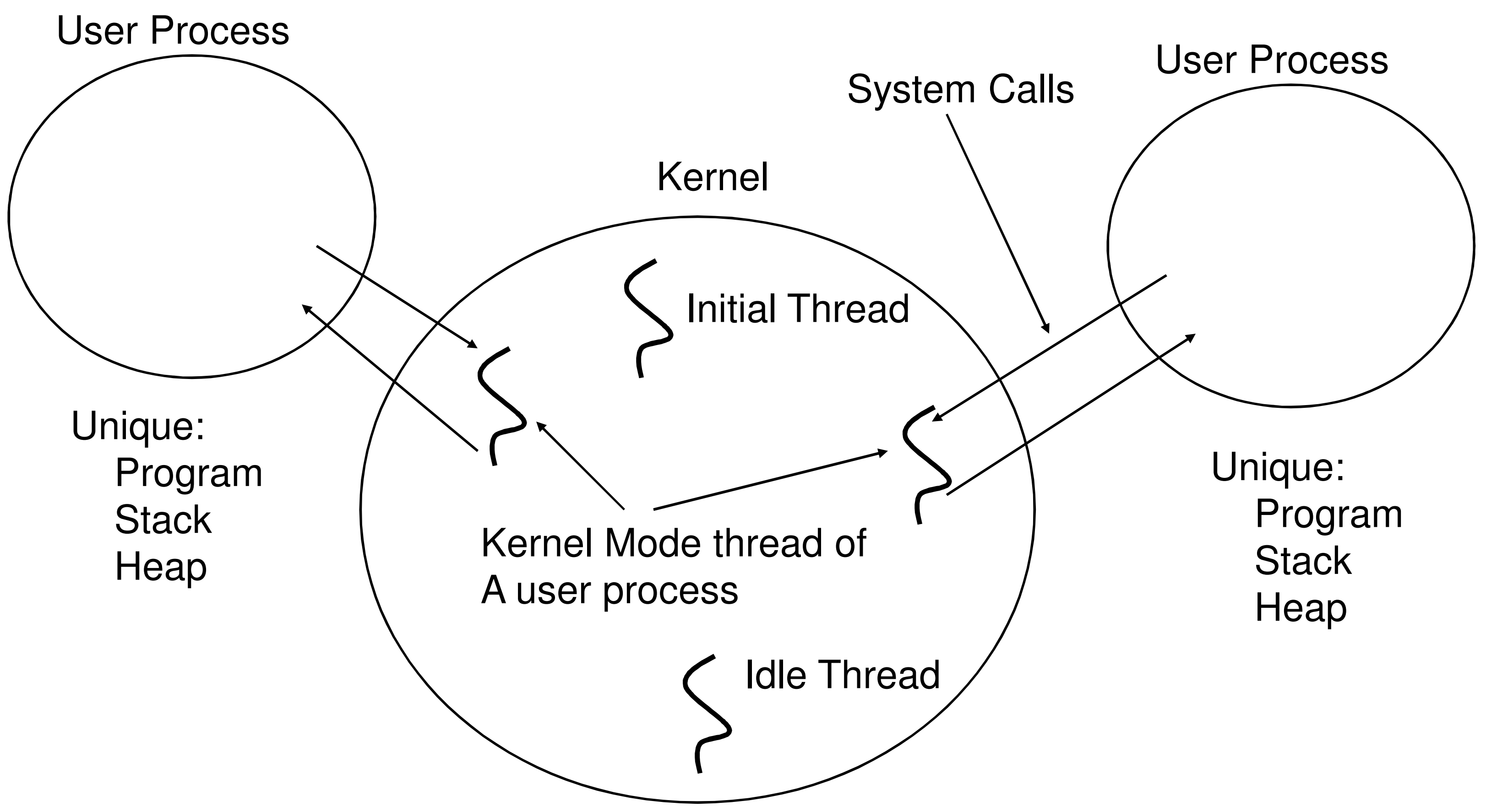
Announcements

- Reading
  - Scheduling
    - Chapter 6 (6th ed) or Chapter 5 (8th ed)

Relationship between Kernel mod and User Mode



Kernel Threads:

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

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 an 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

Selecting a process to run

- called scheduling
- can simply pick the first item in the queue
  - called round-robin scheduling
  - is round-robin scheduling fair?
- can use more complex schemes
  - we will study these in the future
- use alarm interrupts to switch between processes
  - when time is up, a process is put back on the end of the ready queue
  - frequency of these interrupts is an important parameter
    - typically 3-10ms on modern systems
    - need to balance overhead of switching vs. responsiveness

CPU Scheduling

- Manage CPU to achieve several objectives:
  - maximize CPU utilization
  - minimize response time
  - maximize throughput
  - minimize turnaround time
- Multiprogrammed OS
  - multiple processes in executable state at same time
  - scheduling picks the one that will run at any give time (on a uniprocessor)
- Processes use the CPU in bursts
  - may be short or long depending on the job