



Modern Operating Systems

Chapter 1 - Introduction

Zhang Yang


cszyang@scut.edu.cn

Spring 2022



Contents of this lecture

- 1.1 What is an Operating System?
- 1.2 History of Operating Systems
- 1.3 Computer Hardware Review
- 1.4 The Operating System Zoo
- 1.5 Operating System Concepts
- 1.6 System Calls
- 1.7 Operating System Structure
- Summary

- 
- 1.1 What is an Operating System?
 - 1.2 History of Operating Systems
 - 1.3 Computer Hardware Review
 - 1.4 The Operating System Zoo
 - 1.5 Operating System Concepts
 - 1.6 System Calls
 - 1.7 Operating System Structure
 - Summary

What is an Operating System?(1)

- A modern computer consists of
 - One or more processors
 - Main memory
 - Disks
 - Printers
 - Various input/output devices
- Managing all these components requires a layer of software – the operating system.

What is an Operating System ? (2)

■ An Informal Definition of OS

- The OS is a collection of one or more software modules that manages and controls the resources of a computer or other computing or electronic device, and gives users and programs an interface to utilize these resources.

What is an Operating System ? (3)

■ An Informal Definition of OS (ctd.)

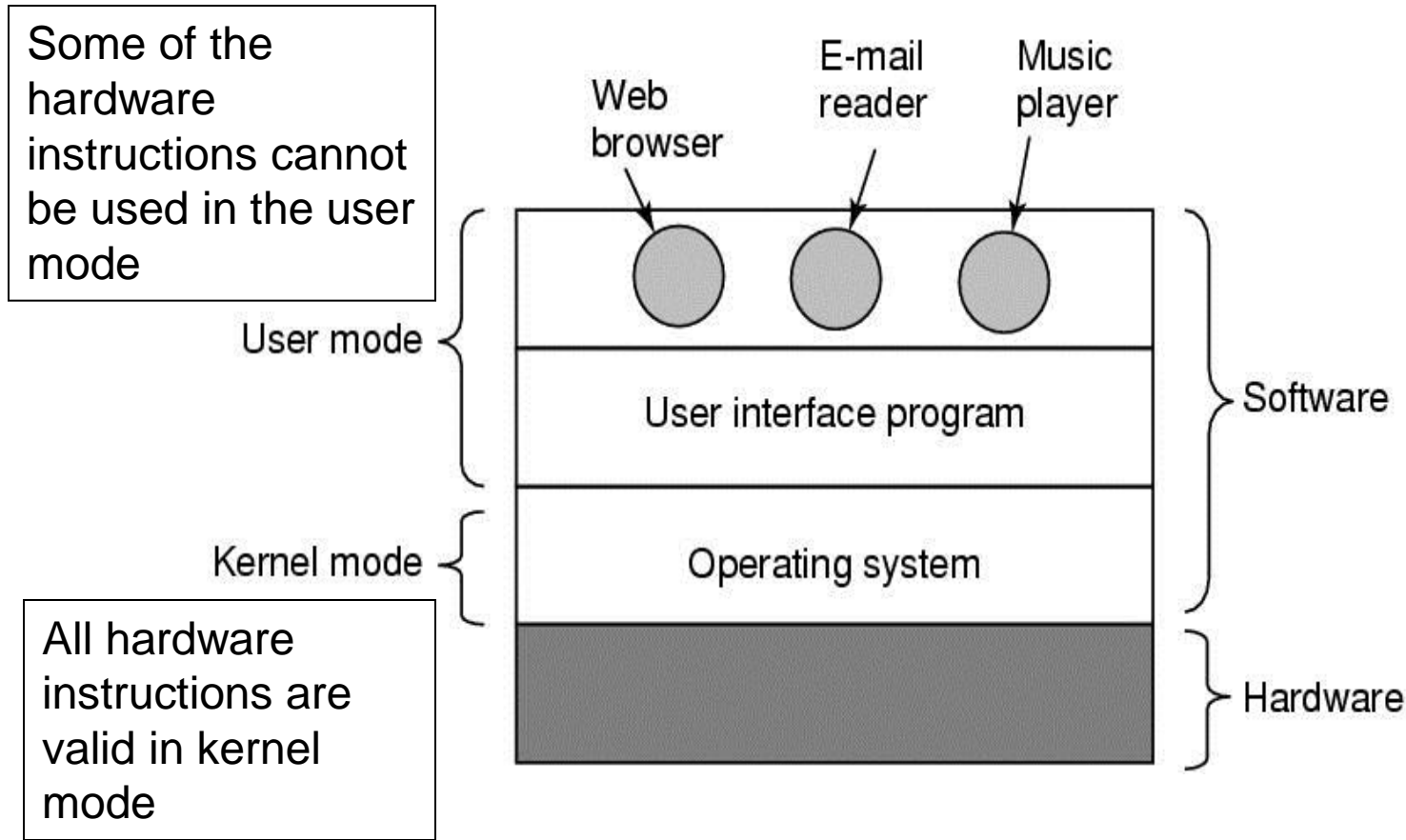


Figure 1-1. Where the operating system fits in.

