



COS 318: Operating Systems

Overview

Kai Li
Computer Science Department
Princeton University

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



Important Times



- ◆ Design review:
 - 9/25 during 6-9pm, 010 Friends center
- ◆ Project 1 due:
 - 10/2 at 11:59pm
- ◆ Precepts:
 - Wed: 8:30-9:30pm, 105 CS building
- ◆ Reminder:
 - Register to the cos318 mailing list today!



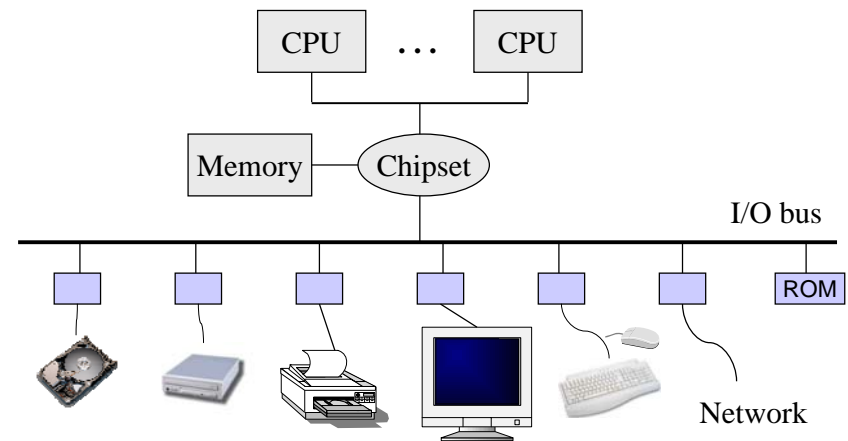
Today



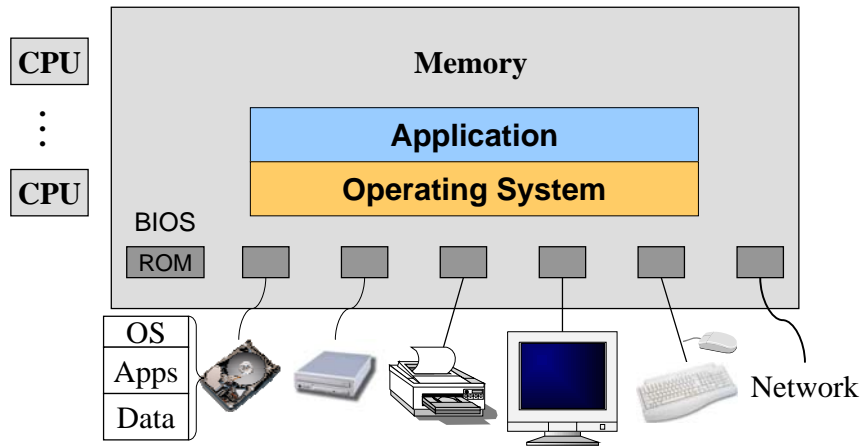
- ◆ Overview of operating system
- ◆ IA32 bootstrap



