Announcements

- **Program #0**
  - Due Friday

- **Reading**
  - Threads - Chapter 4 (ch 5, 6th Ed)
Process State Transitions

- new
- ready
- running
- waiting
- terminated

Transitions:
- ready to running: dispatch
- running to ready: interrupt
- ready to waiting: I/O request or event wait
- waiting to ready: I/O request or event wait done
- running to terminated: exit
- ready to terminated: Kill
- new to ready: admitted
Components of a Process

- **Memory Segments**
  - Program - often called the text segment
  - Data - global variables
  - Stack - contains activation records

- **Processor Registers**
  - program counter - next instruction to execute
  - general purpose CPU registers
  - processor status word
    - results of compare operations
  - floating point registers
Process Control Block

- Stores all of the information about a process
- PCB contains
  - process state: new, ready, etc.
  - processor registers
  - Memory Management Information
    - page tables, and limit registers for segments
  - CPU scheduling information
    - process priority
    - pointers to process queues
  - Accounting information
    - time used (and limits)
    - files used
    - program owner
  - I/O status information
    - list of open files
    - pending I/O operations
Storing PCBs

- Need to keep track of the different processes in the system
- Collection of PCBs is called a process table
- How to store the process table?
- First Option:

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ready</td>
<td>Waiting</td>
<td>New</td>
<td>Term</td>
<td>Waiting</td>
<td>Ready</td>
</tr>
</tbody>
</table>

- Problems with Option 1:
  - hard to find processes
  - how to fairly select a process
Queues of Processes

- Store processes in queues based on state
forking a new process

- create a PCB for the new process
  - copy most entries from the parent
  - clear accounting fields
  - buffered pending I/O
  - allocate a pid (process id for the new process)

- allocate memory for it
  - could require copying all of the parents segments
  - however, text segment usually doesn’t change so that could be shared
  - might be able to use memory mapping hardware to help
    - will talk more about this in the memory management part of the class

- add it to the ready queue
Process Termination

- **Process can terminate self**
  - via the exit system call

- **One process can terminate another process**
  - use the kill system call
  - can any process kill any other process?
    - No, that would be bad.
    - Normally an ancestor can terminate a descendant

- **OS kernel can terminate a process**
  - exceeds resource limits
  - tries to perform an illegal operation

- **What if a parent terminates before the child**
  - called an orphan process
  - in UNIX becomes child of the root process
  - in VMS - causes all descendants to be killed
Termination (cont.) - UNIX example

- **Kernel**
  - frees memory used by the process
  - moved process control block to the terminated queue
- **Terminated process**
  - signals parent of its death (SIGCHILD)
  - is called a zombie in UNIX
  - remains around waiting to be reclaimed
- **parent process**
  - wait system call retrieves info about the dead process
    - exit status
    - accounting information
  - signal handler is generally called the reaper
    - since its job is to collect the dead processes
Relationship between Kernel mod and User Mode

User Process

Unique: Program Stack Heap

Initial Thread

Kernel Mode thread of A user process

Kernel Threads:
Each has own stack (separate from user mode)
Share heap with other kernel threads
Run same program (kernel) as other kernel threads

System Calls

Idle Thread

User Process

Unique: Program Stack Heap
Threads

- processes can be a heavy (expensive) object
- threads are like processes but generally a collection of threads will share
  - memory (except stack)
  - open files (and buffered data)
  - signals
- can be user or system level
  - user level: kernel sees one process
    + easy to implement by users
    - I/O management is difficult
    - in a multi-processor can’t get parallelism
  - system level: kernel schedules threads
Important Terms

- **Threads**
  - An execution context sharing an address space
- **Kernel Threads**
  - Threads running with kernel privileges
- **User Threads**
  - Threads running in user space
- **Processes**
  - An execution context with an address space
  - Visible to and scheduled by the kernel
- **Light-Weight Processes**
  - An execution context sharing an address space
  - Visible to and scheduled by the kernel
Dispatcher

- The inner most part of the OS that runs processes
- Responsible for:
  - saving state into PCB when switching to a new process
  - selecting a process to run (from the ready queue)
  - loading state of another process
- Sometimes called the short term scheduler
  - but does more than schedule
- Switching between processes is called context switching
- One of the most time critical parts of the OS
- Almost never can be written completely in a high level language