CMSC 412
Operating Systems

Fall 2004

Professor Michael Hicks

Welcome!

- Read the Syllabus
  - read the warning about the size of the project
  - make sure you get the 6^{th} edition (or later) of the book
- Discussion Sections (start Wednesday)
  - focus on the project
  - meet only once a week
    - Probably we will have three times total per week; you can attend the section of your choice. See the web page.

Projects

- The best way to understand is by doing
  - So, we will build an operating system that works on bare hardware by extending the GeekOS academic operating system. 6 projects total.
- Project #0 Handout (posted August 31)
  - It will be due late next week.
  - It will get you familiar with the simulator and the Cyclone programming language, which you can use to write some or all of the project. What is Cyclone? Glad you asked …

Cyclone

- Based on C
  - Low-level control over memory management, data representation, and access to the machine
- But type-safe!
  - Rules out many hard-to-find bugs and security holes
    - Buffer overflows
    - Dangling pointers
  - These bugs have killed many projects

2

Cyclone in this class

- You will be required to use it for project 0, to get familiar with it
- From then on it is optional
  - Could very well make your project more reliable, and prevent many bugs
  - But, it's a research compiler, and it will have bugs itself, and may have cryptic error messages
    - We will do our best to overcome these issues with you. If a major problem in the compiler inhibits a good-faith attempt to complete the project, you will not be penalized.

Cyclone in this class

- Completing at least one project using Cyclone (from project 1 to 6) will net some extra credit
  - You must include a writeup of your experience, including what features you used, what difficulties you ran into, and suggestions for improvement.
- See http://www.cs.umd.edu/projects/cyclone/

3

Class Grades Server

http://grades.cs.umd.edu

- Get your LinuxLab account from here
  - CS computing cluster. Projects must work and be submitted on these machines.
- Complete grade information
- Interface for requesting regrades on exams and projects (with deadline!)

Course Material

- Reading
  - Chapter 1
  - Chapter 2 (for Wednesday)
- Coordination with other section
  - lectures and exams may differ
  - projects will be the same

Why Study OSs?

- Understand computer systems
  - From the hardware to applications
  - Helpful for understanding performance, security, reliability, and other issues

Why Study OSs?

- Understand principles of abstraction
  - OSs are large and complex. How do we manage this complexity?
  - Abstraction:
    - Break each piece into self-contained chunks
    - Plug each piece together
  - Different views of service from view of the provider and view of the user
  - These principles will serve you in other software systems projects
    - E.g., thinking about concurrency

Why Study OSs?

- Understand tradeoffs of system design
  – Many types of users (too many!)
    - real-time, desktop, server, etc...
  – Many possible models and abstractions
    - OS researchers are ‘abstraction merchants’
  – There is no perfect OS!

Why Study OSs?

- It’s fun!
  – the details are interesting (I think so!)

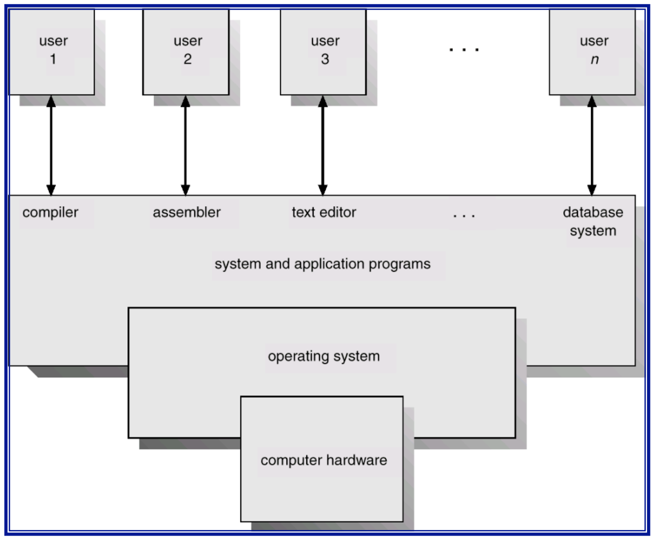
What is an Operating System?

