Announcements

- Program #1
  - Will be on the web shortly
- Reading
  - Chapter 3
  - Chapter 4 (for Thursday)

Project #1

- Much harder than #0
- Adds loading code

System Structure

- Simple Structure (or no structure)
  - any part of the system may use the functionality of the rest of the system
  - MS-DOS (user programs can call low level I/O routines)
- Layered Structure
  - layer n can only see the functionality that layer n-1 exports
  - provides good abstraction from the lower level details
    - new hardware can be added if it provides the interface required of a particular layer
  - system call interface is an example of layering
  - can be slow if there are too many layers
- Hybrid Approach
  - most real systems fall somewhere in the middle

Policy vs. Mechanism

- Policy - what to do
  - users should not be able to read other users files
- Mechanism- how to accomplish the goal
  - file protection properties are checked on open system call
- Want to be able to change policy without having to change mechanism
  - change default file protection
- Extreme examples of each:
  - micro-kernel OS - all mechanism, no policy
  - MACOS - policy and mechanism are bound together

Processes

- What is a process?
  - a program in execution
  - “An execution stream in the context of a particular state”
  - a piece of code along with all the things the code can affect or be affected by.
    - this is a bit too general. It includes all files and transitively all other processes
  - only one thing happens at a time within a process
- What’s not a process?
  - program on a disk - a process is an active object, but a program is just a file

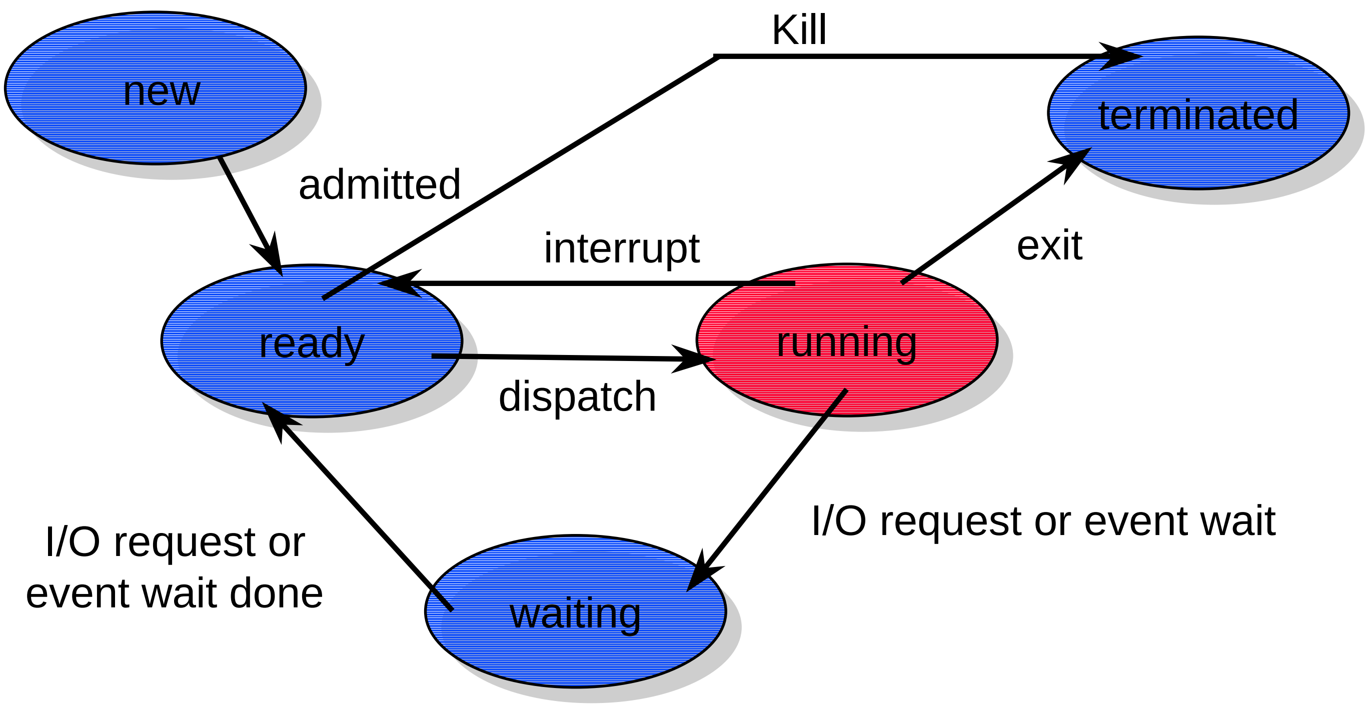
Multi-programming

- Systems that permit more than one process at once
  - virtually all computers today
- Permits more efficient use of resources
  - while one process is waiting another can run
- Provides natural abstraction of different activities
  - windowing system
  - editor
  - mail daemon
- Preemptive vs. non-preemptive muti-programming
  - preemptive means that a process can be forced off the processor by the OS
  - provides processor protection

