Announcements

- Program #2
  - Info on the Web

- Reading
  - Chapter 7

- Don’t send me email from hotmail or yahoo
  - It’s auto-deleted as SPAM

Cooperating Processes

- Often need to share information between processes
  - information: a shared file
  - computational speedup:
    - break the problem into several tasks that can be run on different processors
    - requires several processors to actually get speedup
  - modularity: separate processes for different functions
    - compiler driver, compiler, assembler, linker
  - convenience:
    - editing, printing, and compiling all at once

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 (un-bounded buffer)

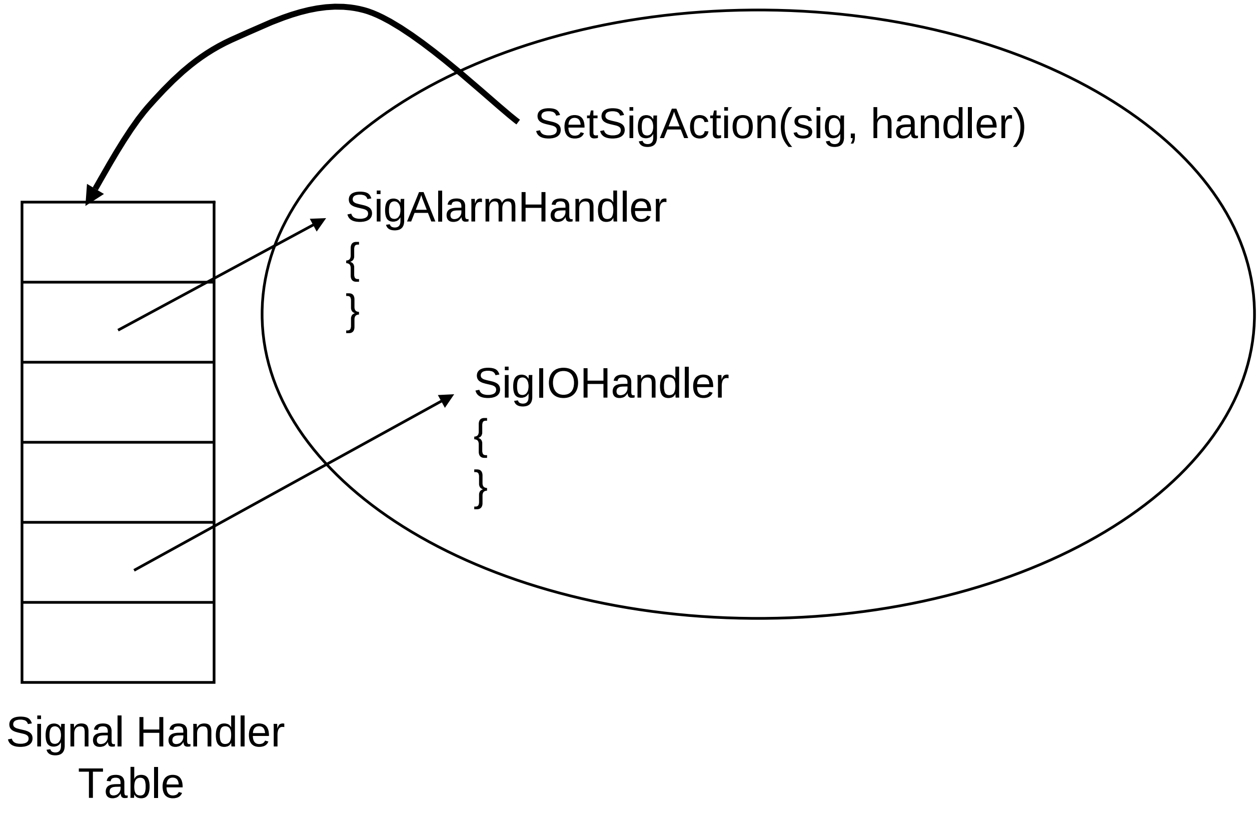
Message Passing

- 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

Signals



SetSigAction(sig, handler)

