Announcements

- Project #2 due tonight
- Midterm next Thursday

What is an Operating System?

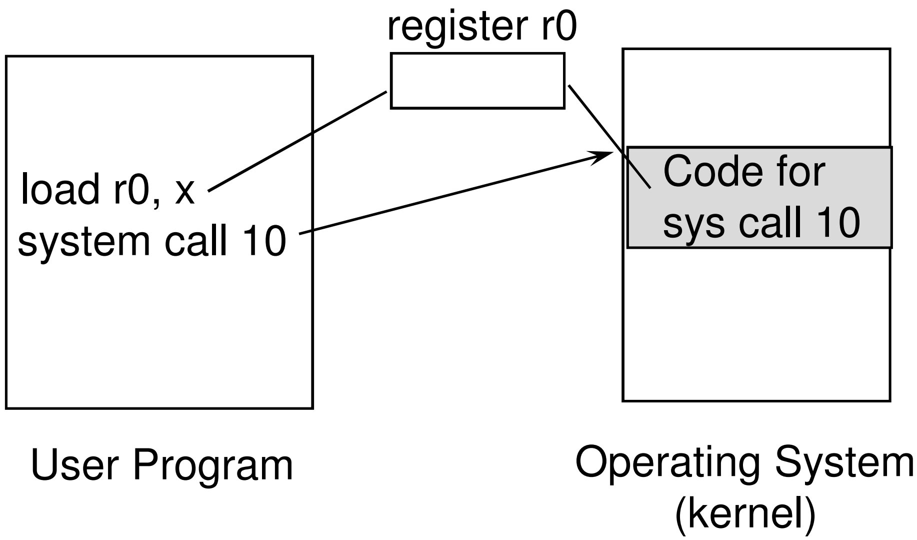
- Resource Manager
  - Resources include: CPU, memory, disk, network
  - OS allocates and de-allocates these resources
- Virtual Machine
  - provides an abstraction of a larger (or just different machine)
  - Examples:
    - Virtual memory - looks like more memory
    - Java - pseudo machine that looks like a stack machine
    - IBM VM - a complete virtual machine (can boot multiple copies of an OS on it)
- Multiplexor
  - allows sharing of resources and protection
  - motivation is cost: consider a $40M supercomputer

What is an OS (cont)?

- Provider of Services
  - includes most of the things in the above definition
  - provide “common” subroutines for the programmer
    - windowing systems
    - memory management
- The software that is always loaded/running
  - generally refers to the Os *kernel.*
    - small protected piece of software
- All of these definitions are correct
  - **but** not all operating have all of these features

System Calls

- Provide the interface between application programs and the kernel
- Are like procedure calls
  - take parameters
  - calling routine waits for response
- Permit application programs to access protected resources



System Call Mechanism

- Use numbers to indicate what call is made
- Parameters are passed in registers or on the stack
- Why do we use indirection of system call numbers rather than directly calling a kernel subroutine?
  - provides protection since the only routines available are those that are export
  - permits changing the size and location of system call implementations without having to re-link application programs

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

Process Creation

- Who creates processes?
  - answer: other processes
  - operations is called fork (or spawn)
  - what about the first process?
- Have a tree of processes
  - parent-child relationship between processes
- what resources does the child get?
  - new resources from the OS
  - a copy of the parent resources
  - a subset of the parent resources
- What program does the child run?
  - a copy of the parent (UNIX fork)
    - a process may change its program (execve call in UNIX)
  - a new program specified at creation (VMS spawn)

Critical Section Problem

- processes must
  - request permission to enter the region
  - notify when leaving the region
- protocol needs to
  - provide mutual exclusion
    - only one process at a time in the critical section
  - ensure progress
    - no process outside a CS may block another process
  - guarantee bounded waiting time
    - limited number of times other processes can enter the critical section while another process is waiting
  - not depend on number or speed of CPUs
    - or other hardware resources
- May assume that some instructions are atomic
  - typically load, store, and test word instructions

Deadlocks

- System contains finite set of resources
  - Process requests resource before using it, must release resource after use
  - Process is in a deadlock state when every process in the set is waiting for an event that can be caused only by another process in the set
- 4 *necessary* deadlock conditions:
  - Mutual exclusion - at least one resource must be held in a non-sharable mode
  - Hold and wait
  - No preemption
  - Circular wait

Deadlock Prevention

- Ensure that one conditions for deadlock never holds
- Hold and wait
  - guarantee that when a process requests a resource, it does not hold any other resources
  - Each process could be allocated all needed resources before beginning execution
- Mutual exclusion
  - Sharable resources
- Circular wait
  - make sure that each process claims all resources in increasing order of resource type enumeration
- No Premption
  - virutalize resources and permit them to be prempted. For example, CPU can be prempted.

Banker's Algorithm

- Each process must declare the maximum number of instances of each resource type it may need
- Maximum cannot exceed resources available to system
- Variables: (n is the number of processes, m is the number of resource types)
  - Available - vector of length m indicating the number of available resources of each type
  - Max - n by m matrix defining the maximum demand of each process
  - Allocation - n by m matrix defining number of resources of each type currently allocated to each process
  - Need: n by m matrix indicating remaining resource needs of each process

Short-term scheduling algorithms

- First-Come, First-Served (FCFS, or FIFO)
  – as process becomes ready, join Ready queue, scheduler always selects process that has been in queue longest
- Round-Robin (RR)
  – use preemption, based on clock - time slicing
- Shortest Process Next (SPN)
  – non-preemptive
  – select process with shortest expected processing time
- Shortest Remaining Time (SRT)
  – preemptive version of SPN
  – scheduler chooses process with shortest expected remaining process time
- Priorities
  – assign each process a priority, and scheduler always chooses process of higher priority over one of lower priority

Synchronization Program

- Have students spend15-20 minutes working on this by themselves before going over it.
- Given an implementation of general (counting) semaphores, implement bounded counting semaphores where each semaphore is declared with initial values, but also a maximum value. A V operation on a bounded counting semaphore that is at its maximum value should return immediately and not change the state of the system.
- P works the same as a general semaphore.
- The API is:
  - CreateBoundedSemaphore(int max, int initialValue)
  - Pbounded(semaphore s)
  - Vbounded(semaphore s)

Solution

```
CreateBoundedSemaphore(int max, int initialValue):
        Shared int s.max = max
        Shared int s.curr = initialValue
        Semaphore s.mutex = 1;
        Semaphore s.wait = initialValue;
Pbounded(semaphore s):
        P(s.mutex)
        s.curr—
        V(s.mutex)
        P(s.wait)
Vbounded(semaphore s)
        P(s.mutex)
        If (s.curr < s.max)
          V(s.wait)
          s.curr++              - This should be only if the if is true!
        V(s.mutex)
```