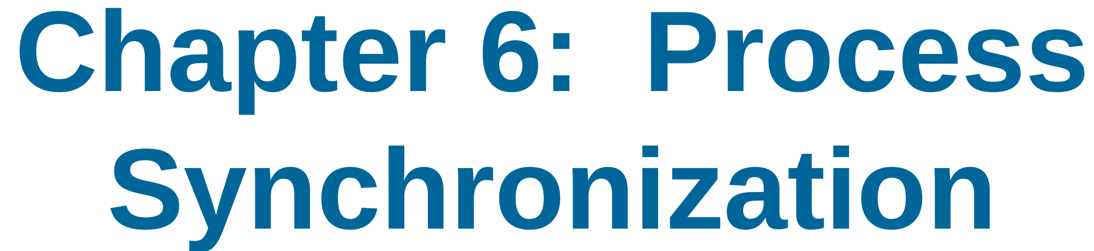


Silberschatz, Galvin and Gagne ©2009

Administrivia

- In lab Friday.
- New assignment.
- Exam: Monday, April 6, on Chapters 1--6.
- Read 8.1—8.5 for Monday.

Outline

- Problems with semaphores.
- Monitors and examples.
- Synchronization facilities in operating environments.

Problems with Semaphores

- Correct use of semaphore operations:
  - signal (mutex) …. wait (mutex)
  - wait (mutex) … wait (mutex)
  - Omitting of wait (mutex) or signal (mutex) (or both)

Monitors

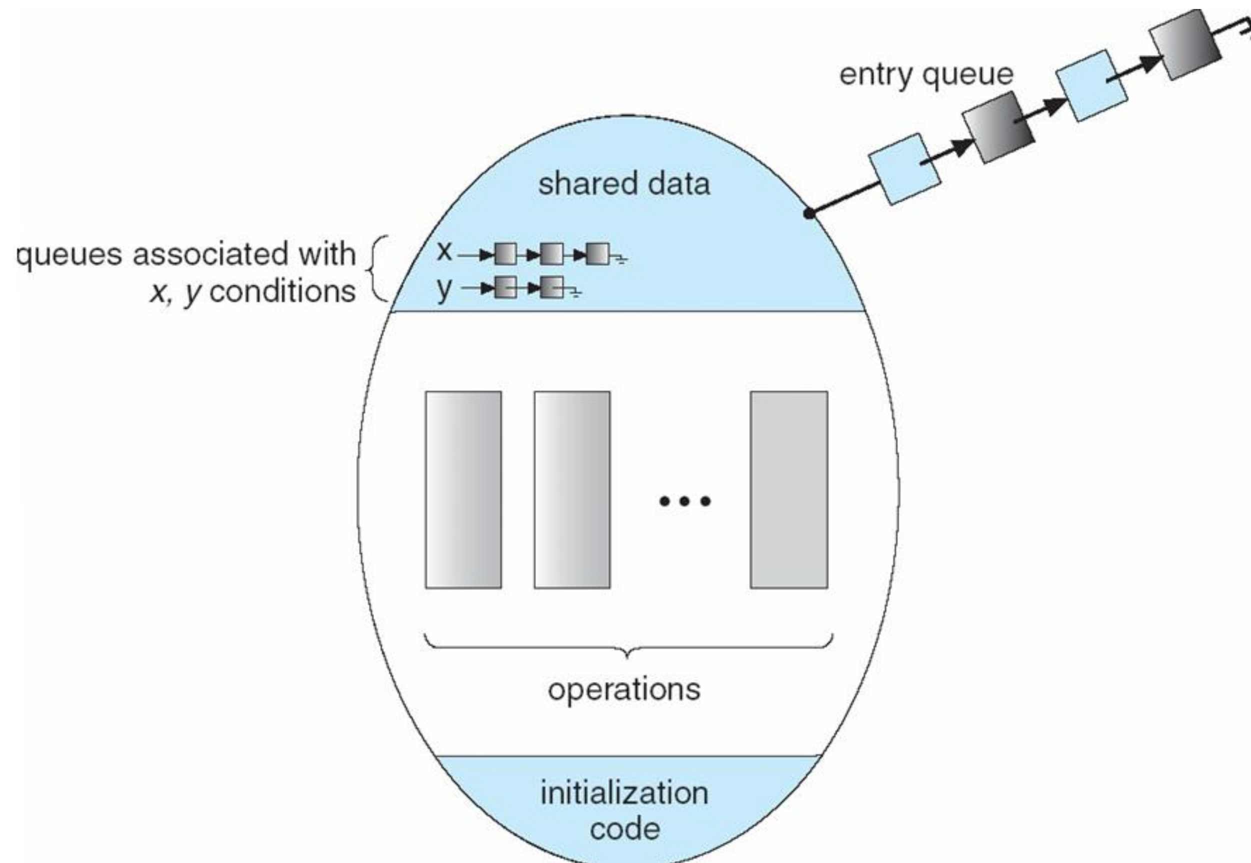
- A high-level abstraction that provides a convenient and effective mechanism for process synchronization
- Only one process may be active within the monitor at a time

```
monitor monitor-name
{
    // shared variable declarations
    procedure P1 (…) { …. }
      …

    procedure Pn (…) {……}

     Initialization code ( ….) { … }
      …
    }
}
```

