ELEC 377 – Operating System

Week 3 – Class 3

Last Class

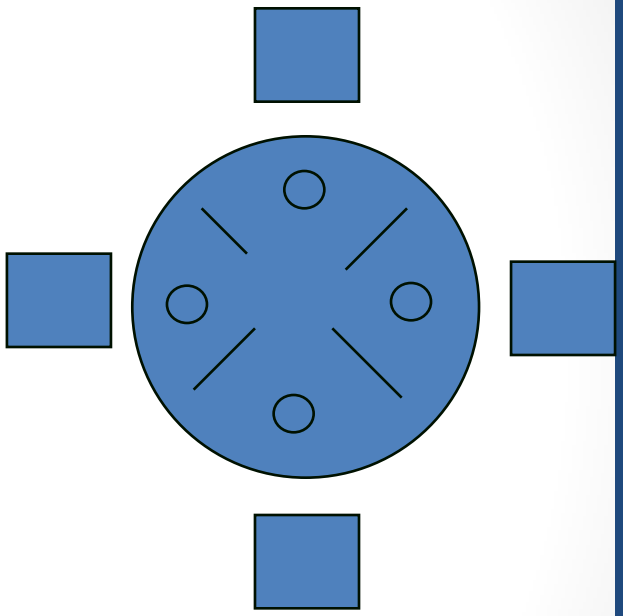
- Classic Problems

Today

- Semaphores
- Classic Problems
- Critical Regions
- Monitors
- Java synchronized keyword

Dining Philosophers

- N philosphers eating rice
- N chopsticks (1 between each philospher)
- Each philosopher needs two chopsticks to eat
- Philosphers alternate between eating and thinking....



Dining Philosophers

```
Sempaphore chopstick[n]; // each init to 1

philosopher(int i){
  do {
    if (i % 2 == 0){  // % integer representation of the remainder
      wait(chopstick[i]);
      wait(chopstick[(i+1)%n];
    } else {
      wait(chopstick[(i+1)%n];
      wait(chopstick[i]);
    }
      // eat
    signal(chopstick[i]);
    signal(chopstick[(i+1)%n];
                          // think
  } while(1)
}
```

Critical Regions

- Semaphores bracket critical sections
- start/end exchanged between cooperating processes
- Simple synchronization has wait(S) in one process and signal(S) in another process.
  - ◊ better than primitive synchronization
  - ◊ Still susceptible to programming errors
- Critical Region is a higher level construct that removes some of the programmer overhead
  - ◊ Higher level construct -> language support!!

Critical Regions

- shared variable is used

```
shared T v;
```

example:

```
shared int v1;

struct xyzzy {
    char * a;
    int b;
 }
shared struct xyzzy v2;
```

Critical Regions

- special language construct to access shared variable

region v when B do S;

◊ B is a boolean condition

◊ S is one or more statements

example:

```
region v1 when (true) do v1++;

region v2 when (v2.b > 0) {
    printf("%s\n",v2.a);
    b--;
}
```

Critical Regions

- Each process has equivalent statement for v but with (possibly) different B and S
  ◊ Only one S can be in execution at a time.

- If B is false, process waits until B is true, then enters S (competes with other processes)

- When a process leaves critical region, all other process that are waiting re-evaluate their B. Before B was false, now it may be true. (or it may have become false!!!)

