

COS 318: Operating Systems

Overview

Jaswinder Pal Singh and a Fabulous Course Staff
Computer Science Department
Princeton University

(<http://www.cs.princeton.edu/courses/cos318/>)



Important Times

- ◆ Precepts:
 - Mon: 7:30-8:20pm, 104 CS building
 - Tues: 7:30-8:20pm, 105 CS building
 - This week (TODAY and TOMORROW):
 - Tutorial on Assembly programming and kernel debugging
- ◆ Project 1
 - Design review:
 - Next Mon/Tues: 3:00–7:00 pm (**Signup online**), 010 Friend
 - Project 1 due: Sunday 9/29 at 11:55pm
- ◆ Immediate To-Do:
 - Make sure you have your project partner



2

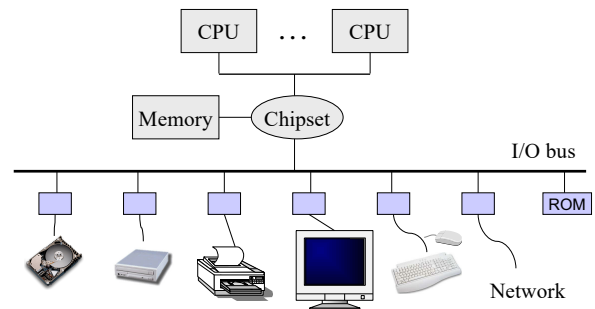
Today

- ◆ Overview of OS functionality
- ◆ Overview of OS components
- ◆ Interacting with the OS
- ◆ Booting a Computer



3

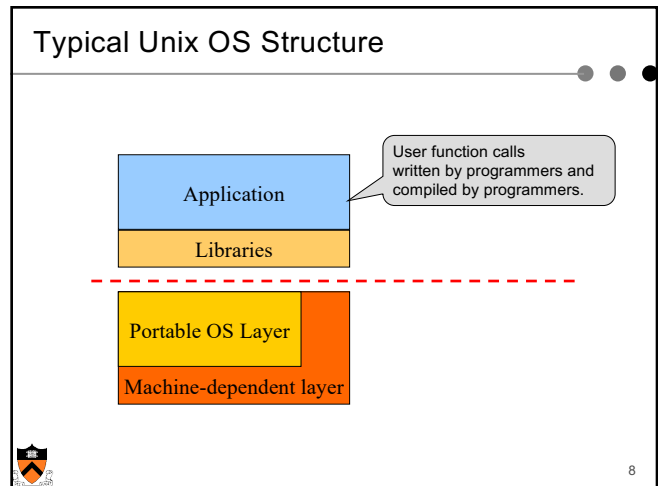
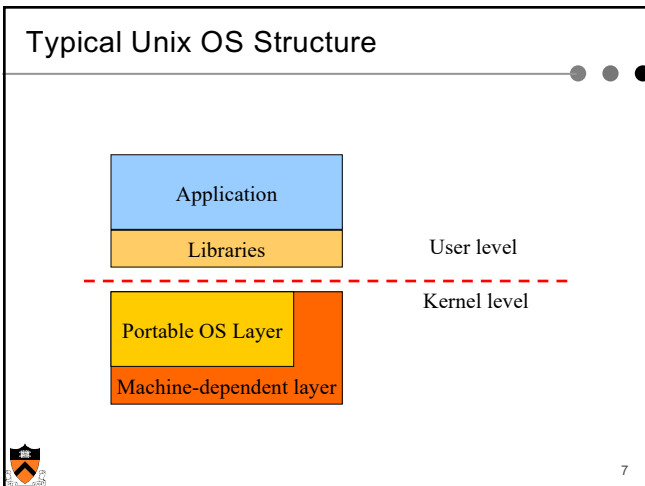
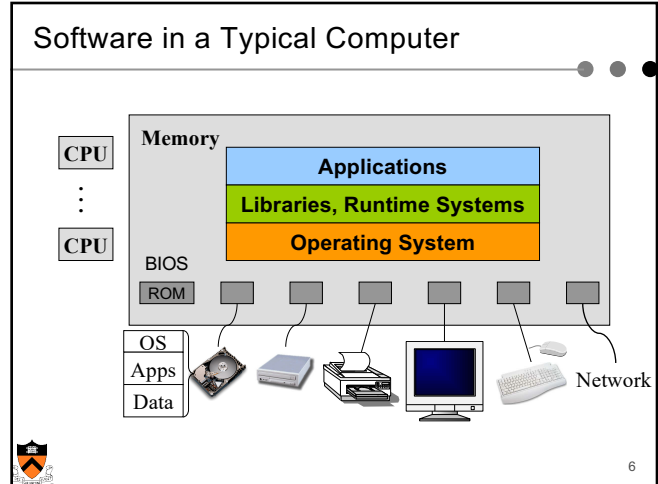
Hardware of A Typical Computer

