Principles of Operating Systems

Lecture 1-2 - Introduction and overview, operating system structure

Ardalan Amiri Sani (ardalan@uci.edu)

[*lecture slides contains some content adapted from : Silberschatz textbook authors, Anerson textbook authors, John Kubiatowicz (Berkeley), John Ousterhout(Stanford), previous slides by Prof. Nalini Venkatasubramanian,* http://www-inst.eecs.berkeley.edu/~cs162/ *and others*]

Staff

- The course will be co-taught by
  - Nalini Venkatasubramanian (nalini@ics.uci.edu)
  - Ardalan Amiri Sani (ardalan@uci.edu)
- Prof. Venkat will cover the first half of the course
- Prof. Amiri Sani will cover the second half

Staff

Teaching Assistants:

Kyle Benson (kebenson@uci.edu)

Joseph Nash (jmnash@uci.edu)

Qiuxi Zhu (qiuxiz@ics.uci.edu)

Readers:

Hao Zhang (hzhang10@uci.edu)

Purvi Kaul (kaulp@uci.edu)

Course logistics and details

- Course web page -
  - http://www.ics.uci.edu/~ics143
- Discussions – Fridays 9:00-9:50 a.m, 10-10:50 a.m, 2-2:50 p.m, ICS 174
  - Pop quizzes

Course logistics and details

- Textbook:

  Operating System Concepts -- Ninth Edition
  A. Silberschatz, P.B. Galvin, and G. Gagne
  (Eighth, Seventh,Sixth and Fifth editions,
  and Java Versions are fine as well).

- Alternate Books
  - Operating Systems: Principles and Practice, by T. Anderson and M. Dahlin (second edition)
  - Modern Operating Systems, by Tanenbaum (Third edition)
  - Principles of Operating Systems, by L.F. Bic and A.C. Shaw, 2003.

Course logistics and details

- Homeworks and Assignments
  - 4 written homeworks in the quarter
  - 1 programming assignment (knowledge of C++ or Java required).
    - Handed out at midterm; submit/demo during Finals Week
    - Multistep assignment – don't start in last week of classes!!!
  - Late homeworks will not be accepted.
  - All submissions will be made using the EEE Dropbox for the course
- Tests
  - Midterm – tentatively Thursday, Week 6 in class
  - Final Exam – per UCI course catalog

Grading Policy

- Homeworks - 30%
  - 4 written homeworks each worth 5% of the final grade.
  - 1 programming assignment worth 10% of the final grade
- Midterm - 30% of the final grade
- Final exam - 40% of the final grade

- Final assignment of grades will be based on a curve.

Lecture Schedule

- Week 1: Prof. Amiri Sani
  - Introduction to Operating Systems, Computer System Structures, Operating System Structures
- Week 2 : Prof. Venkat
  - Processes and Threads
- Week 3: Prof. Venkat/Prof. Amiri Sani
  - Processes and Threads, and CPU Scheduling
- Week 4: Prof. Venkat
  - Scheduling
- Week 5: Prof. Venkat
  - Process Synchronization

Course Schedule

- Week 6 – Prof. Venkat
  - Deadlocks, Midterm review and exam
- Week 7 – Prof. Amiri Sani
  - Memory Management
- Week 8 – Prof. Amiri Sani
  - Memory Management, Virtual Memory
- Week 9 – Prof. Amiri Sani
  - File Systems Interface and Implementation
- Week 10 – Prof. Amiri Sani
  - I/O Subsystems

Office hours

- Instructors
  - Mondays 3-4:30pm
  - Weeks 2-6 at Prof. Venkat's office (DBH 2086)
  - Weeks 7-10 at Prof. Amiri Sani's office (DBH 3062)
- TA's
  - TBD

Piazza

- https://piazza.com/uci/spring2016/compsci143a
- Post questions here.

Overview

- What is an operating system?
- Operating systems history
- Computer system and operating system structure

What is an Operating System?

What is an Operating System?

- OS is the software that acts an intermediary between the user applications and computer hardware.