Today

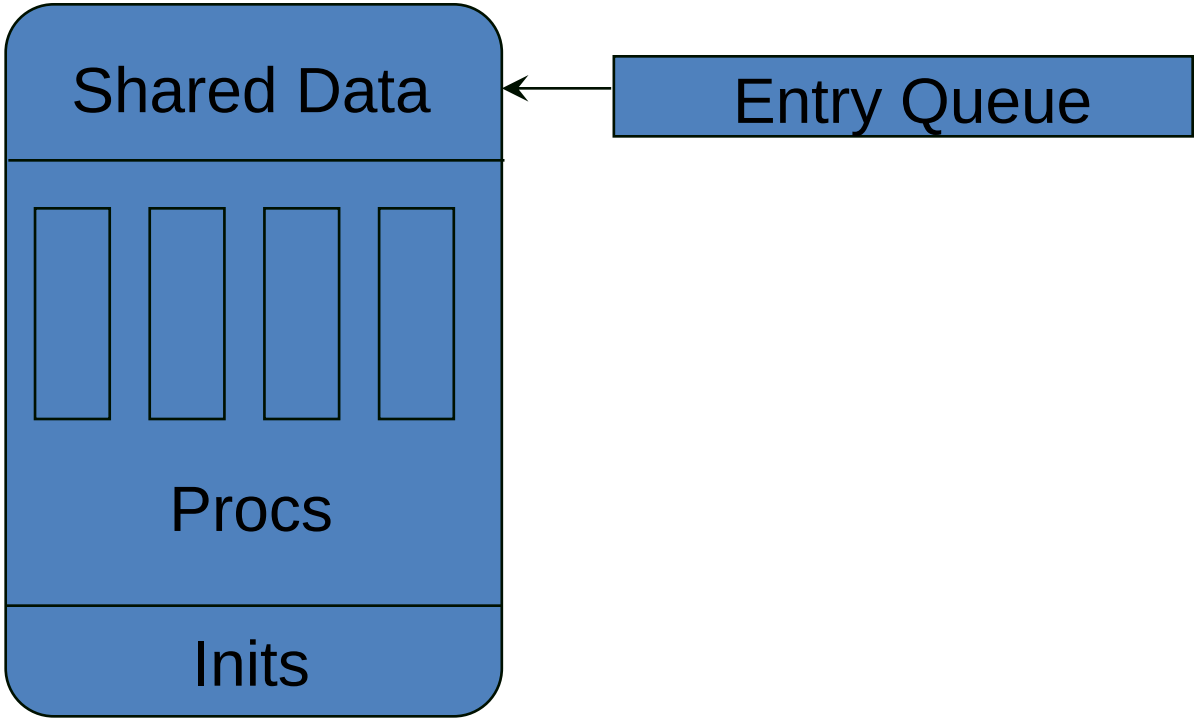
- Semaphores
- Classic Problems
- Critical Regions
- Monitors <<<<<<<<<
- Java synchronized keyword

Monitors

- high level synchronization construct
  - ◊ allows safe sharing of an abstract data type

```
monitor name {
  shared variables
  procedure P1(…){
    …
  }
  procedure P2(…){
    …
  }
  {
    init code
  }
}
```