- 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

Abstract View of System



Operating System Cousins

- 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
- Languages
  - can you write an OS in any language?
    - No: need to be able to explicitly layout data structures to match hardware

OS Cousins (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

OS Examples: Batch system

- Mainframes
  - Automate running of user jobs
  - Users submit jobs, OS schedules them
  - Called a batch system
  - OS referred to as a resident monitor

9

Multiprogramming

- Many jobs active "at once"
  - Each job no longer runs to completion before the next can start
  - Many jobs resident in memory at once; the CPU is shared between them
- Job scheduling
  - which jobs to load into memory

Multiprogramming OSs

- Must manage resources
  - Memory
  - I/O devices
  - CPU
  - Want to be fair and efficient
- CPU scheduling
  - which memory-resident program to run
- Protection
  - Prevent errant job from tainting results of another

10

Multitasking (Timesharing)

- Like multiprogramming, but interactive
  - Switching between tasks is very fast
- Preemption: interrupting a job to transfer control to another job
  - Quantum: the time slice allocated to a process before it's preempted. Modern OS typically 50 ms.
- Interactivity requires more support
  - Filesystem, virtual memory, synchronization, deadlock avoidance

Desktop System

- Catered to a single user
- Goal: flexibility and responsiveness
  - Support interactive I/O devices
  - May not need to be as efficient as multi-user OS to be more responsive
  - May be less concerned with protection

11

Parallel System

- Single computer system of several tightly-coupled CPUs
  - All share memory, clock, devices
- Goals:
  - Better value
    - Increased performance
    - Lower cost
  - Increased reliability
- Two flavors
  - symmetric and asymmetric

Distributed System

- Many machines joined by a network
  - Loosely coupled: each processor has its own memory; communication via the network
    - OS often implements network protocols
- Goals
  - Value
  - Share distributed resources
- Variants: client-server, peer-to-peer

12

Real-time Systems

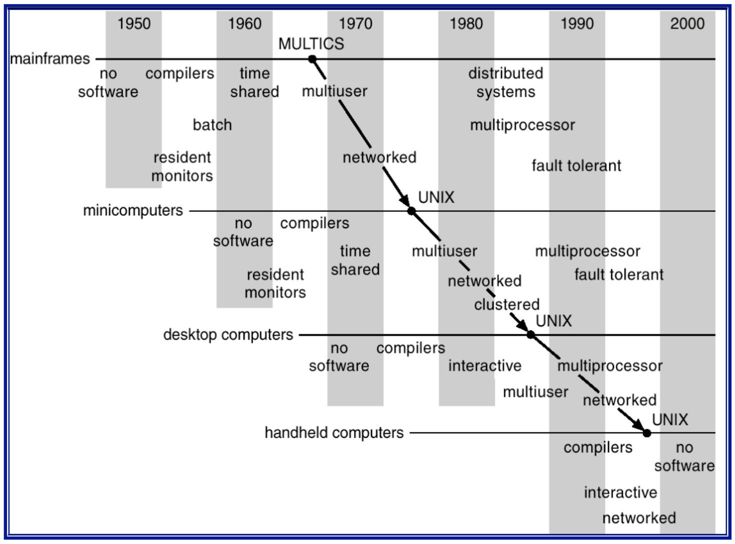
- Control device in a dedicated application
  - Medical imaging, industrial control, etc.
- Well-defined fixed time constraints
- Variants
  - Hard: can never miss a deadline
  - Soft: less stringent. Thus permits unpredictable elements like caches, disks, etc.

Handheld Systems

- PDAs, cell phones, etc.
- Issues
  - Limited memory
  - Slow (power-constrained) processors
  - Small display

13

Migration of OS features



OS Goals: Usability

- 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 of software is in maintenance so make it easy to maintain the software base

Adaptability Goals

- Tailored to the environment
  - server vs. workstation
  - 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

16