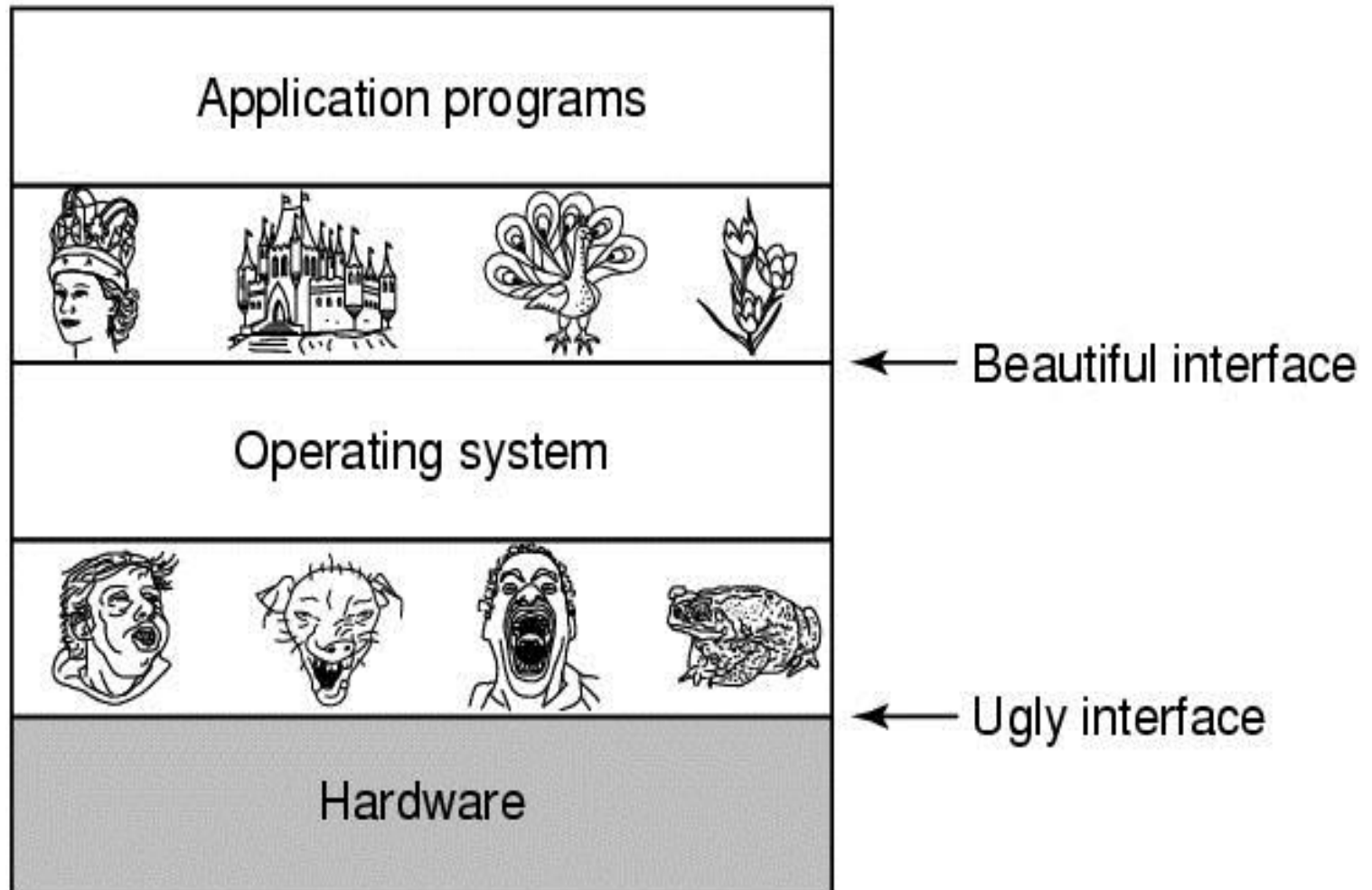
What is an Operating System?(4)

■ The Operating System as an Extended Machine

- The OS hides the messy details which must be performed--Provide standard Library (i.e., abstract resources)
- The OS implements a virtual machine on top of the machine that is easier to use (i.e. write programs for)
- This is the “top down” (or external view) of an operating system.

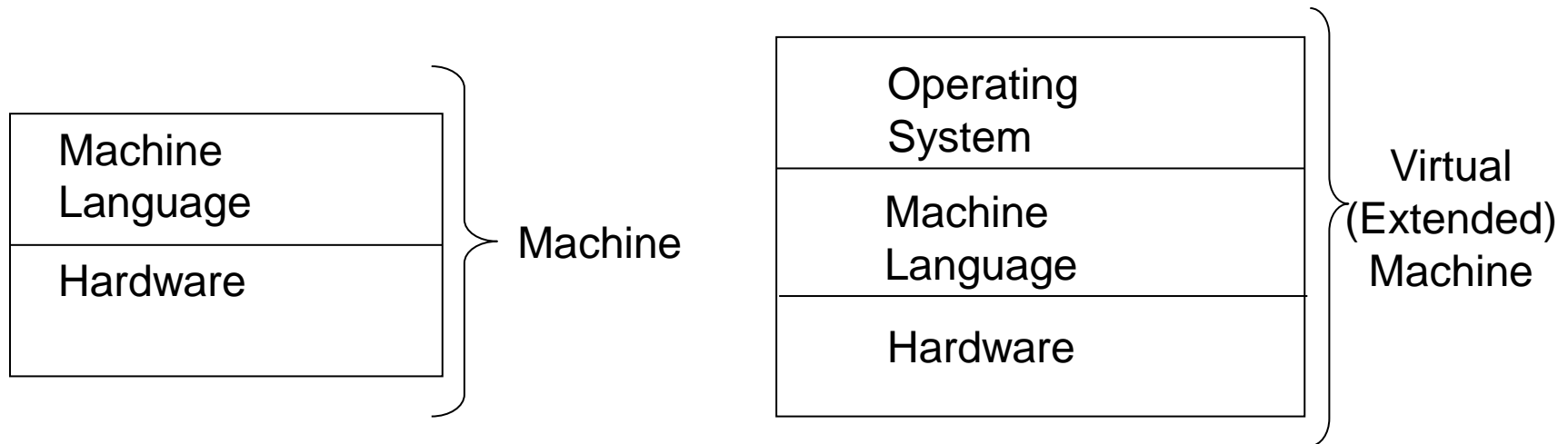
What is an Operating System?(5)

- The Operating System as an Extended Machine (ctd.)



What is an Operating System?(6)

■ The Operating System as an Extended Machine (ctd.)



What is an Operating System?(7)

■ The Operating System as an Extended Machine (ctd.)

□ Example

- In a system where there is no OS installed, we should consider some hardware work as:

(Assuming an MC 6800 computer hardware)

LDAA \$80 → Loading the number at memory location 80

LDAB \$81 → Loading the number at memory location 81

ADDB → Adding these two numbers

STAA \$55 → Storing the sum to memory location 55

- As seen, we considered memory locations and used our hardware knowledge of the system.

What is an Operating System?(8)

■ The Operating System as an Extended Machine (ctd.)

□ Example (ctd.)

- In an OS installed machine, since we have an intermediate layer, our programs obtain *some advantage of mobility* by not dealing with hardware.
- For example, the above program segment would not work for an 8086 machine, where as the
“ $c = a + b ;$ ”
syntax will be suitable for both.


What is an Operating System? (9)

- The Operating System as a Resource Manager
 - Manages and protects multiple computer resources: CPU, Processes, Internal/External memory, Tasks, Applications, Users, Communication channels, etc...
 - Handles and allocates resources to multiple users or multiple programs running at the same time and space (e.g., processor time, memory, I/O devices).
 - Decides between conflicting requests for efficient and fair resource use (e.g., maximize throughput, minimize response time).
 - This is the “bottom-up” (or internal) view of an OS



What Is an OS? Real life example?