Computer System Components

- Hardware
  - Provides basic computing resources (CPU, memory, I/O devices).
- Operating System
  - Controls and coordinates the use of hardware among application programs.
- Application Programs
  - Solve computing problems of users (compilers, database systems, video games, business programs such as banking software).
- Users
  - People, machines, other computers

Abstract View of System

Operating system roles

- Referee:
  - Resource allocation among users, applications
  - Isolation of different users, applications from each other
  - Communication between users, applications

Operating system roles

- Illusionist
  - Each application appears to have the entire machine to itself
  - Infinite number of processors, (near) infinite amount of memory, reliable storage, reliable network transport

Operating system roles

- Glue
  - Libraries, user interface widgets, …
  - Reduces cost of developing software

Example: file systems

- Referee
  - Prevent users from accessing each other's files without permission
- Illusionist
  - Files can grow (nearly) arbitrarily large
  - Files persist even when the machine crashes in the middle of a save
- Glue
  - Named directories, printf, …

OS challenges

OS challenges

- Reliability
  - Does the system do what it was designed to do?

OS challenges

- Availability
  - What portion of the time is the system working?
  - Mean Time To Failure (MTTF), Mean Time to Repair

OS challenges

- Security
  - Can the system be compromised by an attacker?

OS challenges

- Privacy
  - Data is accessible only to authorized users

OS challenges

- Performance
  - Latency/response time
    - How long does an operation take to complete?
  - Throughput
    - How many operations can be done per unit of time?
  - Overhead
    - How much extra work is done by the OS?
  - Fairness
    - How equal is the performance received by different users?
  - Predictability
    - How consistent is the performance over time?

OS challenges

- Portability
  - For programs:
    - Application programming interface (API)
  - For the kernel
    - Hardware abstraction layer

OS needs to keep pace with hardware improvements

	1981	1997	2014	Factor (2014/1981)
Uniprocessor speed (MIPS)	1	200	2500	2.5K
CPUs per computer	1	1	10+	10+
Processor MIPS/$	$100K	$25	$0.20	500K
DRAM Capacity (MiB)/$	0.002	2	1K	500K
Disk Capacity (GiB)/$	0.003	7	25K	10M
Home Internet	300 bps	256 Kbps	20 Mbps	100K
Machine room network	10 Mbps (shared)	100 Mbps (switched)	10 Gbps (switched)	1000
Ratio of users to computers	100:1	1:1	1:several	100+

Why should I study Operating Systems?

Why should I study Operating Systems?

- Need to understand interaction between the hardware and software
- Need to understand basic principles in the design of computer systems
  - efficient resource management, security, flexibility

Why should I study Operating Systems?

- Because it enables you to do things that are difficult/impossible otherwise.

Example: Rio: I/O sharing implemented in the operating system kernel

Observation: I/O devices important for personal computers

A personal computer today

- Super AMOLED display
- Capacitive touchscreen (multitouch)
- Audio (speaker, microphone)
- Vibration
- S pen
- 13 MP front camera
- 2 MP back camera
- Accelerometer
- Gyroscope
- Proximity sensor
- Compass
- Barometer
- Temperature sensor
- Humidity sensor
- Gesture sensor
- GPS
- 4G LTE
- NFC
- WiFi
- Bluetooth
- Infrared
- 64 GB internal storage (extended by microSD)
- Adreno 330 GPU
- Hexagon DSP
- Multimedia processor

A personal computer today

- Super AMOLED display
- Capacitive touchscreen (multitouch)
- Audio (speaker, microphone)
- Vibration
- S pen
- 13 MP front camera
- 2 MP back camera
- Accelerometer
- Gyroscope
- Proximity sensor
- Compass
- Barometer
- Temperature sensor
- Humidity sensor
- Gesture sensor
- GPS
- 4G LTE
- NFC
- WiFi
- Bluetooth
- Infrared
- 64 GB internal storage (extended by microSD)
- Adreno 330 GPU
- Hexagon DSP
- Multimedia processor

interaction

A personal computer today

- Super AMOLED display
- Capacitive touchscreen (multitouch)
- Audio (speaker, microphone)
- Vibration
- S pen
- 13 MP front camera
- 2 MP back camera
- Accelerometer
- Gyroscope
- Proximity sensor
- Compass
- Barometer
- Temperature sensor
- Humidity sensor
- Gesture sensor
- GPS
- 4G LTE
- NFC
- WiFi
- Bluetooth
- Infrared
- 64 GB internal storage (extended by microSD)
- Adreno 330 GPU
- Hexagon DSP
- Multimedia processor

sensing

A personal computer today

- Super AMOLED display
- Capacitive touchscreen (multitouch)
- Audio (speaker, microphone)
- Vibration
- S pen
- 13 MP front camera
- 2 MP back camera
- Accelerometer
- Gyroscope
- Proximity sensor
- Compass
- Barometer
- Temperature sensor
- Humidity sensor
- Gesture sensor
- GPS
- 4G LTE
- NFC
- WiFi
- Bluetooth
- Infrared
- 64 GB internal storage (extended by microSD)
- Adreno 330 GPU
- Hexagon DSP
- Multimedia processor