Monitors

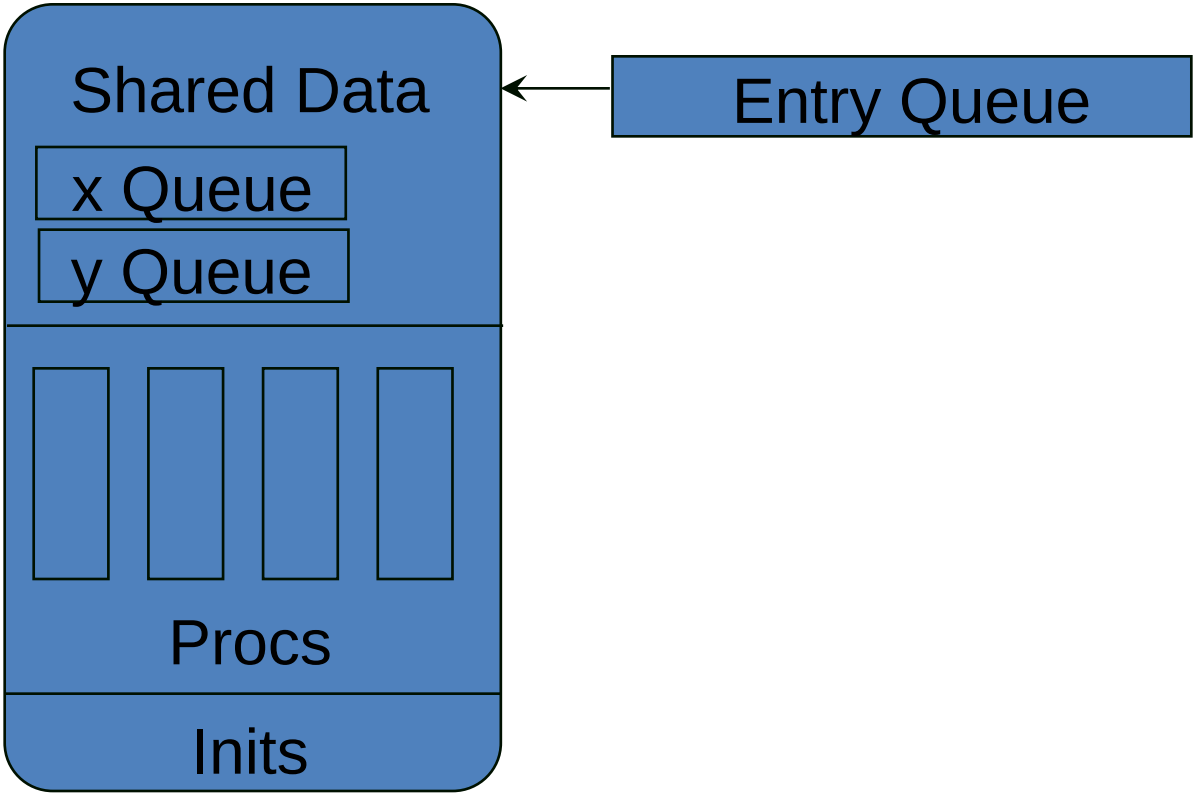


Monitors

- Processes may want to wait for another process
  - e.g. a buffer might be full!
- Condition variable
  - declared in shared variable section, private to monitor

    condition x,y;
  - Two operations, wait and signal
  - x.wait() means go to sleep and yield lock on monitor
  - x.signal() means wake up one process if there is a process that did an x.wait(). A process that did a y.wait() is not affected. If no process are waiting on condition x, then no effect.

Monitors



Signal()

- When a process executes x.signal() and another process is waiting on condition X, what happens?
- Several cases
  - ◊ First process (signaler) goes to sleep until second process exits (releases lock) or waits on another condition
  - ◊ First process continues until it leaves or waits on a condition and then signaled process continues

Producer Consumer

```
monitor buffer {
  condition full, empty;
  procedure add(char X){
    if (buffer is full) full.wait();
    …
    empty.signal();
  }
  char remove(){
    if (buffer is empty) empty.wait();
    …
    full.signal();
  }
  …
}
```

Monitors

- Prioritized waiting
  - x.wait(c) – c is an integer expressions
  - gives priority on queue for X
- System correctness
  - easier than semaphores
  - use monitors to guard shared resources, but not put shared resources inside monitor (may be more than one)
  - must still make sure that process make correct monitor calls incorrect sequence
  - concurrent processing is tricky!!

The two meanings of wait and signal

- Two versions of wait and signal
- Semaphores

Semaphore mutex = 1
wait (mutex)
...
signal(mutex)

- Monitors

Condition x
x.wait()
...
x.signal()

The two meanings of wait and signal

- Semaphores
  integer variable
  user visible value influences operation
- Monitor Condition
  Queue variable
  No user visible value

– wait in a semaphore may go right through (value > 0)
– wait in a monitor always means stop

Today

- Semaphores
- Classic Problems
- Critical Regions
- Monitors
- Java synchronized keyword <<<<<<<

Java Synchronized

- The java synchronized keyword provides high level synchronization

- Two cases:
  - synchronized methods – similar to monitors
  - synchronized blocks – similar to critical regions

Java Synchronized Methods

- synchronized keyword is applied to methods

```
class buffer{
…
public synchronized boolean putX(int x)
…
}
```

- the instance of the class is the shared entity

```
buffer a  = new buffer();
buffer b = new buffer();
```

two processes may not call a.putX() at the same time
they may call a.putX() and b.putX() at the same time

Java Synchronized Blocks

- closer to Critical Regions

```
…
synchronized (x) {
  …
}
…
```

- x must be an object pointer (not integral type)
  - lock is on object given by x
  - other threads with similar synchronized blocks may have different variables, but bound to same object
  - Java only provides a single lock on an object, so a synchronized block with a given object and a synchronized method in class of the object are mutually exclusive

Java Wait() and Notify()

- Yet a third wait, but only two signals
- like wait() and signal() in monitors, but only one (implicit) condition variable.
- Producer/Consumer problem as given uses two condition variables. Can be done with one condition variable.