Today’s Plan

Upcoming:
- Quiz 3
- Assignment 2

Today’s topics:
- From last time:
  - Monitor Examples
  - Implementing a Monitor
  - Inter-process Communication

Last time:
- Synchronization Problems
- Monitors
Monitor Implementation Using Semaphores

- Variables

  semaphore mutex;  // (initially  = 1)
  semaphore next;   // (initially  = 0)
  int next_count = 0;

- Each procedure $F$ will be replaced by

  
  
P(mutex);
  
  ...
  
  // body of $F$
  
  ...
  
  if (next_count > 0)
    V(next)
  else
    V(mutex);
Monitor Implementation Using Semaphores

- 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)
    V($next$);
  else
    V($mutex$);
  P($x$ _sem);
  $x$ _count--;
Monitor Implementation Using Semaphores

The operation x.signal() can be implemented as:

```c
if (x_count > 0) {
    next_count++;
    V(x_sem);
    P(next);
    next_count--;
}
```
Inter-Process Communication

What is IPC?

- Sending messages between processes
- A useful process synchronization tool
- Processes can be on the same machine, or on separate machines

E.g., as a critical section solution:
Inter-Process Communication

**Producer:**

While(1) {
    produce one item;
    ask server to enqueue item;
}

**Consumer:**

While(1) {
    ask server for an item;
    consume an item;
}

**Server:**

```c
buffered_items = 0;
while(1) {
    if (request == "get an item") {
        if (buffered_items > 0) {
            dequeue an item;
            return item to the consumer;
            buffered_items--;
        }
        else if (no items buffered)
            queue request in request list;
    }
    else if (request == "enqueue an item") {
        if (request list has consumers) {
            dequeue request from list;
            respond to request with item;
        }
        else {
            enqueue item in the item list;
            buffered_items++;
        }
    }
}
```
IPC Semantics

Several formats of IPC:

- Send/Receive
- Send/Receive/Reply
- SendMessage/WaitMessage/SendAnswer/WaitAnswer

What to consider when designing communications system:

- Any initialization needed before communication starts?
- Can communication be associated with more than 2 processes?
- What size of message can the system handle?
- Is the communication pathway uni-directional, or bi-directional?
Implementation of IPC Semantics

1. Direct Communication
   - Sender explicitly names receiver, and vice versa
   - Communication is available between any two processes, no initialization required
   - Communication is bi-directional

2. Asymmetric Direct Communication
   - Receiver can receive messages from any sender
3. **Indirect Communication**

- Messages are sent and received through mailboxes

![Diagram of间接通信](image)

- E.g. `Send(mailbox_id, msg)`, `Receive(mailbox_id, msg)`

- Allows for more than 2 processes to be associated with the communication
Implementation of IPC Semantics

- What if two processes both receive from the same mailbox?
  - Must establish some sort of delivering policy

- Who owns the mailbox?
  - Can be owned by a process (i.e. receiver)
  - Can be owned by the system