connectivity, storage

A personal computer today

- Super AMOLED display
- Capacitive touchscreen (multitouch)
- Audio (speaker, microphone)
- Vibration
- S pen
- 13 MP front camera
- 2 MP back camera
- Accelerometer
- Gyroscope
- Proximity sensor
- Compass
- Barometer
- Temperature sensor
- Humidity sensor
- Gesture sensor
- GPS
- 4G LTE
- NFC
- WiFi
- Bluetooth
- Infrared
- 64 GB internal storage (extended by microSD)
- Adreno 330 GPU
- Hexagon DSP
- Multimedia processor

acceleration

Multiple computers for unique I/O

Multiple computers for unique I/O

Multiple computers for unique I/O

I/O sharing

How to build this?

How to build this?

Application layer

- IP Webcam
- Wi-Fi Speaker
- MightyText

Client

Server

Do not meet our criteria

- High engineering effort
- No support for legacy applications
- No support for all I/O device features

Rio: I/O servers for sharing I/O between mobile systems

Ardalan Amiri Sani, Kevin Boos, Min Hong Yun, and Lin Zhong, "Rio: A System Solution for Sharing I/O between Mobile Systems," in *Proc. **ACM MobiSys***, June 2014. **(Best Paper Award)**

Key idea: device files as the boundary

I/O devices abstracted as
(device) files in Unix-like OSes
e.g., /dev/foo

Key idea: device files as the boundary

Key idea: device files as the boundary

Client

Server

Key idea: device files as the boundary

Client

Server

Key idea: device files as the boundary

Client

Server

Video demo of Rio

http://www.ruf.rice.edu/~mobile/rio.html

Operating systems are everywhere

Operating systems are everywhere

Operating systems are everywhere

Operating systems are everywhere

Overview

- What is an operating system?
- Operating systems history
- Computer system and operating system structure

Operating systems history

- Early Operating Systems
- Simple Batch Systems
- Multiprogrammed Batch Systems
- Time-sharing Systems
- Personal and mobile Computer Systems

People-to-Computer Ratio Over Time

Early Systems - Bare Machine (1950s)

Hardware – *expensive* ; Human – *cheap*

From John Ousterhout slides

- Structure
  - Large machines run from console
  - Single user system
    - Programmer/User as operator
  - Paper tape or punched cards
- Early software
  - Assemblers, compilers, linkers, loaders, device drivers, libraries of common subroutines.
- Secure execution
- Inefficient use of expensive resources
  - Low CPU utilization, high setup time.

Batch Systems (1960's)

- Reduce setup time by batching jobs with similar requirements.
- Hire an operator
  - User is NOT the operator
- Automatic job sequencing
  - Forms a rudimentary OS.
  - Resident Monitor
    - Holds initial control, control transfers to job and then back to monitor.
  - Problem
    - Need to distinguish job from job and data from program.
    - Special cards indicate what to do.
    - User program prevented from performing I/O

From John Ousterhout slides

Batch Systems (1960's)

- Solutions to speed up I/O:
- Offline Processing
  - load jobs into memory from tapes, card reading and line printing are done offline.
  - User submits card deck
  - cards put on tape
  - tape processed by operator
  - output written to tape
  - tape printed on printer
  - Separate user from computer
- Problems
  - Long turnaround time - up to 2 DAYS!!!
  - Low CPU utilization
    - I/O and CPU could not overlap; slow mechanical devices.

Batch Systems (1960's)

- Solutions to speed up I/O:
- Spooling (Simultaneous Peripheral Operation On-Line)
  - Use disk (random access device) as large storage for reading as many input files as possible and storing output files until output devices are ready to accept them.
  - Allows overlap - I/O of one job with computation of another.
  - Introduces notion of a job pool that allows OS choose next job to run so as to increase CPU utilization.

Speeding up I/O: Direct Memory Access (DMA)

- Data moved directly between I/O devices and memory
- CPU can work on other tasks

Batch Systems - I/O completion

- How do we know that I/O is complete?
  - Polling:
    - Device sets a flag when it is busy.
    - Program tests the flag in a loop waiting for completion of I/O.
  - Interrupts:
    - On completion of I/O, device forces CPU to jump to a specific instruction address that contains the interrupt service routine.
    - After the interrupt has been processed, CPU returns to code it was executing prior to servicing the interrupt.