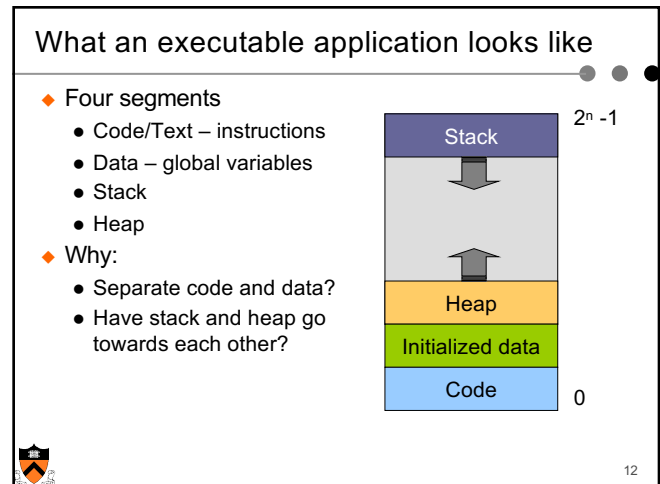
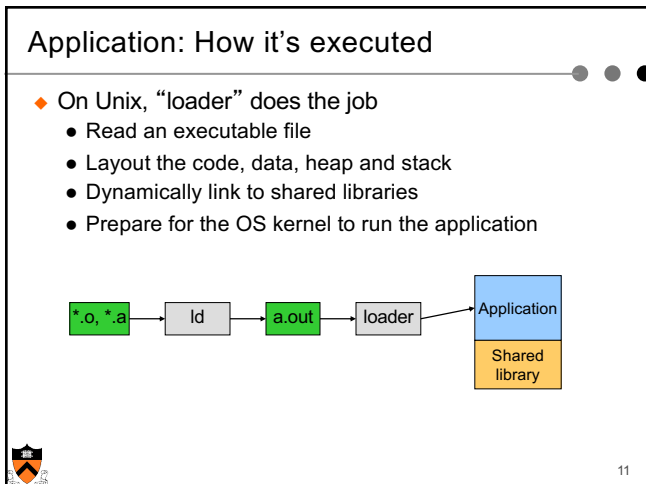
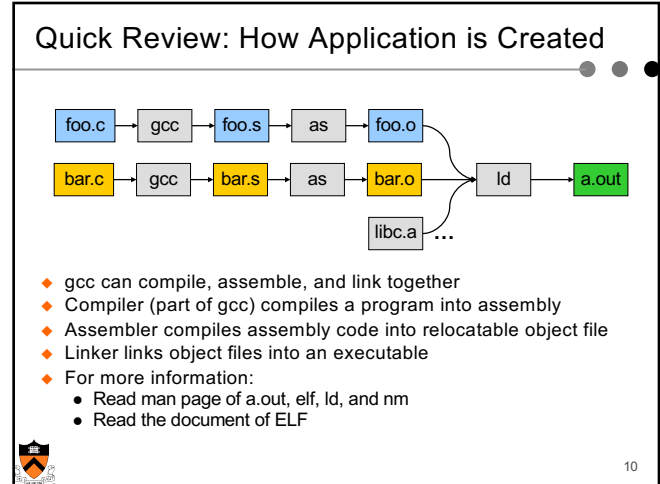
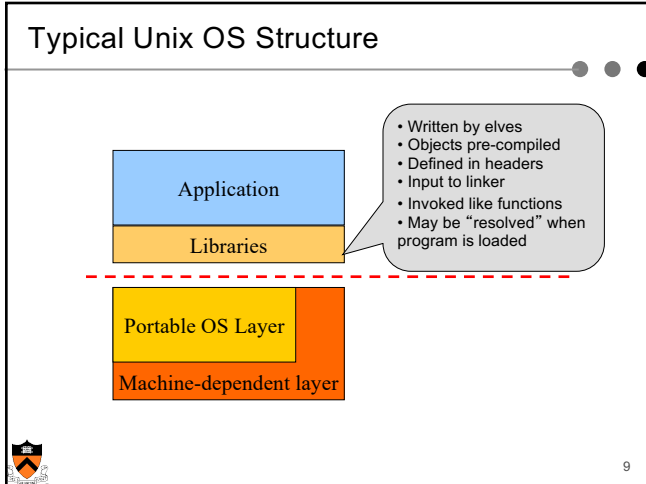