- 2-3 persons group discussion
- 2 minutes

- 
- 1.1 What is an Operating System?
 - 1.2 History of Operating Systems
 - 1.3 Computer Hardware Review
 - 1.4 The Operating System Zoo
 - 1.5 Operating System Concepts
 - 1.6 System Calls
 - 1.7 Operating System Structure
 - Summary



History of Operating Systems (1)

- Two distinct phases of history
 - Phase 1: Computers are expensive
 - Goal: Use computer's time efficiently
 - Maximize throughput (i.e., jobs per second)
 - Maximize utilization (i.e., percentage busy)
 - Phase 2: Computers are inexpensive
 - Goal: Use people's time efficiently
 - Minimize response time



History of Operating Systems (2)

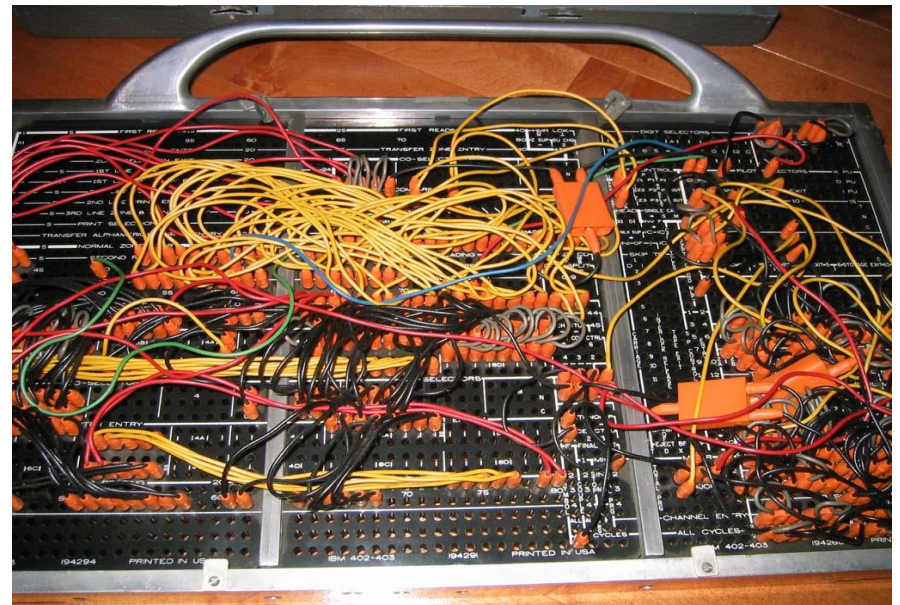
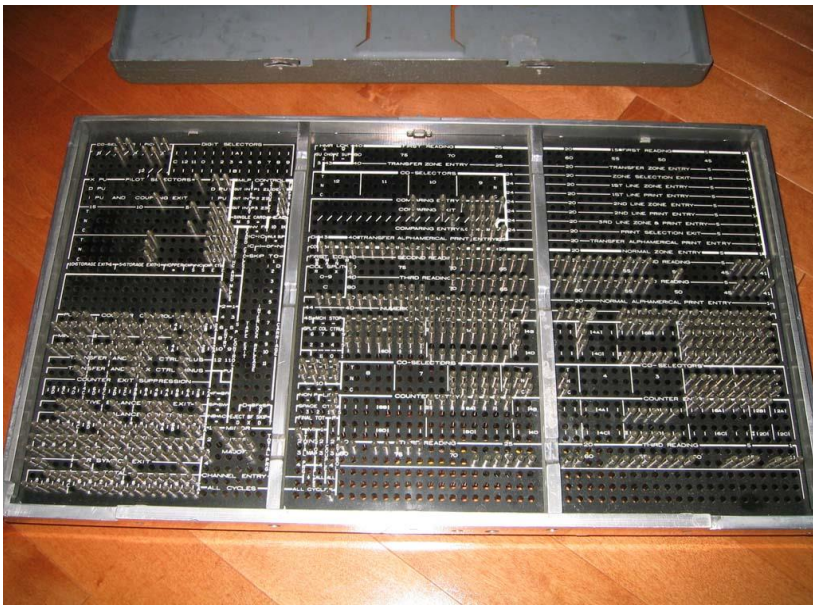
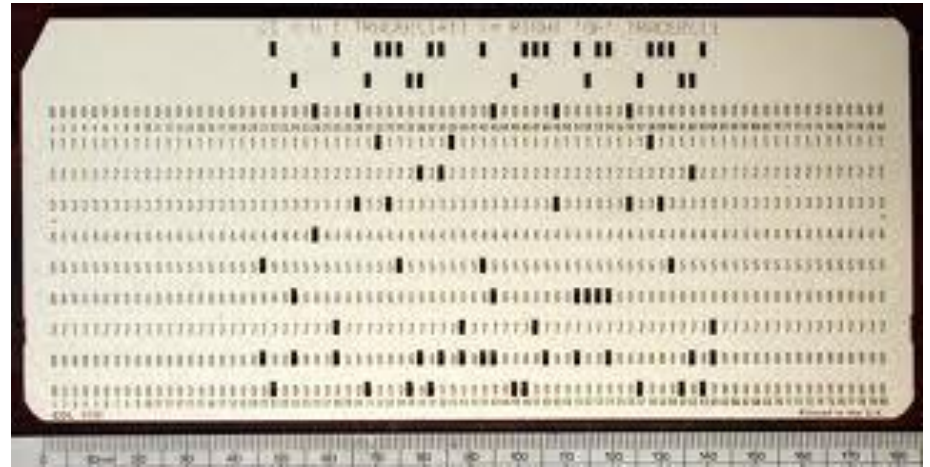
- First Generation 1945 – 1955
 - Vacuum tubes, plug boards, punched cards (no OS)
- Second Generation 1955 – 1965
 - Transistors, batch systems
- Third Generation 1965 – 1980
 - ICs, multiprogramming, spooling, timesharing
- Fourth Generation 1980 – present
 - Large scale integration
 - Personal computers, hand-held devices, sensors
- Fifth Generation 1990 – present
 - Mobile computers



History of Operating Systems (3)

- First Generation 1945 – 1955: direct input
 - Vacuum tubes, plug boards
 - Machine language, no OS
 - Punched cards(by the early 1950s)
 - User has complete control of the machine
 - Utilization is very low; set up time is high

History of Operating Systems (4)



History of Operating Systems (5)

■ Second Generation 1955 – 1965

- Transistors, batch systems
- Motivation: Increase CPU utilization
- Use a slow computer to aggregate programs and feed into fast computer.
- Same on the output side.
- Batch: Group of jobs submitted together
 - Operator collects jobs; orders efficiently; runs one at a time
- Mainframe, job (a program or set of programs), FORTRAN or assembler, punched cards
- FMS (the Fortran Monitor System, IBM's OS for the 709), IBSYS (IBM's OS for the 7090 and 7094), UMES (The University of Michigan Executive System, for IBM 7094)

History of Operating System (6)