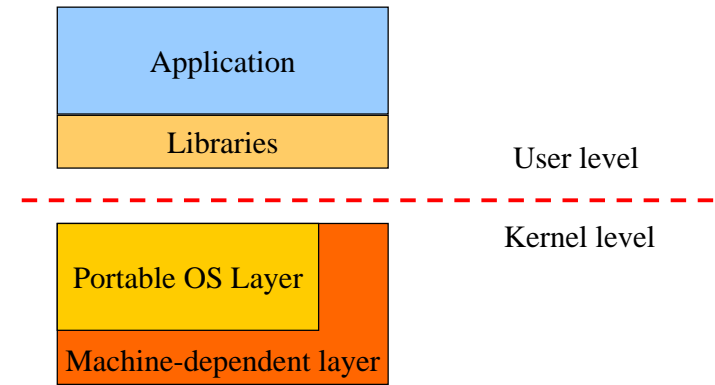
Hardware of A Typical Computer



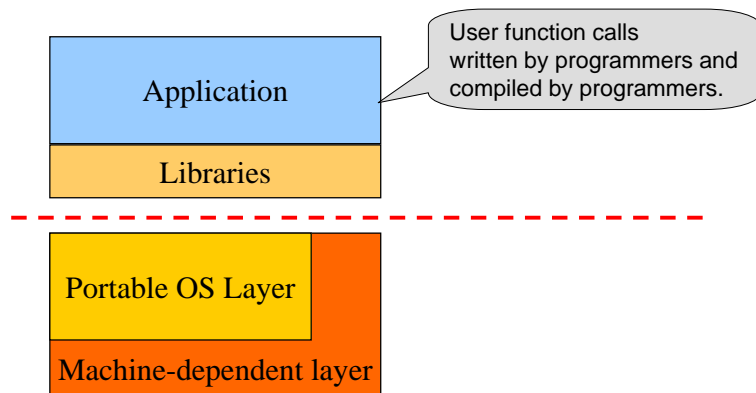
A Typical Computer System



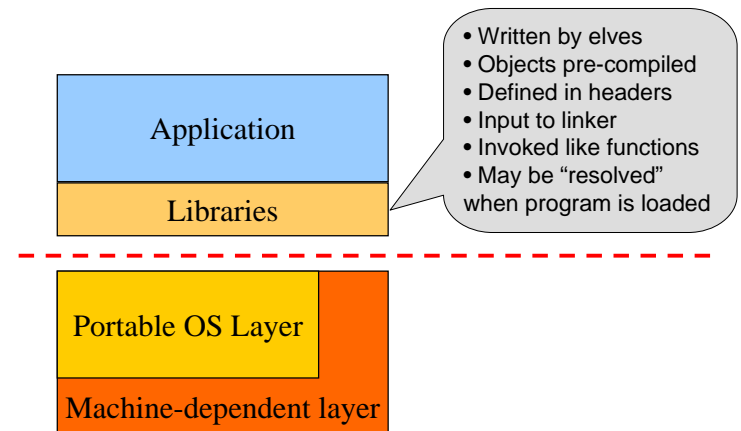
Typical Unix OS Structure



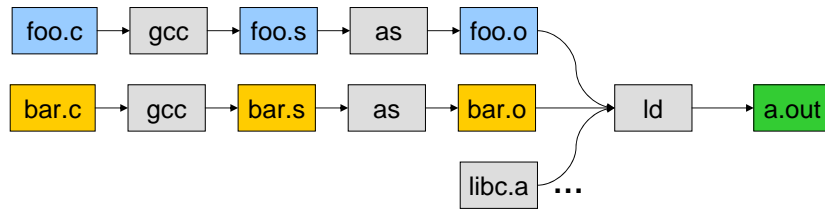
Typical Unix OS Structure



Typical Unix OS Structure



Pipeline of Creating An Executable File



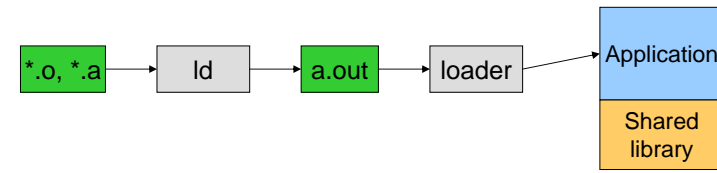
- ◆ gcc can compile, assemble, and link together
- ◆ Compiler part of gcc compiles a program into assembly
- ◆ Assembler compiles assembly code into relocatable object file
- ◆ Linker links object files into an executable
- ◆ For more information:
 - Read man page of a.out, elf, ld, and nm
 - Read the document of ELF



9

Execution (Run An Application)

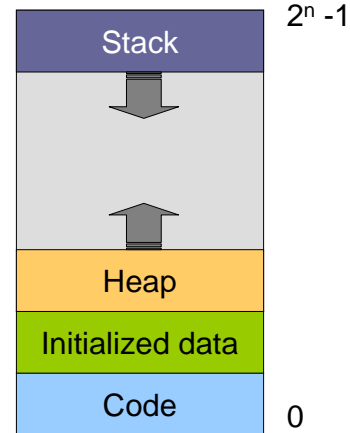
- ◆ On Unix, “loader” does the job
 - Read an executable file
 - Layout the code, data, heap and stack
 - Dynamically link to shared libraries
 - Prepare for the OS kernel to run the application



10

What's An Application?

