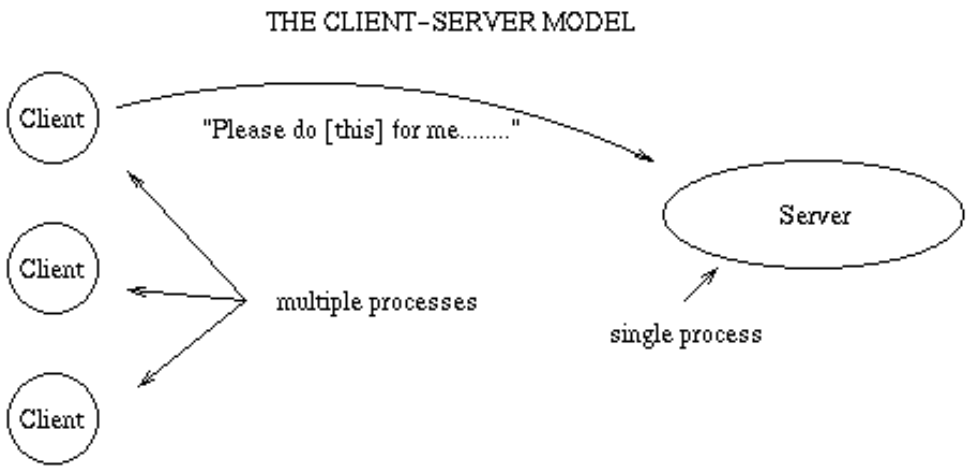
Inter-Process Communication

- 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:
  - place the critical section in a server process

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:
    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;
          }
          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
     - E.g Send(P, msg) and Receive(Q,msg)
   - 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
     - E.g Send(P,msg) and Receive(id, msg)

Implementation of IPC Semantics

3. Indirect Communication

- Messages are sent and received through mailboxes

- 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
    - don't allow it at all
    - Only allow one process to receive at a time
    - FIFO
    - Arbitrary selection
    - Allow multiple receipts of the same message (e.g broadcast)
- Who owns the mailbox?
  - Can be owned by a process (i.e. receiver)
  - Can be owned by the system

Message Capacity of an IPC System

Messages may temporarily be stored in the system waiting to be received, they can be viewed as a linked list of messages.

1. Zero Capacity (no waiting messages)
   - Sender is blocked until receiver receives message
   - Sender and receiver are synchronized at time of message receipt
2. Bounded Capacity (finite # of waiting messages)
   - If queue is not full, message is put on queue and sender continues execution
   - Otherwise, sender is blocked until queue has room
3. Unbounded Capacity (infinite # of waiting messages)
   - Sender never blocks on sending

IPC Message Types

1. Fixed Size Messages
   - Messages are broken up into fragments
   - For small messages →

     entire fixed size message is sent even if message was smaller than that
   - For large messages →

     extra overhead to split message

2. Variable Size Messages
   - Easier for programmer, harder to implement IPC

IPC Message Types – cont…

3. Typed Messages

- Can allow messages to have type information, either implicitly or explicitly
- If communication occurs between two different systems, can run into encoding problem
- Possible solution:

  Have a standard type format e.g XDR