4

An Overview of HW Functionality

- ◆ **Executing the machine code** (CPU, cache, memory)
 - instructions for ALU, branch, memory operations
 - instructions for communicating with I/O devices
- ◆ **Performing I/O operations**
 - I/O devices and the CPU can execute concurrently
 - Every device controller is in charge of one device type
 - Every device controller has a local buffer
 - CPU moves data btwn main memory and local buffers
 - I/O is btwn device and local buffer of device controller
 - Device controller uses **interrupt** to inform CPU it is done
- ◆ **Protection**
 - Timer, paging (e.g. TLB), mode bit (e.g., kernel/user)





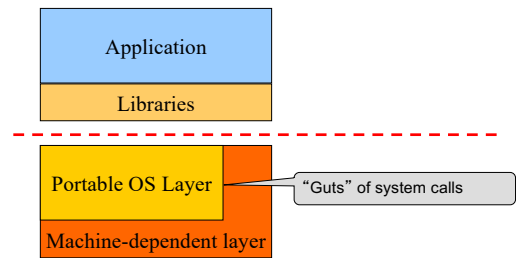
Responsibilities for the segments

- ◆ Stack
 - Layout by ?
 - Allocated/deallocated by ?
 - Local names are absolute/relative?
- ◆ Heap
 - Who sets the starting address?
 - Allocated/deallocated by ?
 - How do application programs manage it?
- ◆ Global data/code
 - Who allocates?
 - Who defines names and references?
 - Who translates references?
 - Who relocates addresses?
 - Who lays them out in memory?



13

Typical Unix OS Structure



14

Must Support Multiple Applications

- ◆ In multiple windows
 - Browser, shell, powerpoint, word, ...
- ◆ Use command line to run multiple applications


```
% ls -al | grep '^d'
```

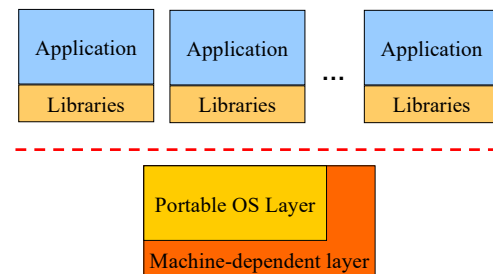
```
% foo &
```

```
% bar &
```



15

Multiple Application Processes



16

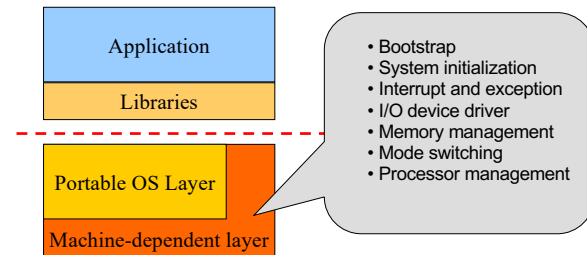
OS Service Examples

- ◆ Examples that are not provided at user level
 - System calls: file open, close, read and write
 - Control the CPU so that users won't cause problems
 - while (1);
 - Protection:
 - Keep user programs from crashing OS
 - Keep user programs from crashing each other
- ◆ Examples that are provided at user level
 - Read time of day
 - Protected user-level activities