- ◆ Four segments
 - Code/Text – instructions
 - Data – initialized global variables
 - Stack
 - Heap
- ◆ Why?
 - Separate code and data
 - Stack and heap go towards each other



11

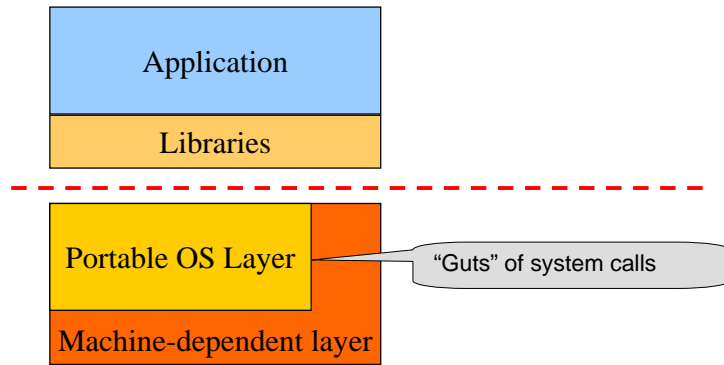
Responsibilities

- ◆ Stack
 - Layout by compiler
 - Allocate/deallocate by process creation (fork) and termination
 - Names are relative off of stack pointer and entirely local
- ◆ Heap
 - Linker and loader say the starting address
 - Allocate/deallocate by library calls such as malloc() and free()
 - Application program use the library calls to manage
- ◆ Global data/code
 - Compiler allocate statically
 - Compiler emit names and symbolic references
 - Linker translate references and relocate addresses
 - Loader finally lay them out in memory



12

Typical Unix OS Structure



13

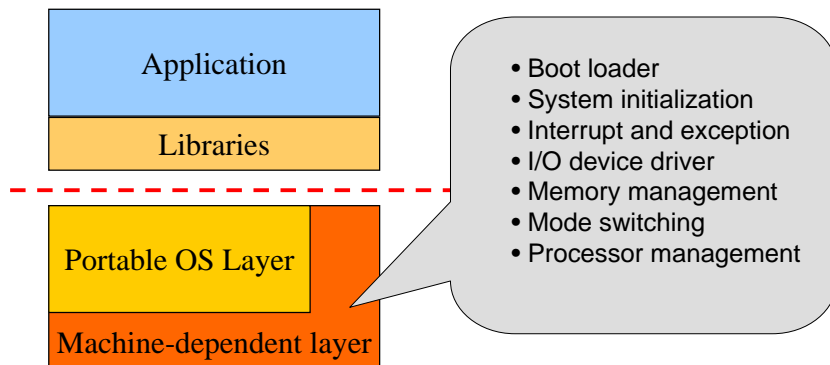
OS Service Examples

- ◆ Examples that are not provided at user level
 - System calls: file open, close, read and write
 - `while (1) ;`
 - Control the CPU so that users won't stuck by running
 - Protection:
 - Keep user programs from crashing OS
 - Keep user programs from crashing each other
- ◆ System calls are typically traps or exceptions
 - System calls are implemented in the kernel
 - When finishing the service, a system returns to the user code



14

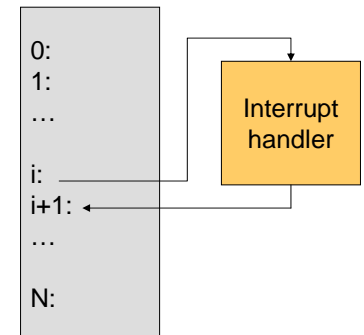
Typical Unix OS Structure



15

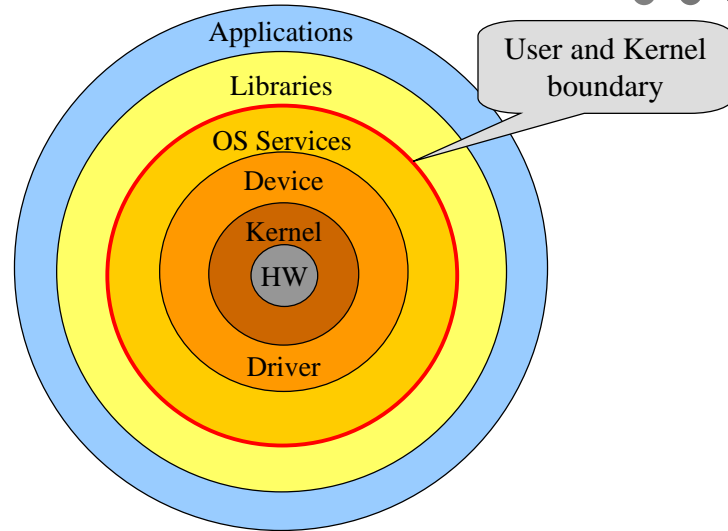
Interrupts

- ◆ Raised by external events
- ◆ Interrupt handler is in the kernel
 - Switch to another process
 - Overlap I/O with CPU
 - ...
- ◆ Eventually resume the interrupted process