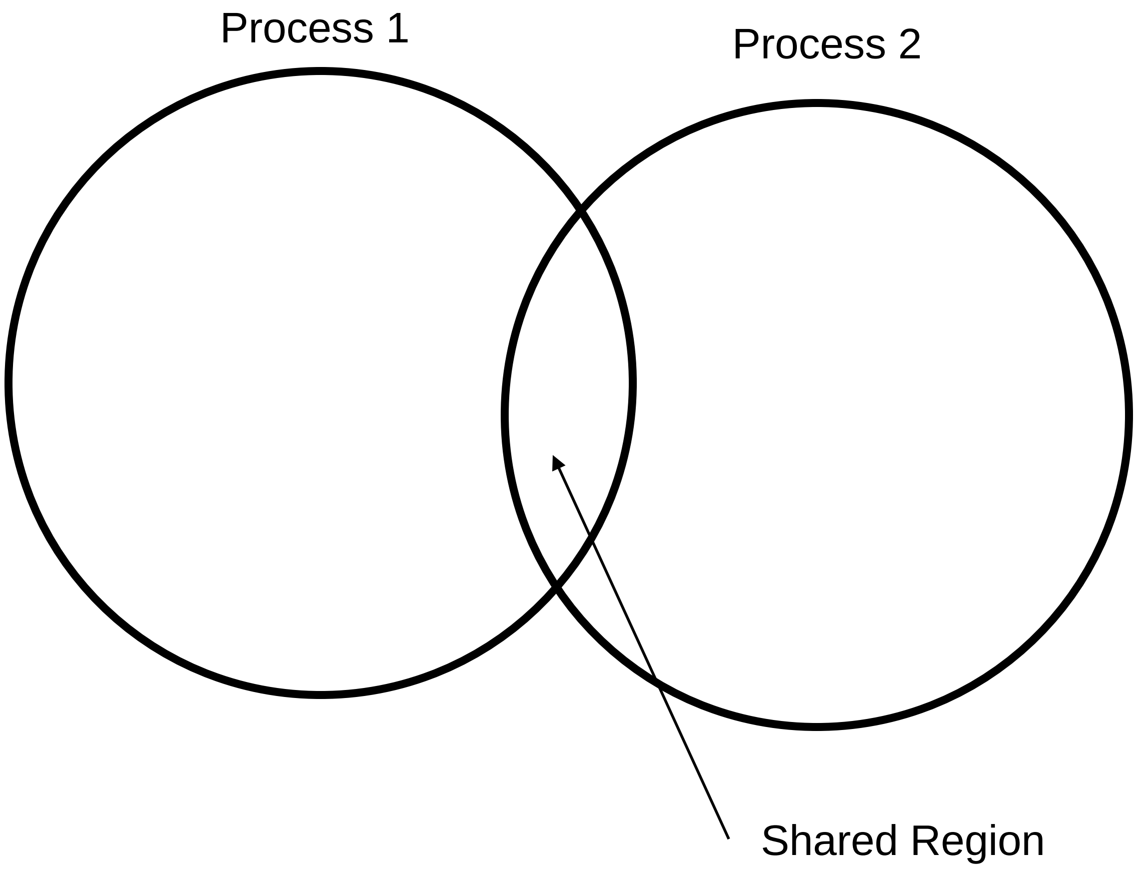
SigAlarmHandler
{
}

SigIOHandler
{
}

Signal Handler
Table

Shared Memory

- Like Threads, but only part of memory shared
- Allows communication without needing kernel action
  - Kernel calls setup shared region



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?
  - NO!

Problems with the Producer-Consumer Shared Memory Solution

- Consider the three address code for the counter

Counter Increment	Counter Decrement
reg_1 = counter	reg_2 = counter
$reg_1 = reg_1 + 1$	$reg_2 = reg_2 - 1$
counter = reg_1	counter = reg_2

- Now consider an ordering of these instructions

T_0	producer	reg_1 = counter	{ $reg_1 = 5$ }
T_1	producer	$reg_1 = reg_1 + 1$	{ $reg_1 = 6$ }
T_2	consumer	reg_2 = counter	{ $reg_2 = 5$ }
T_3	consumer	$reg_2 = reg_2 - 1$	{ $reg_2 = 4$ }
T_4	producer	counter = reg_1	{ counter = 6 }
T_5	consumer	counter = reg_2	{ counter = 4 }

This should be 5!