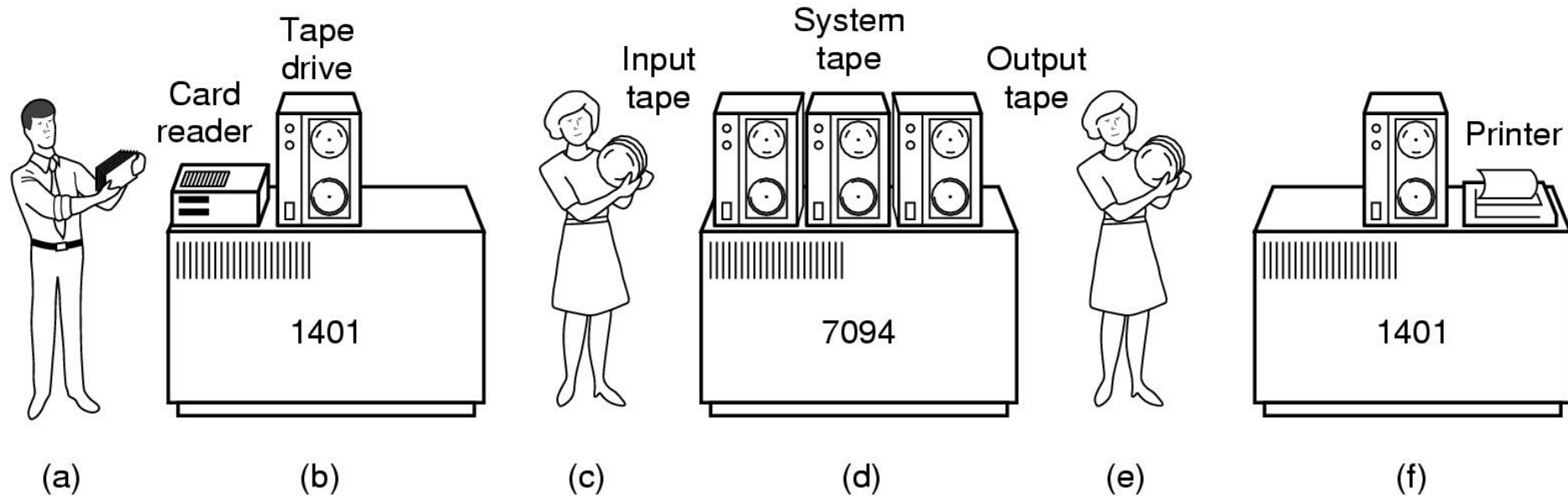


Figure 1-3. An early batch system

- (a) Programmers bring cards to 1401
- (b) 1401 reads batch of jobs onto tape
- (c) Operator carries input tape to 7094
- (d) 7094 does computing
- (e) Operator carries output tape to 1401
- (f) 1401 prints output

History of Operating Systems (7)

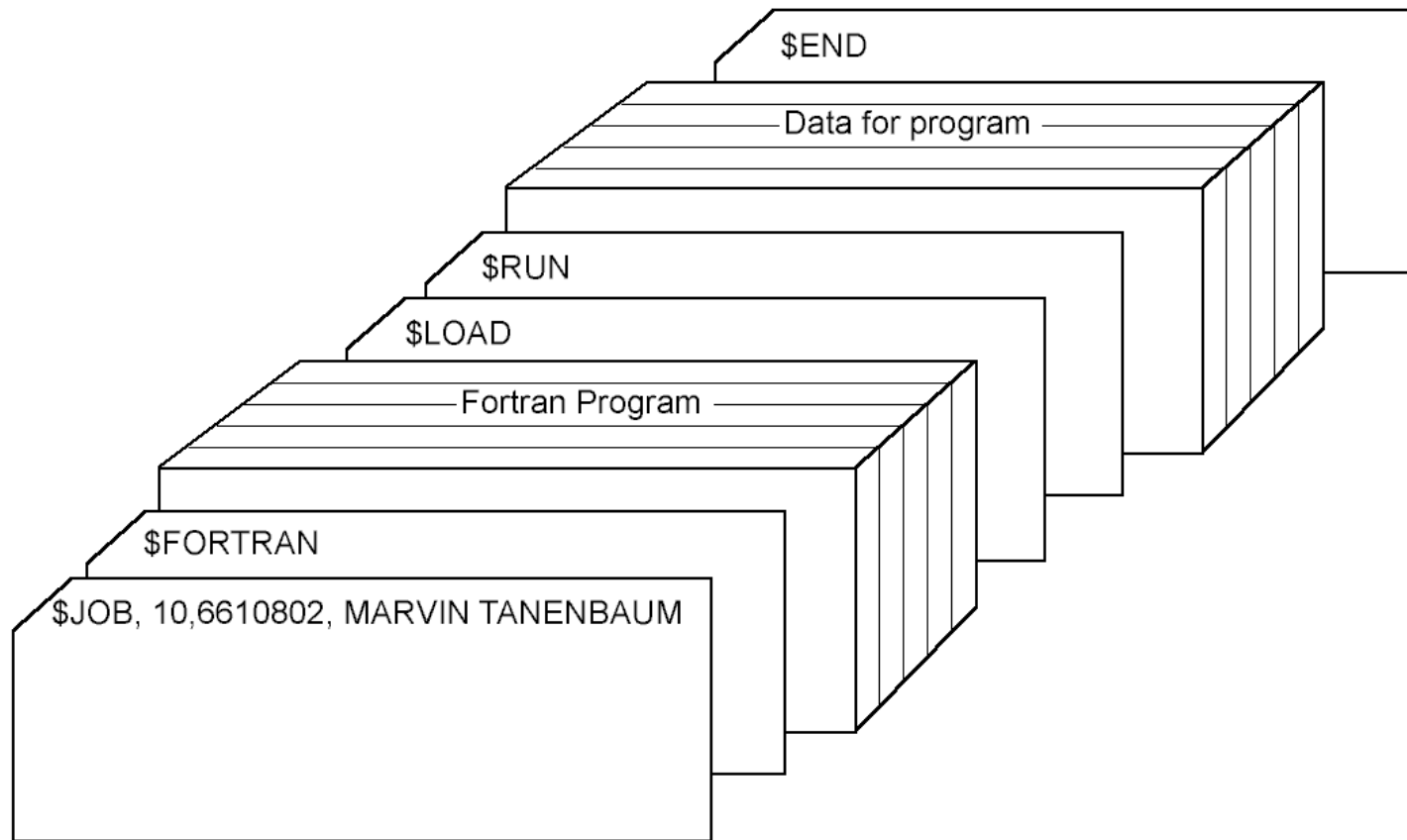


Figure 1-4. Structure of a typical FMS job



History of Operating System (8)

■ Second Generation 1955 – 1965 (ctd.)

□ Advantages

- Amortize setup costs over many jobs.
- Operator more skilled at loading tapes.
- Keep machine busy while programmer thinks.
- Improves throughput and utilization.