16

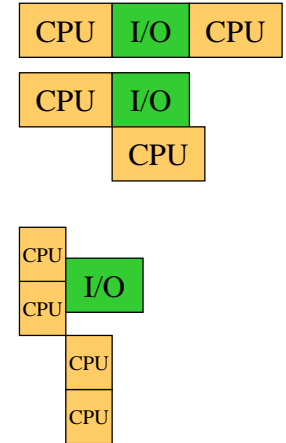
Software "Onion" Layers



17

Processor Management

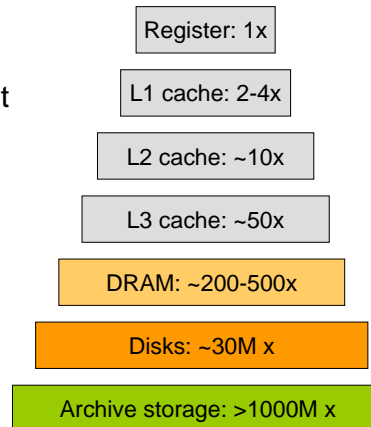
- ◆ Goals
 - Overlap between I/O and computation
 - Time sharing
 - Multiple CPU allocations
- ◆ Issues
 - Do not waste CPU resources
 - Synchronization and mutual exclusion
 - Fairness and deadlock free



18

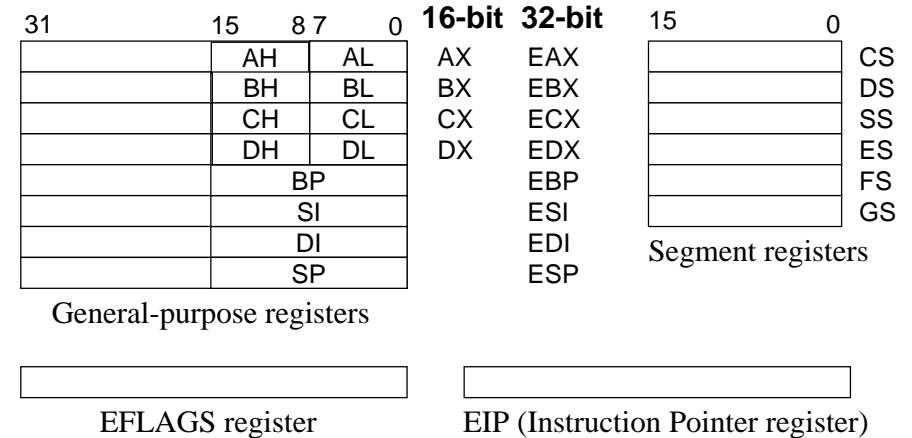
Memory Management

- ◆ Goals
 - Support programs to run
 - Allocation and management
 - Transfers from and to secondary storage
- ◆ Issues
 - Efficiency & convenience
 - Fairness
 - Protection



