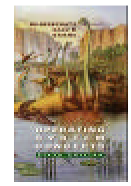
1: Welcome and Overview

Last Modified:
6/1/2004 11:53 AM

-1

Logistics

- Course Web Page
- Mailing List
- Staff
- Textbook



-2

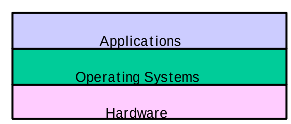
Topics

- OS History, Architectural Support
- Processes, Threads
- Scheduling
- Synchronization, Deadlock
- Memory Management
- File Systems, IO Devices
- Networks, Distributed Systems
- Security

-3

What is an operating system?

- A software layer that
  - manages hardware resources
  - provides an abstraction of the underlying hardware that is easier to program and use



-4

Hardware Resources

- CPU, Functional Units, Registers
- Main memory access
- Storage devices (disk drives, CD-ROMs, tape drives)
- Network Interface Cards
- Human I/O devices (keyboards, monitors, mice)
- Other? Printers, cameras, sensors, …

How much do you know about what it would be like to interact with these devices without an OS?

-5

Benefits of Operating Systems (1)

- Abstracting the Hardware
  - Gory details of the raw hardware hidden so applications can be smaller and simpler
  - Application writers can program to a simpler and more portable “virtual machine”
- Providing useful logical abstractions
  - New types of logical resources (sockets, pipes)

-6

Benefits of Operating Systems (2)

- Protecting applications from each other
  - Enforce "fair" allocation of hardware resources among applications
  - Policies that say what is "fair" and mechanisms to enforce it
- Supporting communication and coordination among applications
  - Support abstractions through which different applications can share data and notify each other of events

-7

What an operating system is not?

- Compiler
- Standard libraries
- Command Shells
- These are closely related pieces of system software, but they are not the OS.

-8

Is OS code like other code?

- Most OSs implemented in C
- Developed without space-age development environments (kernel debuggers?)
- The buck stops here!
  - OS must deal with gory hardware details
    - Try to keep hardware dependent parts isolated
  - What happens when get a device interrupt in the middle of executing an application? What happens when get a device interrupt while servicing another device interrupt?
  - What happens if you take a page fault while executing operating system code
- Performance and reliability are crucial!
- Still a lot more like application code than you might think

-9

Lots of variety of OSes

- Unix (Solaris, HP-UX, AIX, FreeBSD, NetBSD,OpenBSD..)
- Linux
- Windows XP, 2000, NT, ME, 98, 95
- BeOS
- MacOS
- PalmOS
- WindowsCE
- Mach, Amoeba, Sprite, Vino, SPIN, QnX,...

-10

What distinguishes operating systems?

- When people talk about which operating system to run, they often talk about:
  - Look and feel of the desktop windowing system
  - Devices that are supported
  - What hardware platforms does it run on?
  - Applications that are available for that OS
  - Who developed the code? Who supports the code?
  - How often does the system crash? Reliability?
  - Do you pay for it?
- Are these really core OS issues?

-11

Core OS Issues: OS Structure

- How is the OS structured?
  - One monolithic kernel of spaghetti code
  - One monolithic kernel that is internally composed of distinct layers
  - One monolithic kernel that is internally composed of distinct objects
  - Micro-kernel with trusted user level applications that provide major OS functionality like virtual memory, scheduling, file systems, etc.
- Software engineering question
  - Maintainability? Performance? Reliability? Portability?

-12

Core OS Issues

- Concurrency
  - How many and what types of activities can occur simultaneously?
- Protection
  - What is the granularity at which permission to access various resources are granted?
  - How do you verify an entity's right to access a resource?
- Fault Tolerance
  - How do we deal with faults in applications? In devices? In our own OS code?

-13

Core OS Issues

- Resource/services provided to applications
  - Does the OS offer kernel support for events? Signals? Threads? Pipes? Shared memory?
- Naming
  - How do applications refer to and request the resources they want for themselves? Resources they want to share with others?
- Sharing
  - What objects can be shared among applications? What is the granularity of sharing?
- Resource Allocation and Tracking
  - What is the unit (or units) of resource allocation?
  - Can we track (and bill for) resource usage?

