Announcements

- Program #1
  - See additional tips in hot news
  - Due in one week

- Reading
  - Chapter 7 (this entire week)
Interprocess Communication

- Communicating processes establish a link
  - can more than two processes use a link?
  - are links one way or two way?
  - how to establish a link
    - how do processes name other processes to talk to
      - use the process id (signals work this way)
      - use a name in the filesystem (UNIX domain sockets)
      - indirectly via mailboxes (a separate object)

- Use send/receive functions to communicate
  - send(dest, message)
  - receive(dest, message)
Producer-consumer pair

- producer creates data and sends it to the consumer
- consumer read the data and uses it
- examples: compiler and assembler can be used as a producer consumer pair

Buffering
- processes may not produce and consume items one by one
- need a place to store produced items for the consumer
  - called a buffer
- could be fixed size (bounded buffer) or unlimited (unbounded buffer)
Message Passing

- **What happens when a message is sent?**
  - sender blocks waiting for receiver to receive
  - sender blocks until the message is on the wire
  - sender blocks until the OS has a copy of the message
  - sender blocks until the receiver responds to the message
    - sort of like a procedure call
    - could be expanded into a remote procedure call (RPC) system

- **Error cases**
  - a process terminates:
    - receiver could wait forever
    - sender could wait or continue (depending on semantics)
  - a message is lost in transit
    - who detects this? could be OS or the applications

- **Special case: if 2 messages are buffered, drop the older one**
  - useful for real-time info systems
Signals (UNIX)

- provide a way to convey one bit of information between two processes (or OS and a process)
- types of signals:
  - change in the system: window size
  - time has elapsed: alarms
  - error events: segmentation fault
  - I/O events: data ready
- are like interrupts
  - a processes is stopped and a special handler function is called
- a fixed set of signals is normally available
Producer-consumer: shared memory

- Consider the following code for a producer
  ```
  repeat
      ....
      produce an item into nextp
      ...
      while counter == n;
      buffer[in] = nextp;
      in = (in+1) % n;
      counter++;
  until false;
  ```

- Now consider the consumer
  ```
  repeat
      while counter == 0;
      nextc = buffer[out];
      out = (out + 1) % n;
      counter--;
      consume the item in nextc
  until false;
  ```

- Does it work?  Answer: NO!
Problems with the Producer-Consumer Shared Memory Solution

- Consider the three address code for the counter
  
  **Counter Increment**
  
  \[
  \begin{align*}
  \text{reg}_1 &= \text{counter} \\
  \text{reg}_1 &= \text{reg}_1 + 1 \\
  \text{counter} &= \text{reg}_1
  \end{align*}
  \]

  **Counter Decrement**
  
  \[
  \begin{align*}
  \text{reg}_2 &= \text{counter} \\
  \text{reg}_2 &= \text{reg}_2 - 1 \\
  \text{counter} &= \text{reg}_2
  \end{align*}
  \]

- Now consider an ordering of these instructions
  
  \[
  \begin{align*}
  T_0 & \quad \text{producer} \quad \text{reg}_1 &= \text{counter} \quad \{ \text{reg}_1 = 5 \} \\
  T_1 & \quad \text{producer} \quad \text{reg}_1 &= \text{reg}_1 + 1 \quad \{ \text{reg}_1 = 6 \} \\
  T_2 & \quad \text{consumer} \quad \text{reg}_2 &= \text{counter} \quad \{ \text{reg}_2 = 5 \} \\
  T_3 & \quad \text{consumer} \quad \text{reg}_2 &= \text{reg}_2 - 1 \quad \{ \text{reg}_2 = 4 \} \\
  T_4 & \quad \text{producer} \quad \text{counter} &= \text{reg}_1 \quad \{ \text{counter} = 6 \} \\
  T_5 & \quad \text{consumer} \quad \text{counter} &= \text{reg}_2 \quad \{ \text{counter} = 4 \}
  \end{align*}
  \]

This should be 5!