17

Typical Unix OS Structure



18

Today

- ◆ Overview of OS functionality
- ◆ Overview of OS components
- ◆ Interacting with the OS
- ◆ Booting a Computer



20

OS components

- ◆ Resource manager for each HW resource
 - CPU: processor management
 - RAM: memory management
 - Disk: file system and secondary-storage management
 - I/O device management (keyboards, mouse, ...)
- ◆ Additional services:
 - networking
 - window manager (GUI)
 - command-line interpreters (e.g., shell)
 - resource allocation and accounting
 - protection
 - Keep user programs from crashing OS
 - Keep user programs from crashing each other



Processor Management

- Goals
 - Overlap between I/O and computation
 - Time sharing
 - Multiple CPU allocation
- Issues
 - Do not waste CPU resources
 - Synchronization and mutual exclusion
 - Fairness and deadlock

22

Memory Management

- Goals
 - Support for programs to run and to be written more easily
 - Allocation and management
 - Transfers from and to secondary storage
- Issues
 - Efficiency & convenience
 - Fairness
 - Protection

23

File System

- Goals:
 - Manage disk blocks
 - Map between files and disk blocks
- Typical file system calls
 - Open a file with authentication
 - Read/write data in files
 - Close a file
- Issues
 - Reliability
 - Safety
 - Efficiency
 - Manageability

24

I/O Device Management

- Goals
 - Interactions between devices and applications
 - Ability to plug in new devices
- Issues
 - Diversity of devices, third-party hardware
 - Efficiency
 - Fairness
 - Protection and sharing

25

Window Systems

- ◆ Goals
 - Interacting with a user
 - Interfaces to examine and manage apps and the system
- ◆ Issues
 - Inputs from keyboard, mouse, touch screen, ...
 - Display output from applications and systems
 - Where is the Window System?
 - All in the kernel (Windows)
 - All at user level
 - Split between user and kernel (Unix)



26

Summary

- ◆ Overview of OS functionality
 - Layers of abstraction
 - Services to applications
 - Resource management
- ◆ Overview of OS components
 - Processor management
 - Memory management
 - I/O device management
 - File system
 - Window system
 - ...



27

Today

- ◆ Overview of OS functionality
- ◆ Overview of OS components
- ◆ Interacting with the OS
- ◆ Booting a Computer



28

How the OS is Invoked

- ◆ Exceptions
 - Normal or program error: faults, traps, aborts
 - Special software generated: INT 3
 - Machine-check exceptions
- ◆ Interrupts
 - Hardware (by external devices)
 - Software: INT n
- ◆ System calls
- ◆ See Intel document volume 3 for details



29

Interrupts

- ◆ Raised by external events
- ◆ Interrupt handler is in kernel
- ◆ Eventually resume the interrupted process
- ◆ A way to
 - Switch CPU to another process
 - Overlap I/O with CPU
 - Handle other long-latency events

The diagram illustrates a memory stack with addresses 0, 1, ..., i, i+1, ..., N. An arrow points from the i+1 address to a box labeled 'Interrupt handler', indicating that the interrupt handler is invoked when the instruction pointer reaches i+1.

30

Interrupt and Exceptions (1)

Vector #	Mnemonic	Description	Type
0	#DE	Divide error (by zero)	Fault
1	#DB	Debug	Fault/trap
2		NMI interrupt	Interrupt
3	#BP	Breakpoint	Trap
4	#OF	Overflow	Trap
5	#BR	BOUND range exceeded	Trap
6	#UD	Invalid opcode	Fault
7	#NM	Device not available	Fault
8	#DF	Double fault	Abort
9		Coprocessor segment overrun	Fault
10	#TS	Invalid TSS (Task State Segment), Kernel/HW bug.	