-14

Core OS Issues

- Service Time Guarantees
  - What guarantees (if any) are made to applications about the servicing of their requests or about the servicing of device interrupts?
  - Real-time OSs
- Scale/Load
  - What are the limits of resource allocation? (Biggest file, Maximum number of processes, etc.)
  - What happens as the demand for resources increases? (graceful degradation?)

-15

Core OS Issues

- Extensibility /Tuning
  - What interfaces are provided to change operating system behavior?
  - How does (or does) the OS optimize its behavior based on the characteristics of the hardware or the application mix?

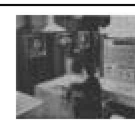
-16

Evolution of Operating Systems

- At first, OS = library of shared code
  - Every programmer did not write code to manage each device
  - Each application when compiled contained the "OS"
  - Load into memory and execute
    - By who? People = Operators
    - How? By mechanical switches at first, then punch cards
  - Just one application at a time so no need for protection and no need for sharing
  - No virtual memory; either the entire program fit into memory or programmers handled moving sections of their own code in and out of memory

-17

Batch Processing



- Still only one application at a time
- Operating system rather than operators loaded one job after another off of punch cards or tape
- OS knew how to read next job in, execute it and when it is done take control back to read next job
- Operating system stayed in memory permanently

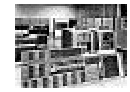
-18

Spooling

- Problem
  - Card readers are slow
  - Time to read the next job from punch cards means lost CPU time (expensive!)
- Solution: while executing one job load the next one into memory
- Might even bring multiple into memory and allow them to be executed out of order
  - Need scheduling algorithm to choose the next one to run

-19

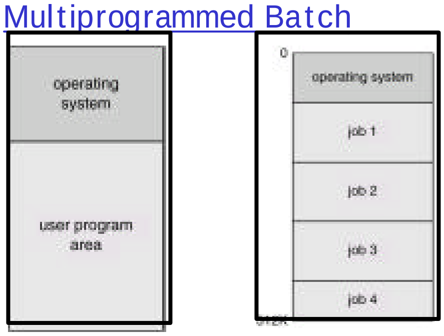
Multiprogrammed Batch Systems



- Keep multiple job in memory at the same time and interleave their execution (not pick one and run it to completion)
  - The applications still couldn't communicate directly (no pipes, sockets, shared memory, etc.)
  - So why allow more than one to run at a time?
- Able to overlap I/O of one application with the computation of another!
  - If one job requests I/O, don't leave the CPU idle while I/O completes- pick something else to run in the meantime
  - Each job take longer to actually run on the machine, but better machine utilization and throughput (important for expensive CPUs)

-20

Batch vs Multiprogrammed Batch



-21

Multiprogramming

- Requires much of the core OS functionality we will study
  - CPU scheduling algorithm to decide which one of the runnable jobs to run next
  - Memory management (simple at first)
  - Protection of I/O devices from multiple applications desiring to use them
  - Asynchronous I/O
    - CPU issues a command to a device then can go do something else until job is done
    - Device notifies CPU of completion with an interrupts or CPU periodically polls device for completion

-22

Time Sharing

- Batch systems (even multiprogrammed batch systems) required users to submit jobs with their inputs and then later get output back
- Time sharing systems provided interactive computing
  - Connect to computer through a dumb terminal (monitor, keyboard, serial connection to computer)
  - Each interactive user feels like they have their own computer, but in reality jobs are swapped on and off the CPU rapidly enough that users don't notice
  - Enables interactive applications like editors and command shells even debugging running programs
  - User interact with job throughout its run time

-23

Scheduling for Time Sharing

- Need to swap jobs on and off CPU quickly enough that users don't notice
- Each job given a "time slice"
- Batch scheduling was very different – let application run until it did some I/O then swap it out until its I/O completes
- Batch optimizes for throughput; Time sharing optimizes for response time

-24

Shared File Systems for Time Sharing

- How do users who log in over dumb terminal say which programs to run with what input?
  - No longer submit batch jobs with their input on punch cards
  - Log in over a serial line
- Command shells: execute user command then await the next one
- Thus time sharing systems needed shared file systems that held commonly used programs
- Users could log in, run utilities, store input and output file in shared file system

