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

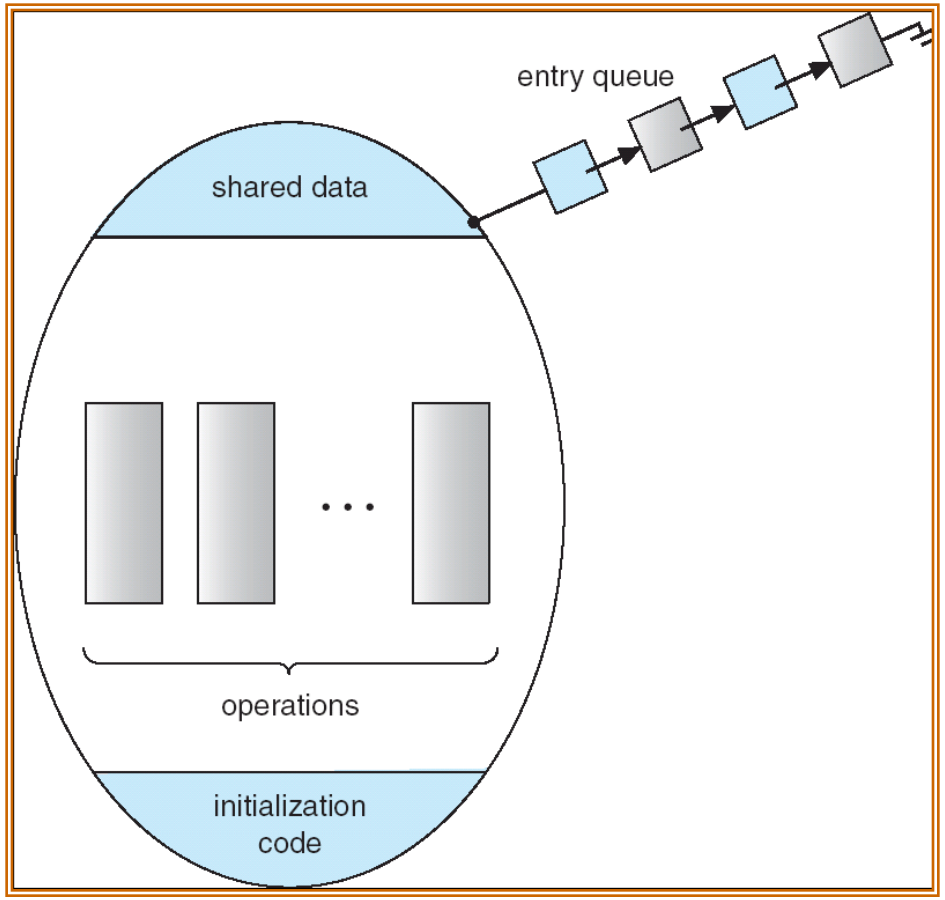
- A high-level abstraction that provides a convenient and effective mechanism for process synchronization
- Only one process may be active within the monitor at a time

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

    procedure Pn (…) {……}

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

Schematic view of a Monitor

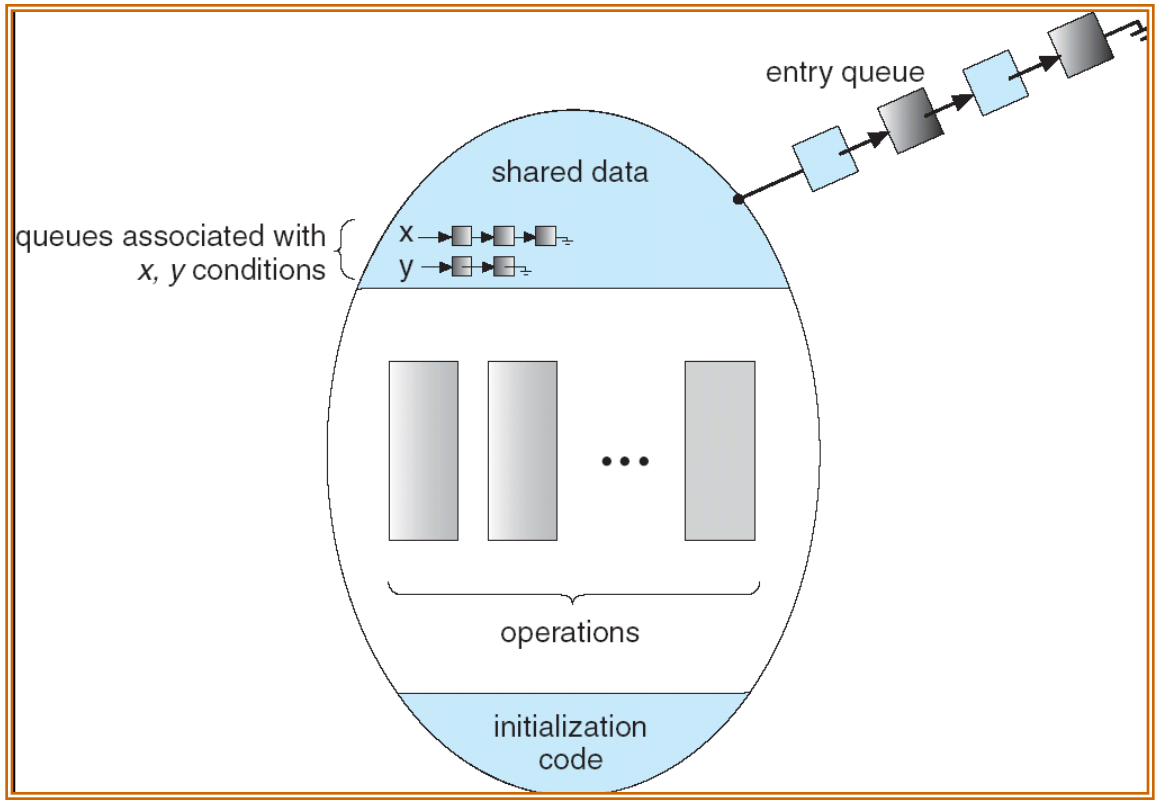


Condition Variables

- 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;
 }
}
```

Solution to Dining Philosophers (cont)

- Each philosopher *I* invokes the operations pickup() and putdown() in the following sequence:

```
dp.pickup (i)

    EAT

dp.putdown (i)
```

Monitor Implementation Using Semaphores

- Variables

```
semaphore mutex;   // (initially  = 1)
semaphore next;    // (initially  = 0)
int next-count = 0;
```

- Each procedure ***F*** will be replaced by

```
wait(mutex);
    …
        body of F;

    …
if (next-count > 0)
   signal(next)
else
   signal(mutex);
```

- Mutual exclusion within a monitor is ensured.

Monitor Implementation

- For each condition variable ***x***, we have:

```
semaphore x-sem; // (initially = 0)
int x-count = 0;
```

- The operation x.wait can be implemented as:

```
x-count++;
if (next-count > 0)
      signal(next);
else
      signal(mutex);
wait(x-sem);
x-count--;
```

Monitor Implementation

- The operation x.signal can be implemented as:

```
if (x-count > 0) {
    next-count++;
    signal(x-sem);
    wait(next);
    next-count--;
}
```