31

Interrupt and Exceptions (2)

Vector #	Mnemonic	Description	Type
11	#NP	Segment not present	Fault
12	#SS	Stack-segment fault	Fault
13	#GP	General protection	Fault
14	#PF	Page fault	Fault
15		Reserved	Fault
16	#MF	Floating-point error (math fault)	Fault
17	#AC	Alignment check	Fault
18	#MC	Machine check	Abort
19-31		Reserved	
32-255		User defined	Interrupt

32


System Calls

- ◆ Operating system API
 - Interface between an application and the operating system kernel
- ◆ Categories of system calls
 - Process management
 - Memory management
 - File management
 - Device management
 - Communication

33

How many system calls?

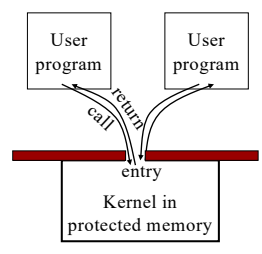

- ◆ 6th Edition Unix: ~45
- ◆ POSIX: ~130
- ◆ FreeBSD: ~130
- ◆ Linux: ~250
- ◆ Windows 7: > 900



34

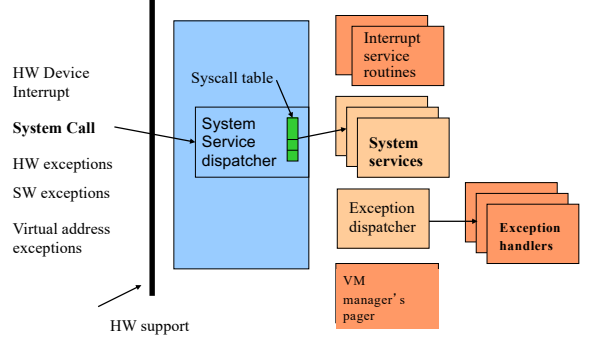

System Call Mechanism

- ◆ Assumptions
 - User code can be arbitrary
 - User code cannot modify kernel memory
- ◆ Design Issues
 - User makes a system call with parameters
 - The call mechanism switches code to kernel mode
 - Execute system call
 - Return with results

35

OS Kernel: Trap Handler

36


Interrupt, trap and syscall vector

- ◆ Table set up by OS kernel; pointers to code to run on different events

Processor Register	Interrupt Vector Table
→ handleTimerInterrupt() { }

→ handleDivideByZero() { }

→ handleSystemCall() { }



37

From <http://minnie.tuhs.org/UnixTree/V6>

V6/usr/sys/ken/sysent.c

```

Find as most [ 5 ] related files [ Search ]
[ ] including files from this version of Unix.

#
/*
/* This table is the switch used to transfer
* to the appropriate routine for processing a system call.
* Each row contains the number of arguments expected
* and a pointer to the routine.
*/
int sysent[]
{
    0, syscalls, /* 0 = indir */
    0, areset, /* 1 = exit */
    0, cfork, /* 2 = fork */
    2, cread, /* 3 = read */
    2, cwrite, /* 4 = write */
    2, copen, /* 5 = open */
    0, cclose, /* 6 = close */
    0, cwait, /* 7 = wait */
    2, cacct, /* 8 = acct */
    2, clink, /* 9 = link */
    1, cunlink, /* 10 = unlink */
    2, cexec, /* 11 = exec */
    1, cchdir, /* 12 = chdir */
    0, cptime, /* 13 = time */
    3, cchmod, /* 14 = chmod */
    2, cchown, /* 15 = chown */
    1, cbreak, /* 16 = break */
    2, cstat, /* 17 = stat */
    2, cseek, /* 18 = seek */
    0, csetgid, /* 19 = setgid */
    3, cmount, /* 21 = mount */
    1, cunmount, /* 22 = umount */
    0, csetuid, /* 23 = setuid */
    0, cgetuid, /* 24 = getuid */
    0, cstime, /* 25 = stime */
    3, cptime, /* 26 = prtime */
    0, cnoops, /* 27 = x */
    1, cfsync, /* 28 = fsync */
    0, cnoops, /* 29 = x */
    1, cswdate, /* 30 = sdate; inoperative */
    1, catty, /* 31 = tty */
    1, catty, /* 32 = gty */
    0, cnoops, /* 33 = x */
    0, cnoops, /* 34 = x */
    0, cnoops, /* 35 = sleep */
    0, cnoops, /* 36 = sync */
    1, ckill, /* 37 = kill */
    0, cgetstat, /* 38 = stat */
    0, cnoops, /* 39 = x */
    0, cnoops, /* 40 = x */
    0, cdup, /* 41 = dup */
    0, cpipe, /* 42 = pipe */
    1, ctime, /* 43 = times */
    4, cprofil, /* 44 = prof */
    0, cnoops, /* 45 = x */
    0, csetgid, /* 46 = setgid */
    0, csetgid, /* 47 = getgid */
    1, cswap, /* 48 = swg */
}
    
```

