.

Each guarded action in a monitor is implemented as a **`synchronized`** method which uses a while loop and **`wait()`** to implement the guard. The while loop condition is the negation of the guard condition.

Changes in the state of the monitor are signaled to waiting threads using **`notify()`** or **`notifyAll()`.**

Summary

Concepts

- **monitors: encapsulated data + access procedure**
- **mutual exclusion + condition synchronization**

Practice

- **private data and synchronized methods in Java**
- **wait(), notify() and notifyAll() for condition synchronization**
- **single thread active in the monitor at a time**