-25

Security for Time Sharing

- Batch systems had multiple applications running at the same time but there inputs and actions were fixed at submission time with no knowledge of what else would be run with it
- Time Sharing systems mean multiple interactive users on a machine poking around = Increased threat to privacy and security

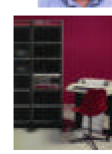
-26

CTSS and Multics

- Compatible Time Sharing System (CTSS) one of first time sharing system
  - Developed at MIT
  - first demonstrated in 1961 on the IBM 709, swapping to tape.
- Multics (Multiplexed Information and Computing Service)
  - Ambitious timesharing system developed in 1960's by MIT, Bell Labs and GE
  - Many OS concepts conceived of in Multics, but hard to implement in 1960
  - Last Multics installation in Hallifax Nova Scotia decommissioned 10/31/2000!

-27

UNIX



- Bell Labs pulled out of MULTICs effort in 1969 convinced it was economically infeasible to produce a working system
- Handful of researchers at Bell Labs including Ken Thompson and Dennis Ritchie developed a scaled down version on MULTICS called UNICs (UNiplexed Information and Computing Service) – am "emasculated MULTICS"
- AT&T licensed completed UNIX
  - Provided licensees (including UC Berkeley) with the software code and manuals because Department of Justice didn't allow AT&T to sell software

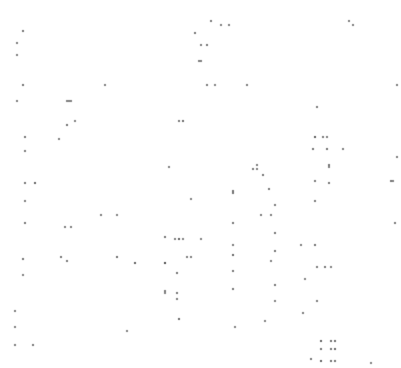
-28

UNIX (con't)

- In 1977, the first Berkeley Software Distribution (BSD) version of UNIX was released.
- AT&T transferred its own UNIX development efforts to Western Electric
- In 1982, Western Electric released System III UNIX (marketing thought that System III sounded more stable than System I ☺)
- In 1984, UC Berkeley released version 4.2BSD which included a complete implementation of the TCP/IP networking protocols

-29

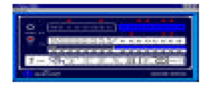
Wow!



-30

- We've been following the development of corporate/academic computing
- Next, we switch gears to personal computing

Personal Computers



- Computers become cheap enough that one can be dedicated to an individual
- First PC was the Altair
  - produced by MITS in 1975
  - 8 bit Intel 8080, 256 bytes(!) of memory
  - No keyboard (front panel switches instead), monitor, tape or disk!
  - $400
  - Popular with hobbyists (like building radios or TVs)
- 1975-1980, many companies make PCs (or microcomputers) based on the 8080 chip
  - Still for hobbyists
  - For an OS, most run CP/M (Control Program Microcomputer) from Digital Research

Apple Computer

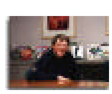
- 1976 - Members of a California hobbyist group, Steve Wozniak and Steve Jobs, sell a fully assembled microcomputer, Apple I
  - No more lights and switches
  - $666 for machine with video terminal, keyboard and 4K RAM, 4 K more for $120, cassette tape interface for $75

- 1977 - Apple II
  - Looks basically like the desktop PC we know and love
  - Mouse, speakers and color (to play Breakout ☺)

IBM PC

- 1980 - IBM decides to get into the PC business
- Rather than build its own hardware, it goes with the Intel 8088
- Rather than write its own software, it looked to get a language processor and an OS from elsewhere
  - Licenses Microsoft's BASIC interpreter
  - Still need an OS
    - Digital Research's new version of CP/M way behind schedule
    - UNIX needs too many resources (100K of memory and a hard disk)
    - They ask Microsoft if it could deliver an OS too

DOS

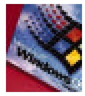


- In 1981, QDOS (Quick-and-Dirty OS) purchased by Microsoft and renamed MS-DOS
  - QDOS was a scaled down version of the CP/M OS for the 8088 family of computers
- Features of DOS 1.0 and 2.0
  - OS back to a library linked in with applications
    - 1 M address space; Applications got only 640K
  - Apps do anything they want! - No memory protection; no hardware protection
  - No hierarchical file system – single directory at most 64 files