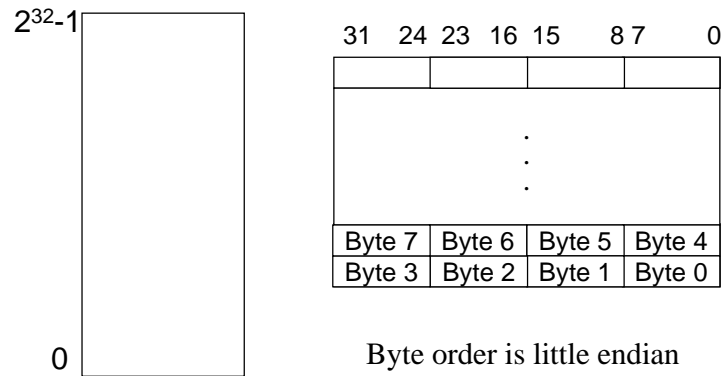
19

IA32 Architecture Registers



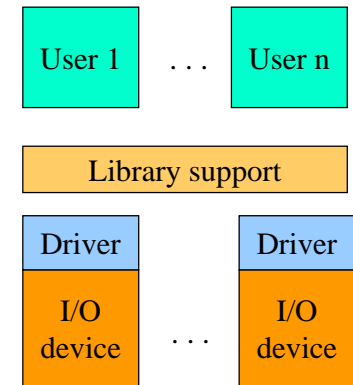
20

IA32 Memory



I/O Device Management

- ◆ Goals
 - Interactions between devices and applications
 - Ability to plug in new devices
- ◆ Issues
 - Efficiency
 - Fairness
 - Protection and sharing



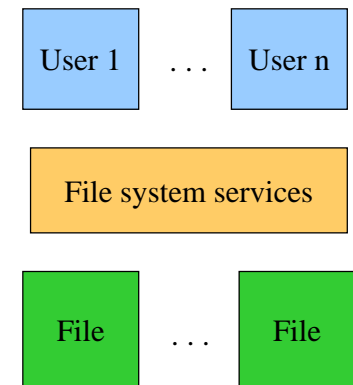
Window Systems

- ◆ All in the kernel (Windows)
 - Pros: efficient
 - Cons: difficult to develop new services
- ◆ All at user level
 - Pros: easy to develop new apps
 - Cons: protection
- ◆ Split between user and kernel (Unix)
 - Kernel: display driver and mouse driver
 - User: the rest



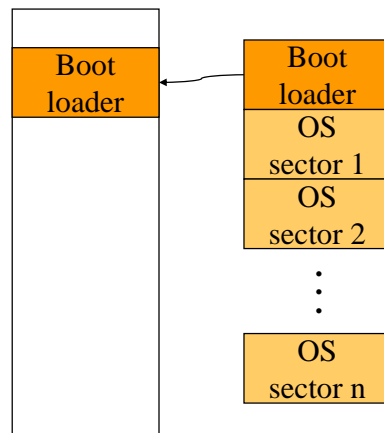
File System

- ◆ A typical file system
 - Open a file with authentication
 - Read/write data in files
 - Close a file
- ◆ Can the services be moved to user level?



Bootstrapping

- ◆ Power up a computer
- ◆ Processor reset
 - Set to known state
 - Jump to ROM code (BIOS is in ROM)
- ◆ Load in the boot loader from stable storage
- ◆ Jump to the boot loader
- ◆ Load the rest of the operating system
- ◆ Initialize and run
- ◆ Question: Can BIOS be on disk?



25

System Boot

- ◆ Power on (processor waits until Power Good Signal)
- ◆ Processor jumps on a PC (“Intel Inside”) to address FFFF0h (Maps to FFFFFFF0h= $2^{32}-16$)
 - $1\text{M} = 1,048,576 = 2^{20} = \text{FFFFFh} + 1$
 - $\text{FFFFFh} = \text{FFFFF0h} + 16$ is the end of the (first 1MB of system memory)
 - The original PC using Intel 8088 had 20 address lines :-)
- ◆ (FFFFFFF0h) is a JMP instruction to the ROM BIOS startup program



26

ROM BIOS Startup (1)

- ◆ POST (Power-On Self-Test)
 - If pass then AX:=0; DH:=5 (Pentium);
 - Stop booting if fatal errors, and report
- ◆ Look for video card and execute built-in ROM BIOS code (normally at C000h)
- ◆ Look for other devices ROM BIOS code
 - IDE/ATA disk ROM BIOS at C8000h (=819,200d)
 - SCSI disks may provide their own BIOS
- ◆ Display startup screen
 - BIOS information
- ◆ Execute more tests
 - memory
 - system inventory



27

ROM BIOS Startup (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 IRQ
- ◆ Detect and configure PnP devices
- ◆ Display configuration information on screen



28

ROM BIOS Startup (3)



- ◆ Search for a drive to BOOT from
 - Floppy or Hard disk
 - Boot at cylinder 0, head 0, sector 1
- ◆ 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
- ◆ Is it okay to boot at first sector on the floppy or disk?



Ways to Develop An Operating System



- ◆ A hardware simulator
- ◆ A virtual machine
- ◆ A good kernel debugger
 - When OS crashes, always goes to the debugger
 - Debugging over the network