Multiprogramming

- Use interrupts to run multiple programs simultaneously
  - When a program performs I/O, instead of polling, execute another program till interrupt is received.
- Requires secure memory, I/O for each program.
- Requires intervention if program indefinite loops.
- Requires CPU scheduling to choose the next job to run.

Timesharing

Hardware – *getting cheaper*; Human – *getting expensive*

- Programs queued for execution in FIFO order.
- Like multiprogramming, but timer device interrupts after a quantum (timeslice).
  - Interrupted program is returned to end of FIFO
  - Next program is taken from head of FIFO
- Control card interpreter replaced by command language interpreter.

Timesharing (cont.)

- Interactive (action/response)
  - when OS finishes execution of one command, it seeks the next control statement from user.
- File systems
  - online filesystem is required for users to access data and code.
- Virtual memory
  - Job is swapped in and out of memory to disk.

Personal Computing Systems - desktops

Hardware – *cheap* ; Human – *expensive*

- Single user systems, portable.
- I/O devices - keyboards, mice, display screens, small printers.
- Single user systems may not need advanced CPU utilization or protection features.
- Advantages:
  - user convenience, responsiveness, ubiquitous

Personal Computing Systems - Mobile and wearable Systems

Hardware – ***very cheap*** ; **Human** – ***very expensive***

- Single user, multiple computers
- Laptops
- Smartphones
- Tablets
- Smart glasses
- Smart watches

Overview

- What is an operating system?
- Operating systems history
- Computer system and operating system structure

Computer System & OS Structures

- Computer System Organization
- Process abstraction and hardware protection
- System call and OS services
- Storage architecture
- OS organization
- OS tasks
- Virtual Machines

Computer System Organization

CPU execution

Execution sequence:

- Fetch Instruction at PC
- Decode
- Execute (possibly using registers)
- Write results to registers/mem
- PC = Next Instruction(PC)
- Repeat

From Berkeley OS course

Computer System Organization

I/O devices

- I/O devices and the CPU execute concurrently.
- Each device controller is in charge of a particular device type
  - Each device controller has a local buffer. I/O is from the device to local buffer of controller
- CPU moves data from/to main memory to/from the local buffers
- Device controller interrupts CPU on completion of I/O

Interrupts

- Interrupt transfers control to the interrupt service routine
  - Interrupt Service Routine: Segments of code that determine action to be taken for interrupt.

- Determining the type of interrupt
  - Polling: same interrupt handler called for all interrupts, which then polls all devices to figure out the reason for the interrupt
  - Interrupt Vector Table: different interrupt handlers will be executed for different interrupts

Interrupt Number	Address
0	0003h
1	000Bh
2	0013h
3	001Bh
4	0023h
5	002Bh
6	0033h
7	003Bh
8	0043h
9	004Bh
10	0053h
11	005Bh
12	0063h
13	006Bh
14	0073h
15	007Bh

Interrupt Number	Address
16	0083h
17	008Bh
18	0093h
19	009Bh
20	00A3h
21	00ABh
22	00B3h
23	00BBh
24	00C3h
25	00CBh
26	00D3h
27	00DBh
28	00E3h
29	00EBh
30	00F3h
31	00FBh

Interrupt handling

- OS preserves the state of the CPU
  - stores registers and the program counter (address of interrupted instruction).
- Incoming interrupts are disabled while another interrupt is being processed to prevent a *lost interrupt*.

Different types of I/O processing for programs

- Synchronous I/O
  - After I/O is requested, *wait* until I/O is done. Program will be idle.

- Asynchronous I/O
  - After I/O is requested, control returns to user program without waiting for I/O completion.

Direct Memory Access (DMA)

- Used for high speed I/O devices able to transmit information at close to memory speeds.
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.

Process Abstraction

Process Abstraction

- Process: an *instance* of a program, running with limited rights

Process Abstraction

- Process: an *instance* of a program, running with limited rights
  - Thread: a sequence of instructions within a process
    - Potentially many threads per process (for now 1:1)
  - Address space: set of rights of a process
    - Memory that the process can access
    - Other permissions the process has (e.g., which system calls it can make, what files it can access)

How to limit process rights?

Hardware Protection

- Dual Mode Operation
- Memory Protection
- CPU Protection
- I/O Protection

Should a process be able to execute any instructions?

Should a process be able to execute any instructions?