Passing Parameters

- ◆ Pass by registers
 - # of registers
 - # of usable registers
 - # of parameters in system call
 - Spill/fill code in compiler
- ◆ Pass by a memory vector (list)
 - Single register for starting address
 - Vector in user's memory
- ◆ Pass by stack
 - Similar to the memory vector
 - Procedure call convention

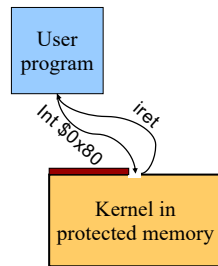


Library Stubs for System Calls

```

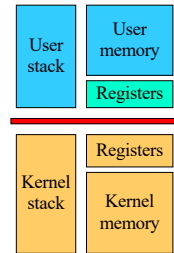
Example:
int read( int fd, char * buf, int size)
{
    move fd, buf, size to R1, R2, R3
    move READ to R0
    int $0x80
    move result to Rresult
}
    
```

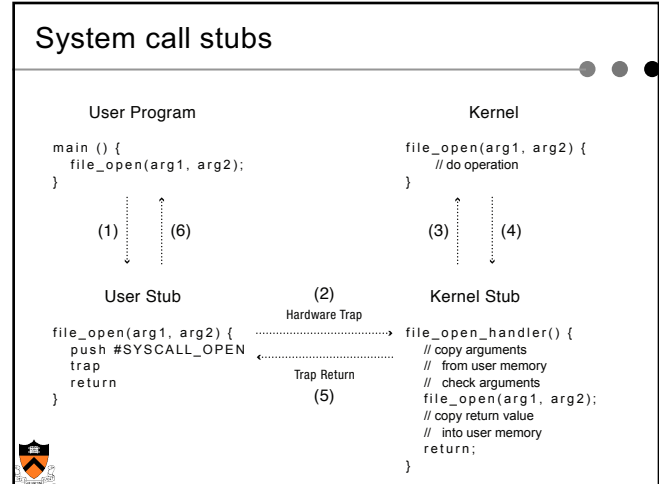
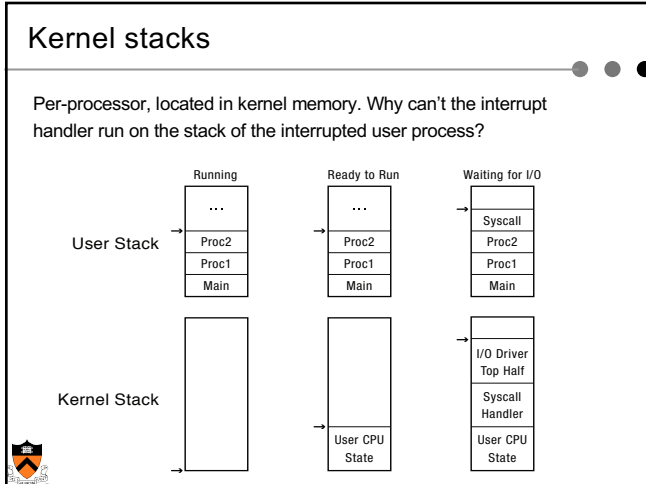
Linux: 80
NT: 2E



System Call Entry Point

- EntryPoint:**
- switch to kernel stack
 - save context
 - check R₀
 - call the real code pointed by R₀
 - place result in R_{result}
 - restore context
 - switch to user stack
 - iret (change to user mode and return)
- (Assumes passing parameters in registers)





- ### Design Issues
- ◆ System calls
 - There is one result register; what about more results?
 - How do we pass errors back to the caller?
 - ◆ System calls vs. library calls
 - What should be system calls?
 - What should be library calls?