Windows On Top, DOS underneath

- 1981 – Microsoft begins development of the Interface Manager that would eventually become Microsoft Windows
- 1985 – Windows 1.0
  - runs as a library on top of DOS
  - allowed users to switch between several programs—without requiring them to quit and restart individual applications
- 1987 – Windows 2.0 offers overlapping windows

Windows



- Two Windows product lines
  - 1994 – Windows NT
    - entirely new OS kernel (not DOS!) designed for high-end server machines
    - Microkernel based concepts pioneered in CMU research project MACH
  - 1995 – Windows 95
    - Included MS-DOS 7.0, but took over from DOS completely after starting
    - pre-emptive multitasking, advanced file systems, threading, networking
- 2000 - Windows 2000
  - Upgrade to the Windows NT code base
  - Designed to permanently replace Windows 95 and its DOS roots

-37

Linux

- Linus Torvald, a student in Finland, extends an educational operating system Minix into an Unix style operating system for PCs (x86 machines) as a hobby
- In 1991, he posts to the comp.os.minix newsgroup an invitation for others to join him in developing this free, open source OS
- Different distributions package the same Linux kernel together with other various collections of open source software (GNU-Linux)
- Companies sell support or installation CDs, but freely software available
- Linux is now the fastest growing segment of the operating system market

-38

PC-OSs meet Timesharing

- Both Linux and later versions of Windows have brought many advanced OS concepts to the desktop
  - Multiprogramming first added back in because people like to do more than one thing at a time (spool job to printer and continue typing)
  - Memory protection added back in to protect against buggy applications – not other users!
  - Linux (and even Windows now) allow users to log in remotely and multiple users to be running jobs
- Steady increases in hardware performance and capacity made this possible

-39

Parallel and Distributed Computing

- Harness resources of multiple computer systems
  - Parallel computing focused on splitting up a single task and getting speed-up proportional to the number of machines
  - Distributed computing focused on harnessing resources (hardware or data) from geographically dispersed machines
- Hardware
  - SIMD, MIMD, MPPs, SMPs, NOWs, COWs,..
  - Tightly or Loosely Coupled machines? Do they share memory? Do they share a high speed internal network? Maybe a bus? Do they share a clock? Do all processors operate the same instruction at the same time but on different data?

-40

Parallel and Distributed (con't)

- Need communication between machines
  - Networking hardware and software protocols?
- Fault tolerance: helps or hurts?
  - Ability to offer fail-over to duplicated resources?
  - "A distributed system is one where I can't do work because a machine I never heard of goes down"
- Load balancing, synchronization, authentication, naming

-41

Real Time OSes

- If application demands guaranteed response times, OS can be designed to provide service guarantees
- Hard-real time
  - Usually need guaranteed physical response to sensors
  - Ex. Industrial control, Safety monitoring, medical imaging
- Soft-real time
  - OS priorities and can provide desired response time most of the time
  - Ex. Robotics, virtual reality

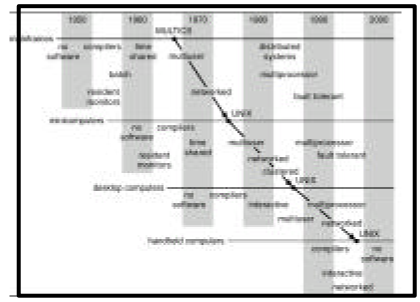
-42

Embedded OSes

- Cheap processors everywhere – in toys, appliances, cars, cell phones, PDAs
- Typically designed for one dedicated application
- Very constrained hardware resource
  - Slow processor, no disk, little memory, small displays, no keyboard
  - Better off than early mainframes though ?
- Will march of technology bring power of today's desktops and full OS features to all these devices too?

-43

Lessons from history?



-44

This Semester

- Architectural support for OS; Application demand on OS
- Major components of an OS
  - Scheduling, Memory Management, Synchronization, File Systems, Networking,..
- How is the OS structured internally and what interfaces does it provide for using its services and tuning its behavior?
- What are the major abstractions modern OSes provide to applications and how are they supported?

-45