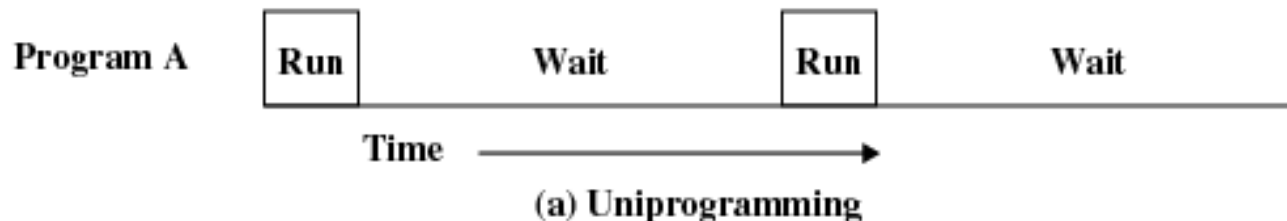
□ Problems

- User must wait until batch is done for results.
- Machine idle when job is reading from cards and writing to printers.

History of Operating Systems (9)

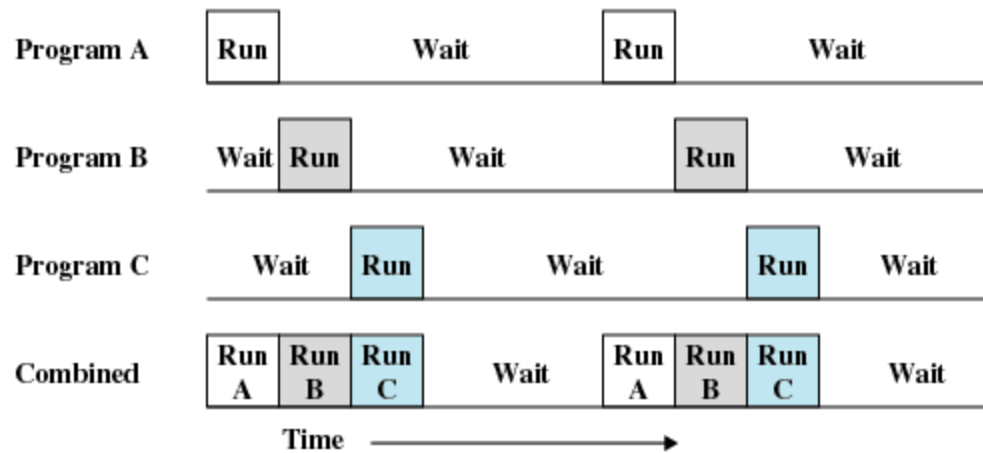
■ Third Generation 1965 – 1980

- ICs, multiprogramming, spooling, timesharing
- A family of software compatible computers, IBM 360 series, OS/360 (M) for IBM System 360, System 370 and System 4300.
- Uniprogramming
 - Processor must wait for I/O instruction to complete before preceding



History of Operating Systems (10)

- Third Generation 1965 – 1980 (ctd.)
 - Multiprogramming
 - When one program needs to wait for I/O, the processor can switch to the other program



(c) Multiprogramming with three programs

History of Operating Systems (11)

■ Third Generation 1965 – 1980 (ctd.)

□ Multiprogramming (ctd.)

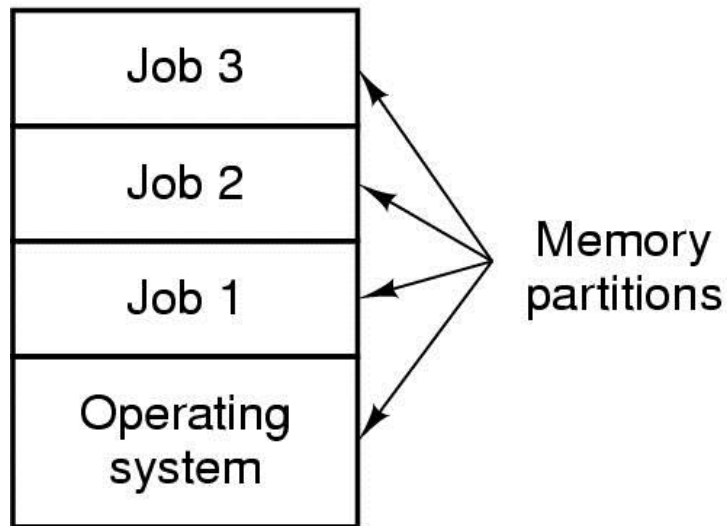


Figure 1-5. A multiprogramming system with three jobs in memory

- Multiple jobs in memory
 - Protected from one another.
- Operating system protected from each job as well.
- Resources (time, hardware) split between jobs.
- Still not interactive
 - User submits job.
 - Computer runs it.
 - User gets results minutes (hours, days) later.

History of Operating Systems (12)

- Third Generation 1965 – 1980 (ctd.)
 - Spooling
 - Original batch systems used tape drives.
 - Later batch systems used disks for buffering.
 - Operator read cards onto disk attached to the computer.
 - Computer read jobs from disk.
 - Computer wrote job results to disk.
 - Operator directed that job results be printed from disk.
 - Disks enabled simultaneous peripheral operation on line (spooling)
 - Computer overlapped I/O of one job with execution of another.
 - Better utilization of the expensive CPU.
 - Still only one job active at any given time.



History of Operating Systems (13)

■ Third Generation 1965 – 1980 (ctd.)

□ Timesharing

- Using multiprogramming to handle multiple interactive jobs.
- Processor's time is shared among multiple users.
- Multiple users simultaneously access the system through terminals.

□ Compatible Time-Sharing System (CTSS)

