Lecture 1

Operating Systems

- Review Syllabus
  - read the warning about the size of the project
  - make sure you get the 6th edition (or later) of the book
- Class Grades Server
  - Grades.cs.umd.edu
- Program #0 Handout
  - its due in just under one week
  - purpose is to get familiar with the simulator
- Discussion Sections
  - will focus on the project and meet only once a week (W)
- Reading
  - Chapter 1
  - Chapter 2 (for Tuesday)

What is an Operating System?

- Resource Manager
  - Resources include: CPU, memory, disk, network
  - OS allocates and de-allocates these resources
- Virtualizer
  - 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
    - 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

Closely Related to an Operating System

- Hardware
  - OS is managing hardware resources so needs to know about the ugly details of the hardware
    - interrupt vectors
    - page tables
    - I/O registers
  - some features can be implemented either in hardware or the OS
    - Example: page tables on MIPS
- Languages
  - can you write an OS in any language?
    - No: need to be able to explicitly layout data structures to match hardware

OS Related Topics (cont)

- Language Runtime systems
  - memory management requirements
    - explicit heap management
    - garbage collection
    - stack layout
  - concurrency and synchronization
  - calling convention (how are parameters passed)
- Data Structure and Algorithms
  - efficient access to information in an OS
    - for most things need linear time and space
    - for many things want log or constant time

Why Study Operating Systems?

- They are large and complex programs
  - good software engineering examples
- There is no perfect OS
  - too many types of users
    - real-time, desktop, server, etc...
  - many different models and abstractions are possible
    - OS researchers have been termed abstraction merchants
- Many levels of abstraction
  - hardware details: where the bits really go and when
  - high level concepts: deadlock, synchronization

Why Study Operating Systems (cont.)

- Necessity
  - reliability: when the OS is down, computer is down
  - recovery: when the OS goes down it should not take all of your files with it.
- It’s fun
  - the details are interesting (at least I think so :)
  - thinking about concurrency makes you better at writing software for other areas

Usability Goals

- Robustness
  - accept all valid input
  - detect and gracefully handle all invalid input
  - should not be possible to crash the OS
- Consistency
  - same operation should mean the same thing
    - read from a file or a network should look the same
    - a “-” flag should be the same in different commands
  - conventions
    - define the convention
    - **follow the convention when adding new items**

Usability Goals (cont)

- Proportionality
  - simple, common cases are easy and fast
    - good default values
  - complex, rare cases are possible but more complex and slower
    - “rm *” should give a warning
    - formatting the disk should not be on the desktop next to the trash can

Cost Goals

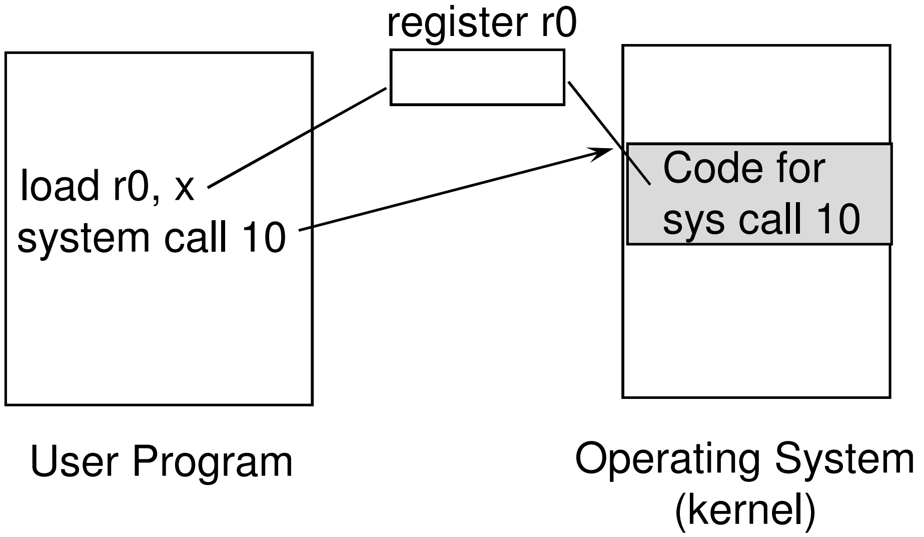
- Good Algorithms
  - time/space tradeoff are important
  - use special hardware where needed
    - smart disk controllers, memory protection
- Low maintenance cost
  - should not require constant attention
- Maintainability
  - most of cost in OS is in maintenance so make it easy to maintain the software base

Adaptability Goals

- Tailored to the environment
  - server vs. workstation vs. mobile
  - multi-media vs. data entry
- Changes over time
  - added memory
  - new devices
- Extensible
  - third parties can add new features
    - database vendors often need custom features
  - end customers can extend the system
    - new devices
    - new policies

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

Types of System Calls

- File Related
  - open, create
  - read, write
  - close, delete
  - get or set file attributes
- Information
  - get time
  - set system data (OS parameters)
  - get process information (id, time used)
- Communication
  - establish a connection
  - send, receive messages
  - terminate a connection
- Process control
  - create/terminate a process (including self)
  - Get/set process meta data (i.e. Limit system call for project #0)