Backward compatibility...

The Open Group Base Specifications Issue 6
 IEEE Std 1003.1, 2004 Edition
 Copyright © 2001-2004 The IEEE and The Open Group, All Rights reserved.

NAME

open - open a file

SYNOPSIS

```

[OH] #include <sys/stat.h>
#include <fcntl.h>
int open(const char *path, int oflag, ... );
    
```

The use of `open()` to create a regular file is preferable to the use of `creat()`, because the latter is redundant and included only for historical reasons.

Division of Labor (Separation Of Concerns)

Memory management example

- ◆ Kernel
 - Allocates “pages” with protection
 - Allocates a big chunk (many pages) to library
 - Does not care about small allocations
- ◆ Library
 - Provides malloc/free for allocation and deallocation
 - Applications use them to manage memory
 - When reaching the end, library asks kernel for more



48

Today

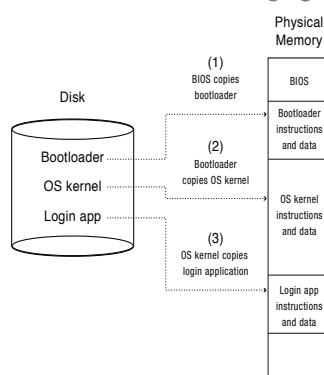
- ◆ Overview of OS functionality
- ◆ Overview of OS components
- ◆ Interacting with the OS
- ◆ Booting a Computer



50

Booting a Computer

- ◆ Power up a computer
- ◆ Processor reset
 - Set to known state
 - Jump to ROM code (for x86, this is the BIOS)
- ◆ Load in the boot loader from stable storage
- ◆ Jump to the boot loader
- ◆ Load the rest of the operating system
- ◆ Initialize and run



51

System Boot

- ◆ Power on (processor waits until Power Good Signal)
- ◆ Processor jumps to a fixed address, which is the start of the ROM BIOS program



COS318 Lec 2

52

ROM Bios Startup Program (1)

- ◆ POST (Power-On Self-Test)
 - Stop booting if fatal errors, and report
- ◆ Look for video card and execute built-in BIOS code (normally at C000h)
- ◆ Look for other devices ROM BIOS code
 - IDE/ATA disk ROM BIOS at C8000h 9=818200d)
- ◆ Display startup screen
 - BIOS information
- ◆ Execute more tests
 - memory
 - system inventory



COS318 Lec 2

53

ROM BIOS startup program (2)

- ◆ Look for logical devices
 - Label them
 - Serial ports: COM 1, 2, 3, 4
 - Parallel ports: LPT 1, 2, 3
 - Assign each an I/O address and interrupt numbers
- ◆ Detect and configure Plug-and-Play (PnP) devices
- ◆ Display configuration information on screen



COS318 Lec 2

54

ROM BIOS startup program (3)

- ◆ Search for a drive to BOOT from
 - Hard disk or USB drive or CD/DVD
- ◆ Load code in boot sector
- ◆ Execute boot loader
- ◆ Boot loader loads program to be booted
 - If no OS: "Non-system disk or disk error - Replace and press any key when ready"
- ◆ Transfer control to loaded program
 - Could be OS or another feature-rich bootloader (e.g. GRUB), which then loads the actual OS



COS318 Lec 2

55

Summary

- ◆ Protection mechanism
 - Architecture support: two modes
 - Software traps (exceptions)
- ◆ OS structures
 - Monolithic, layered, microkernel and virtual machine
- ◆ System calls
 - Implementation
 - Design issues
 - Tradeoffs with library calls



56