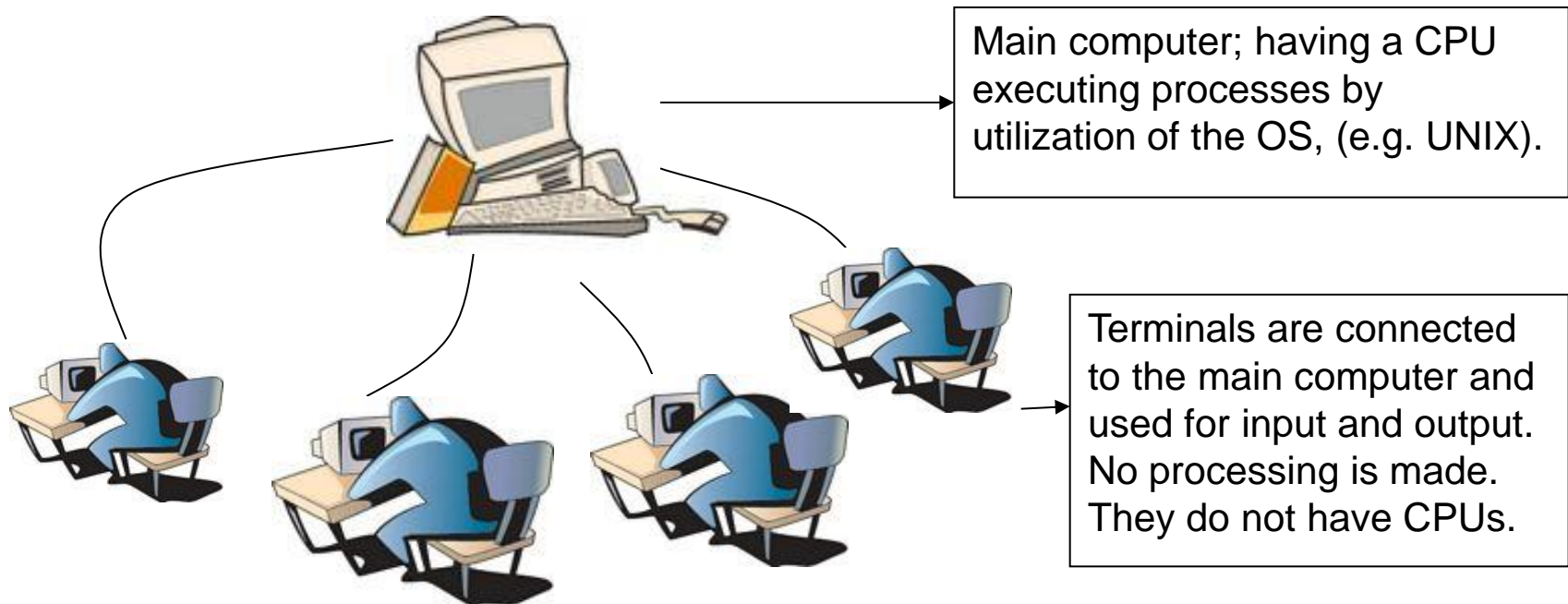
- First general-purpose time-sharing system developed at MIT.

□ MULTiplexed Information and Computing Services (MULTICS)

History of Operating Systems (14)

■ Third Generation 1965 – 1980 (ctd.)

□ Timesharing (ctd.)



History of Operating Systems (15)

■ Fourth Generation 1980 – present

□ Windows is the other Dominant Family of OS

- CP/M was first PC Operating System.
- DOS (Disk Operating System) later to be renamed MS-DOS dominated the PC Market.
- Windows provided GUI for DOS (up to Windows 3.x).
- Windows 95 was first free standing version of Windows.
- Followed by Windows 98.
- Windows NT was first 32-bit OS.
- Windows 2000 was introduced as Windows NT version 4.
- Windows VISTA
- Windows 7
- Windows 8
- Windows 10


History of Operating Systems (16)

■ Fourth Generation 1980 – present (ctd.)

- Unix
- Linux

■ Fifth Generation 1990 – present

- Combining telephony and computing in a phone-like device since the 1970s
- The first real smartphone: Nokia N9000, mid-1990s
- Symbian OS
- RIM's Blackberry OS introduced for 2002
- Apple's iOS released for the first iPhone in 2007
- Google's Android in 2008

- 
- 1.1 What is an Operating System?
 - 1.2 History of Operating Systems
 - 1.3 Computer Hardware Review
 - 1.4 The Operating System Zoo
 - 1.5 Operating System Concepts
 - 1.6 System Calls
 - 1.7 Operating System Structure
 - Summary

A Typical Computer System

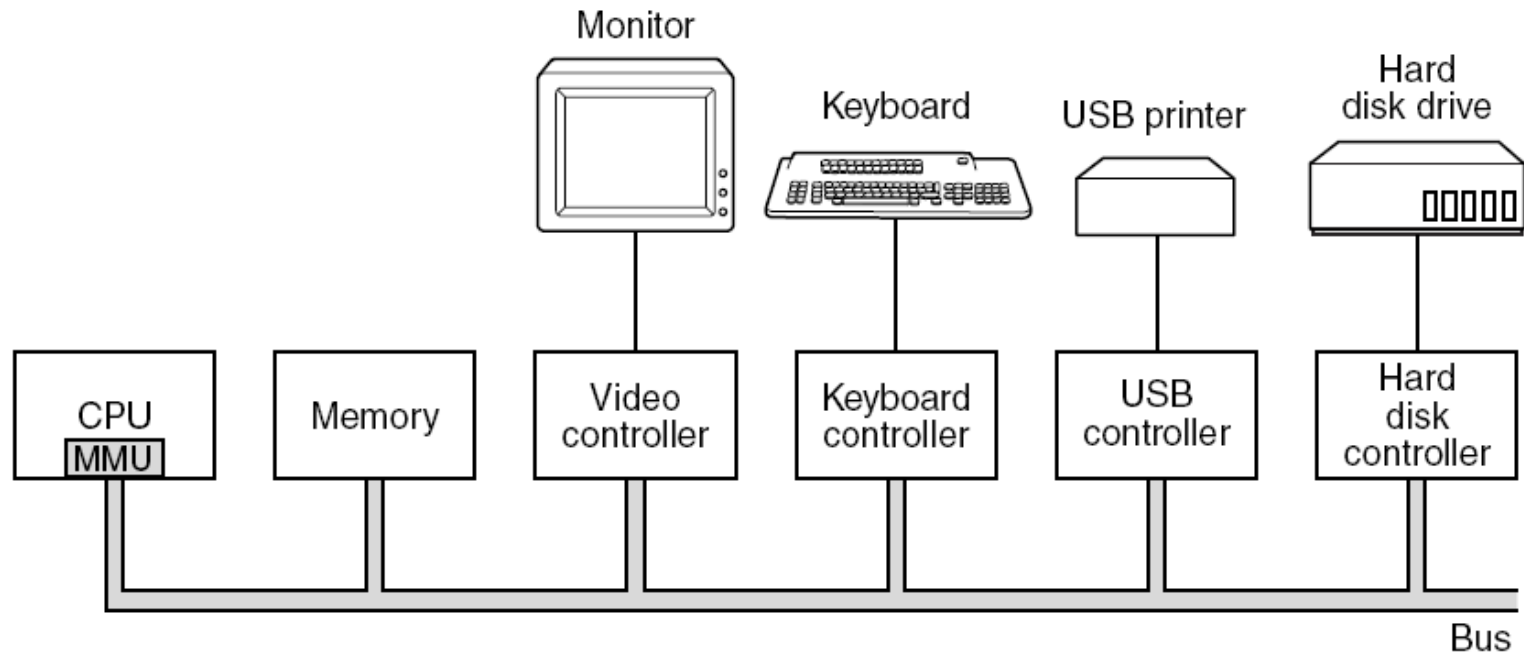


Figure 1-6. Some of the components of a simple personal computer.