- No
  - Can alter system configuration
  - Can access unauthorized memory
  - Can access unauthorized I/O
  - etc.
- How to prevent?

Dual-mode operation

- Provide hardware support to differentiate between at least two modes of operation:
  1. User mode -- execution done on behalf of a user.
  2. Kernel mode (monitor/supervisor/system mode) -- execution done on behalf of operating system.
- "Privileged" instructions are only executable in the kernel mode
- Executing privileged instructions in the user mode "traps" into the kernel mode
  - Trap is a software generated interrupt caused either by an error or a user request

Dual-mode operation(cont.)

- Mode bit added to computer hardware to indicate the current mode: kernel(0) or user(1).
- When an interrupt or trap occurs, hardware switches to kernel mode.

How to isolate memory access?

Process address space

- Address space ⇒ the set of accessible addresses + state associated with them:
  - For a 32-bit processor there are 2^{32} = 4 billion addresses

Virtual Address

Providing the Illusion of Separate Address Spaces

Address translation and memory protection

Memory Protection

- Must provide memory protection at least for the interrupt vector and the interrupt service routines.
- When a process is running, only memory in that process address space must be accessible.
- When executing in kernel mode, the kernel has unrestricted access to all memory.

Memory Protection: base and bounds

- To provide memory protection, add two registers that determine the range of legal addresses a program may address.
  - Base Register - holds smallest legal physical memory address.
  - Limit register - contains the size of the range.
- Memory outside the defined range is protected.

Virtually Addressed Base and Bounds

The load instructions for the ***base*** and ***limit*** registers are ***privileged*** instructions.

CPU Protection

- How to prevent a process from executing indefinitely?

CPU Protection

- Timer - interrupts computer after specified period to ensure that OS maintains control.
  - Timer is decremented every clock tick.
  - When timer reaches a value of 0, an interrupt occurs.
- Timer is commonly used to implement time sharing.
- Timer is also used to compute the current time.
- Load timer is a privileged instruction.

I/O Protection

- All I/O instructions are privileged instructions.
- Must ensure that a user program could never gain control of the computer in kernel mode, for e.g. a user program that as part of its execution, stores a new address in the interrupt vector.

Question

- Given the I/O instructions are privileged, how do users perform I/O?

Question

- Given the I/O instructions are privileged, how do users perform I/O?

- Via system calls - the method used by a process to request action by the operating system.

System Calls

- User code can issue a syscall, which causes a trap
- Kernel handles the syscall

System Calls

- Interface between running program and the OS.
  - Assembly language instructions (macros and subroutines)
  - Some higher level languages allow system calls to be made directly (e.g. C)
- Passing parameters between a running program and OS via registers, memory tables or stack.
- Linux has about 300 system calls
  - read(), write(), open(), close(), fork(), exec(), ioctl(),…..

System services or system programs

- Convenient environment for program development and execution. User view of OS is defined by system services, not system calls.
  - Command Interpreter (sh, csh, ksh) - parses/executes other system programs
  - File manipulation - copy (cp), print (lpr), compare(cmp, diff)
  - File modification - editing (ed, vi, emacs)
  - Application programs - send mail (mail), read news (rn)
  - Programming language support (cc)
  - Status information, communication
  - etc….

Command Interpreter System

- Commands that are given to the operating system via command statements that execute
  - Process creation and deletion, I/O handling, Secondary Storage Management, Main Memory Management, File System Access, Protection, Networking.
- Obtains the next command and executes it.
- Programs that read and interpret control statements also called -
  - Control card interpreter, command-line interpreter, shell (in UNIX)

System Design and Implementation

- Establish design goals
  - User Goals
  - System Goals
- Software Engineering -
  - Separate mechanism from policy. Policies determine what needs to be done, mechanisms determine how they are done.
- Choose a high-level implementation language
  - faster implementation, more compact, easier to debug

System Generation

- OS written for a class of machines, must be configured for each specific site.
- SYSGEN is the process to customize OS for the exact hardware
  - Source code can be customized
  - Pre-compiled modules selected and linked based on hardware
  - All code is compiled, but right code is chosen at execution time, e.g., by using a table

System Boot

- Bootstrap program - loader program loads kernel, kernel loads rest of OS.
  - Bootstrap program stored in ROM

Storage Structure

- Main memory - only large storage media that the CPU can access directly.
- Secondary storage - extension of main memory that has large nonvolatile storage capacity.
  - Magnetic disks - rigid metal or glass platters covered with magnetic recording material.
    - Disk surface is logically divided into tracks, subdivided into sectors.
    - Disk controller determines logical interaction between device and computer.