Condition Variables

- condition x, y;
- Two operations on a condition variable:
  - x.wait () – a process that invokes the operation is suspended.
  - x.signal () – resumes one of processes (if any) that invoked x.wait ()

Monitor with Condition Variables



Solution to Dining Philosophers

```
monitor DP
  {
    enum { THINKING; HUNGRY, EATING) state [5] ;
    condition self [5];

    void pickup (int i) {
           state[i] = HUNGRY;
           test(i);
           if (state[i] != EATING) self [i].wait;
    }

     void putdown (int i) {
           state[i] = THINKING;
                 // test left and right neighbors
            test((i + 4) % 5);
            test((i + 1) % 5);
      }
```

Solution to Dining Philosophers (cont)

```
void test (int i) {
        if ( (state[(i + 4) % 5] != EATING) &&
        (state[i] == HUNGRY) &&
        (state[(i + 1) % 5] != EATING) ) {
             state[i] = EATING ;
            self[i].signal () ;
         }
 }

 initialization_code() {
       for (int i = 0; i < 5; i++)
       state[i] = THINKING;
}
}
```

A Monitor to Allocate A Single Resource

```
monitor ResourceAllocator
{
    boolean busy;
    condition x;
    void acquire(int time) {  // time is max usage time
        if (busy)
            x.wait(time);  // wait()'s param used to order the wait queue.
                           // Implements a "shortest time first" priority.
        busy = TRUE;
    }
    void release() {
        busy = FALSE;
        x.signal();
    }
initialization code() {
     busy = FALSE;
    }
}
```

Synchronization Examples

- Java
- Windows XP
- Linux
- Pthreads

Java Synchronization

- Synchronized class methods --- Every Java object has an associated lock.
- If lock is held by another thread, entering thread is queued on entry set.
- Java provides wait() and notify(), similar to wait() and signal().
- Java 5 provides semaphores, condition variables, and mutex locks.

Windows XP Synchronization

- Uses interrupt masks to protect access to global resources on uniprocessor systems
- Uses spinlocks on multiprocessor systems. Threads holding spinlocks never preempted.
- Also provides dispatcher objects which may act as either mutexes or semaphores
- Dispatcher objects may also provide events
  - An event acts much like a condition variable

Linux Synchronization

- Linux:
  - Prior to kernel Version 2.6, kernel was nonpreemptive.
  - Version 2.6 and later, fully preemptive kernel.
- Linux provides:
  - Semaphores.
  - Spinlocks (SMP systems). Uniprocessor systems disable/enable kernel preemption.

Pthreads Synchronization

- Pthreads API is OS-independent
- It provides:
  - mutex locks
  - condition variables
- Non-portable extensions include:
  - read-write locks
  - spin locks

End of Chapter 6