Processors

- Each CPU has a specific set of instructions.
- All CPUs contain
 - General registers inside to hold key variables and temporary results.
 - Special registers visible to the programmer.
 - Program counter contains the memory address of the next instruction to be fetched.
 - Stack pointer points to the top of the current stack in memory.
 - PSW (Program Status Word) contains the condition code bits which are set by comparison instructions, the CPU priority, the mode (user or kernel) and various other control bits.

How Processors Work (1)

- Execute Instructions

- CPU Cycles

- Fetch (from mem) → decode → execute
 - Program Counter (PC)
 - When is PC changed?
 - Pipeline: fetch $n+2$ while decode $n+1$ while execute n
 - Superscalar

How Processors Work (2)

- Execute Instructions (ctd.)

- CPU Cycles (ctd.)

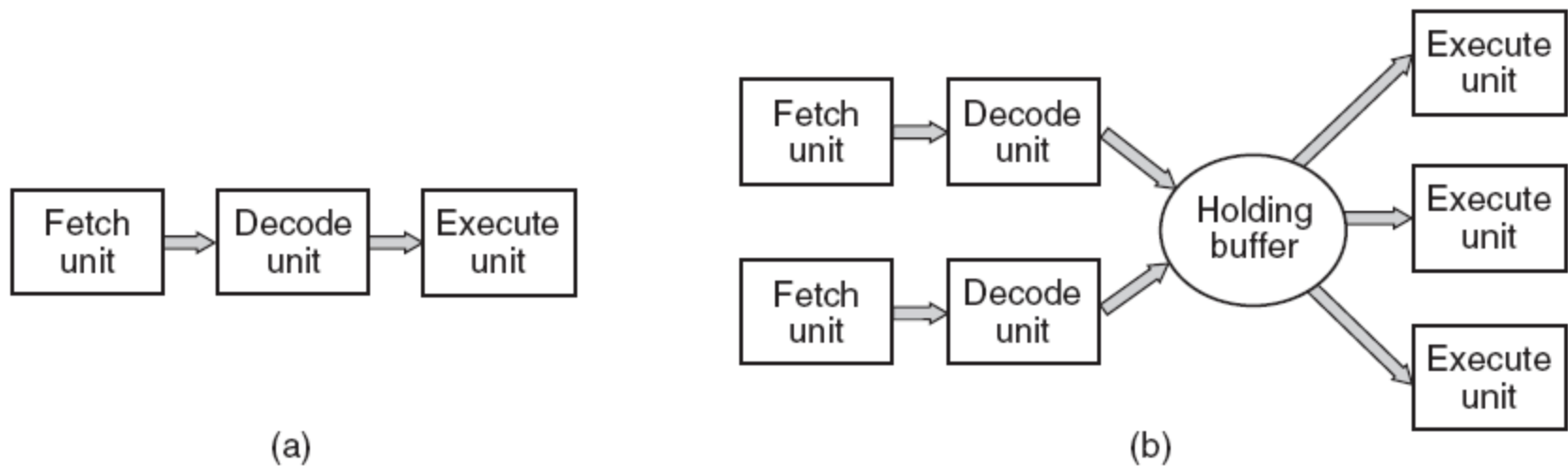


Figure 1-7. (a) A three-stage pipeline. (b) A superscalar CPU.

How Processors Work (3)

- Execute Instructions (ctd.)

- Two Modes of CPU (why?)

- User mode (a subset of instructions)

- Kernel mode (all instruction)

- Trap (special instruction)

- Switch from user mode to kernel mode and starts the OS.

Multithreaded and Multicore Chips

- Multithreading or Hyperthreading
 - Allow the CPU to hold the state of two different threads and then switch back and forth on a nanosecond time scale.

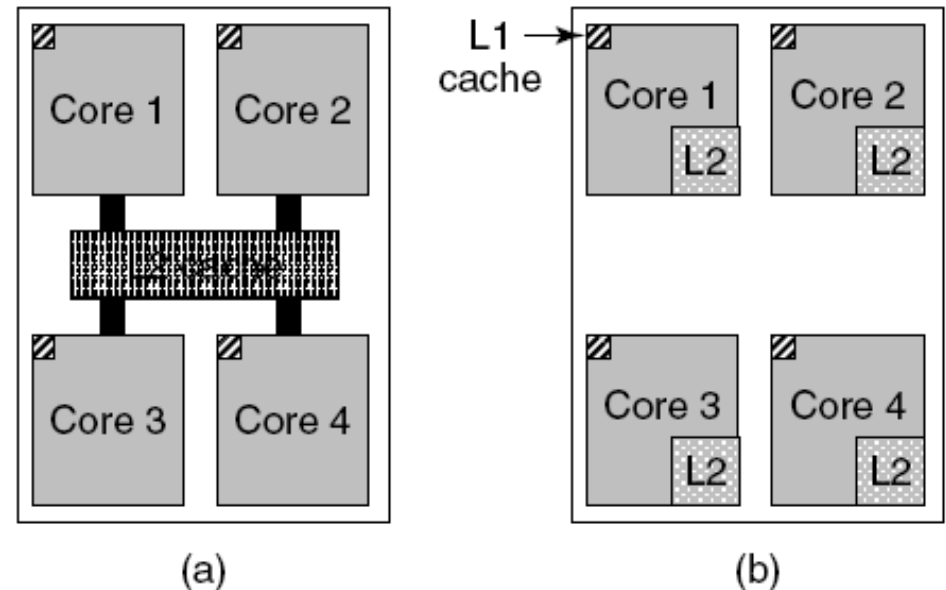


Figure 1-8. (a) A quad-core chip with a shared L2 cache. (Intel)
(b) A quad-core chip with separate L2 caches. (AMD)

