Operating Systems and Systems Programming

Professor: Anthony D. Joseph

Spring 2001

Lecture 2: Operating System Structures

0.0 Administrivia:

1.0 Main points:

History of operating systems

Overview of Course Topics

Overview of Main OS "structures"

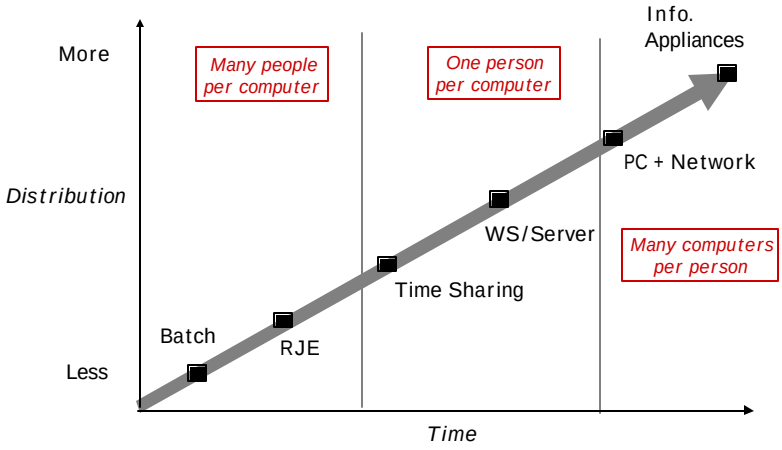
1.1 History of Operating Systems: Change!

One view of change:

Typical academic computer in 1981 and 2001

	1981	2001	factor
CPU MHz, cycles/inst	10 3-4	1500 0.5-1	150 3-8
DRAM capacity	128KB	256MB	2000
Disk capacity	10 MB	75GB	7500
net bandwidth	9600 b/s	155 Mb/s	15000
# address bits	16	64	4
# users/mach.	10s	<= 1	0.1
Price	$25000	$3000	8.3

Another view of change:



What does this mean?

Techniques have to vary over time, adapt to changing tradeoffs.

1.1.1 History Phase 1: hardware expensive, humans cheap (1948 – 1970)

When computers cost millions of $'s, optimize for more efficient use of the hardware! Lack of interaction between user and computer.

- **User at console:** one user at a time.

- **Batch monitor:** load program, run, print.
  Computers are expensive, so optimize to make better use of the hardware. When the user at the console is thinking, he's not using the computer. So instead, feed the computer jobs in batches and make users wait for their results. Makes better use of hardware, but a lot worse for users.

No protection: what if batch program had a bug and wrote over batch monitor?

- **Data channels, interrupts:** overlap of I/O and computation.
  DMA – Direct Memory Access for I/O devices. OS requests I/O, goes back to computing, gets interrupt when I/O device has finished.

- **Multiprogramming**: several programs run at the same time; users share system.
  Multiprogramming benefits:
  1. Small jobs not delayed by large jobs
  2. More overlap between I/O and CPU

Multiprogramming requires memory protection to keep bugs in one program from crashing the system or corrupting other programs.

- **Bad news**: OS must manage all these interactions between programs. Each step seems logical, but at some point, fall off cliff – just gets too complicated.
  - Multics: announced in 1963 -> ran in 1969
  - OS 360 released with 1000 bugs.
  - OS's finally begin to be an important science. How do we deal with this complexity?
    - UNIX based on Multics, but simplified so they could get it to work!

1.1.2 History, Phase 2: hardware cheap, humans expensive (1970-1985)

Computers available for tens of thousands of dollars instead of millions. Can afford to start using them for more kinds of jobs

***Interactive* timesharing:** Use cheap terminals ($1-2K) to let multiple users interact with the system at the same time. Sacrifice CPU time to get better response time for users. Users start doing things like program development and debugging, test editing, and e-mail online.

Problem: thrashing – performance falls off a cliff as you add users.

1.1.3 History, Phase 3: hardware very cheap, humans very expensive (1981-)

Computer costs $1K, Programmer costs $100K/year
If you can make someone 1% more efficient by giving them a computer, it's worth it!
Key challenge is to use computers to make more efficient use of people's time.

Personal computing – Computers are cheap, so give everyone a PC. Initially, PCs had limited hardware resources, so OS became a subroutine library for running one application at a time again (MSDOS). But since then, PCs have become powerful computers and the OS has regained all the complexity of a "big" OS: memory protection, multiprogramming, etc (NT, OS/2).

Question: As hardware gets cheaper and cheaper does the need for some (or all) OS functions go away?

1.1.4 History, Phase 4: Distributed systems (1981-)

Networking: allow different machines to share resources easily (printers, file servers, web servers).

Question: Why are distributed systems needed?

1.1.5 History of Operating Systems: Summary

Point of change isn't: look how stupid batch processing is.
It was right for the tradeoffs of the time – but not anymore.

Point is: have to change with changing technology.

Situation today is much like it was in the late 60's: OS's today are enormous, complex things:

- small OS – 100K lines
- big OS – 10M lines

100-1000 people-years

NT under development since early 90's, still doesn't work very well. Jury is still out for Windows 2000.

Key aspect of this course – understand OS's, so we can simplify them!

1.2 Overview of Course Topics

The course will be divided into three main topics (with exams corresponding roughly to these):

1. Concurrency: processes and threads
2. Memory management, File Systems, and I/O
3. Distributed Systems and Networking

1.3 Operating System Structures

What follows is a brief "tour" of the overall structure of OS's. There are several ways to divide things up and each yields a different kind of understanding about an OS.

1.3.1 System Components (What are the pieces of the OS?)

- Process management
- Main-memory management
- I/O system management
- File management
- Networking
- UI

1.3.2 Operating System Services (what things does the OS do?)

Services that (more-or-less) map onto components:

- Program execution
- I/O operations
- File system manipulation
- Communications

Cross-cutting capabilities:

- Error detection & recovery
- Resource allocation
- Accounting
- Protection

1.3.3 System Calls (what's the API?)

See Chap. 3 of the book for an example list.

1.3.4 System Structure (what's the implementation structure?)

"Simple" structure

- unstructured (e.g. DOS)
- kernel/user (e.g. original Unix)

Layered approach

- Idea: each layer uses only the routines of the layers below it.
- Advantage: modularity => easier debugging and maintenance
- One of the most important layerings: machine-dependent vs portable layer.

Micro-kernel plus server processes

- Small kernel plus user-level servers
- Even more modular and debuggable and maintainable
- Very configurable

1.3.5 System Design Goals (what is this OS trying to achieve?)

- $2000 price point?
- Fault tolerant?
- High performance?
- Real-time capable?

1.3.5.1 Implementation Issues (how is the OS implemented?)

- Mechanism vs. policy
- Algorithms used
- Event models used (threads vs. event loops)
- Backward compatibility issues (Windows 2000 worries a lot about this)
- System generation/configuration (i.e., how to make a generic OS run on a particular piece of hardware)