Process State

- Processes switch between different states based on internal and external events
- Each process is in exactly one state at a time
- Typical States of Processes (varies with OS)
  - New: The process is just being created
  - Running: Instructions are being executed
    - only one process per processor may be running
  - Waiting: The process is waiting for an event to occur
    - examples: I/O events, signals
  - Ready: The process is waiting to be assigned to a processor
  - Terminated: The process has finished execution

Process State Transitions



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

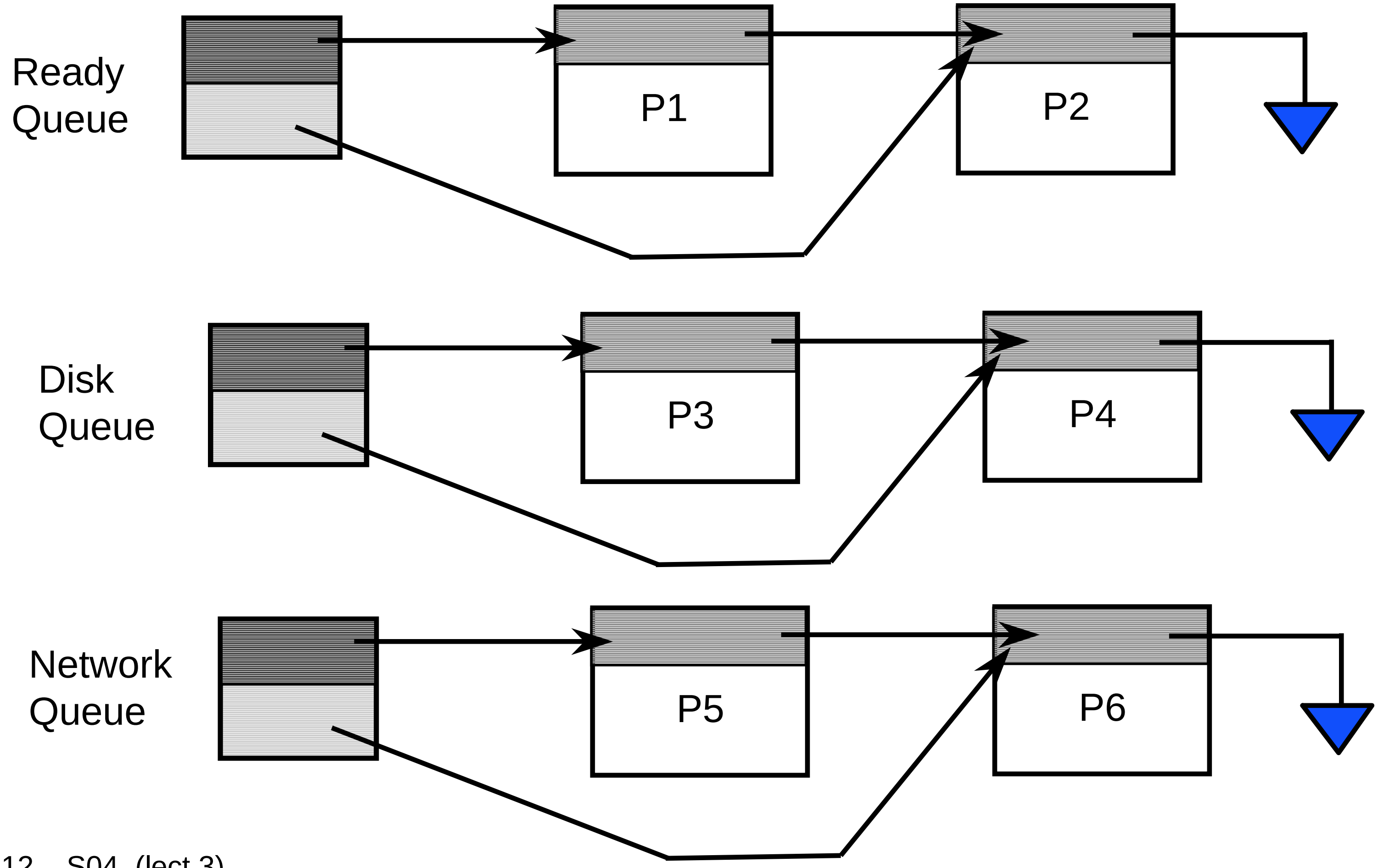
- Need to keep track of the different processes in the system
- Collection of PCBs is called a process table
- How to store the process table?
- First Option:

P1	P2	P2	P3	P4	P5
Ready	Waiting	New	Term	Waiting	Ready

- 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