Storage Hierarchy

- Storage systems are organized in a hierarchy based on
  - Speed
  - Cost
  - Volatility

- Caching - process of copying information into faster storage system; main memory can be viewed as fast cache for secondary storage.

Storage Device Hierarchy

Operating Systems: How are they organized?

- Simple
  - Only one or two levels of code
- Layered
  - Lower levels independent of upper levels
- Modular
  - Core kernel with Dynamically loadable modules
- Microkernel
  - OS built from many user-level processes

OS Structure - Simple Approach

- MS-DOS - provides a lot of functionality in little space.
  - Not divided into modules, Interfaces and levels of functionality are not well separated

Original UNIX System Structure

- Limited structuring, has 2 separable parts
  - Systems programs
  - Kernel
    - everything below system call interface and above physical hardware.
    - Filesystem, CPU scheduling, memory management

Layered OS Structure

- OS divided into number of layers - bottom layer is hardware, highest layer is the user interface.
- Each layer uses functions and services of only lower-level layers.
- THE Operating System and Linux Kernel has successive layers of abstraction.

User Programs
Interface Primitives
Device Drivers and Schedulers
Virtual Memory
I/O
CPU Scheduling
Hardware

Layered Operating System

Modules-based Structure

- Most modern operating systems implement modules
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexibility
  - Linux makes extensive use of modules

Monolithic vs. Microkernel OS

- Monolithic OSes have large kernels with a lot of components
  - Linux, Windows, Mac
- Microkernels moves as much from the kernel into "*user*" space
  - Small core OS running at kernel level
  - OS Services built from many independent user-level processes

- Communication between modules with message passing
- Benefits:
  - Easier to extend a microkernel
  - Easier to port OS to new architectures
  - More reliable (less code is running in kernel mode)
  - Fault Isolation (parts of kernel protected from other parts)
  - More secure

- Detriments:
  - Performance overhead severe for naïve implementation

A microkernel OS

Slide adapted from http://web.cecs.pdx.edu/~walpole/class/cs533/fall2015/home.html

OS Task: Process Management

- Process - fundamental concept in OS
  - Process is an instance of a program in execution.
  - Process needs resources - CPU time, memory, files/data and I/O devices.
- OS is responsible for the following process management activities.
  - Process creation and deletion
  - Process suspension and resumption
  - Process synchronization and interprocess communication
  - Process interactions - deadlock detection, avoidance and correction

OS Task: Memory Management

- Main Memory is an array of addressable words or bytes that is quickly accessible.
- Main Memory is volatile.
- OS is responsible for:
  - Allocate and deallocate memory to processes.
  - Managing multiple processes within memory - keep track of which parts of memory are used by which processes. Manage the sharing of memory between processes.
  - Determining which processes to load when memory becomes available.

OS Task: Secondary Storage and I/O Management

- Since primary storage is expensive and volatile, secondary storage is required for backup.
- Disk is the primary form of secondary storage.
  - OS performs storage allocation, free-space management and disk scheduling.
- I/O system in the OS consists of
  - Buffer caching and management
  - Device driver interface that abstracts device details
  - Drivers for specific hardware devices

OS Task: File System Management

- File is a collection of related information defined by creator - represents programs and data.

- OS is responsible for
  - File creation and deletion
  - Directory creation and deletion
  - Supporting primitives for file/directory manipulation.
  - Mapping files to disks (secondary storage).
  - Backup files on archival media (tapes).

OS Task: Protection and Security

- Protection mechanisms control access of programs and processes to user and system resources.
  - Protect user from himself, user from other users, system from users.
- Protection mechanisms must:
  - Distinguish between authorized and unauthorized use.
  - Specify access controls to be imposed on use.
  - Provide mechanisms for enforcement of access control.
  - Security mechanisms provide trust in system and privacy
    - authentication, certification, encryption etc.

OS Task: Networking

- Connecting processors in a distributed system
- Distributed System is a collection of processors that do not share memory or a clock.
- Processors are connected via a communication network.
- Advantages:
  - Allows users and system to exchange information
  - provide computational speedup
  - increased reliability and availability of information

Virtual Machines

Virtual Machines

Virtual Machines

- Use cases
  - Resource configuration
  - Running multiple OSes, either the same or different OSes
  - Run existing OS binaries on different architecture

Summary of this week's lecture

- What is an operating system?
- Operating systems history
- Computer system and operating system structure