Memory-Storage Hierarchy

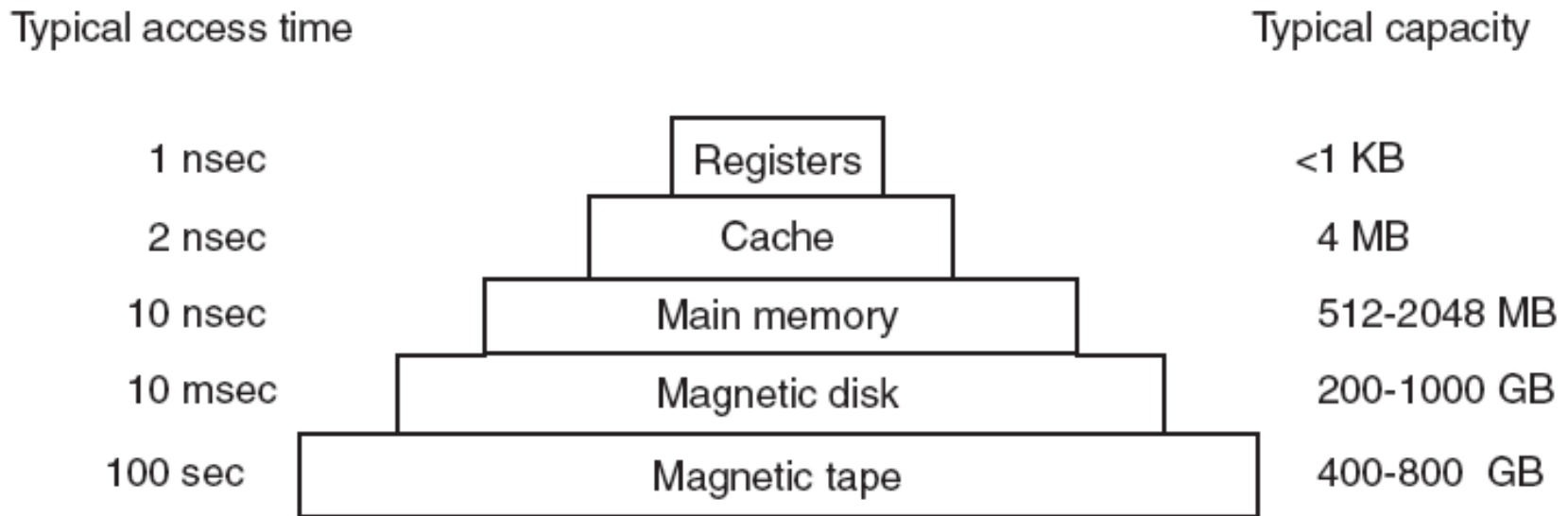
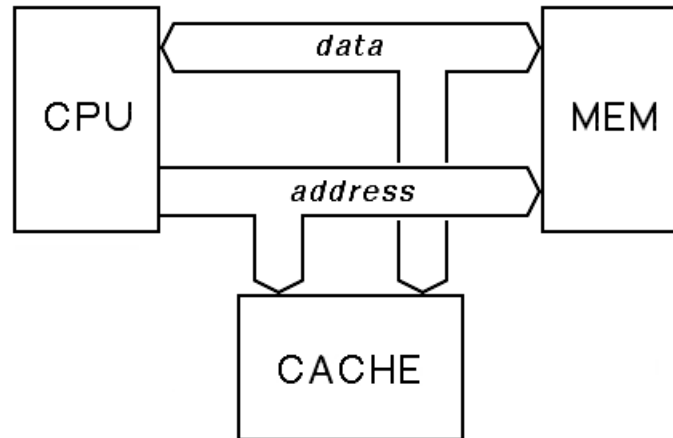


Figure 1-9. A typical memory hierarchy.

The numbers are very rough approximations.

Real life analogy?

Cache (1)



■ Cache Hit

- No need to access memory

■ Cache Miss

- Data obtained from memory, possibly update cache



Cache (2)

- Questions when dealing with cache
 - When to put a new item into the cache.
 - Which cache line to put the new item in.
 - Which item to remove from the cache when a slot is needed.
 - Where to put a newly evicted item in the larger memory.

Disks

- One or more metal platters
- 5400/7200/10800 rpm
- Track
- Cylinder
- Sector
- Virtual memory

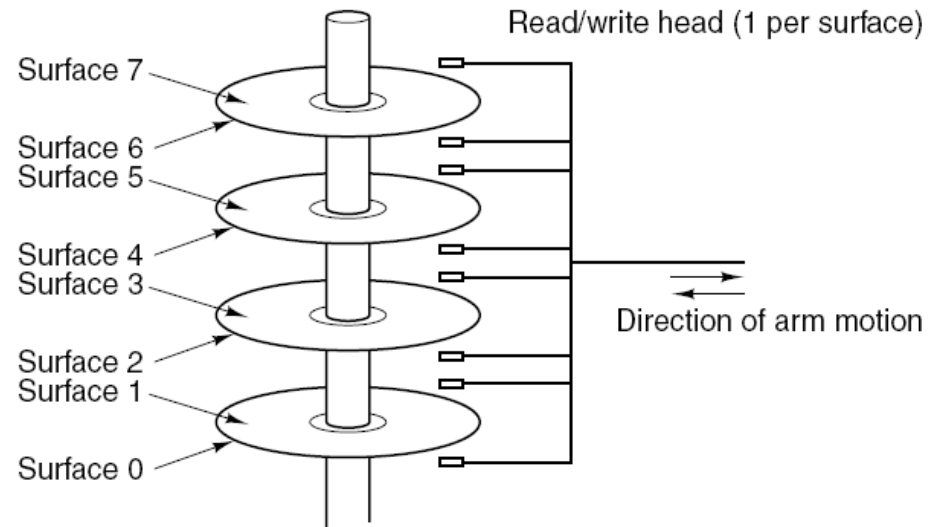


Figure 1-10. Structure of a disk drive.



I/O Devices (1)

■ Controller

- Example: Disk Controller
- Controllers are complex converting OS request into device parameters.
- Controllers often contain small embedded computers.

■ Device

- Fairly simple interfaces and standardized.
- SATA (Serial AT Attachment) – standard disk type on many computers.

I/O Devices (2)

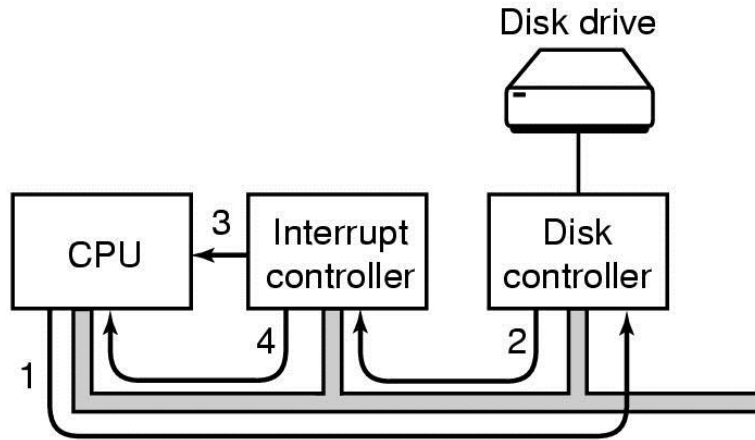
■ Device Driver

- Software that talks to a controller, giving it commands and accepting responses.
- Needed since each type of controller may be different.
- It has to be put into the OS so it can run in kernel mode.
- Each controller manufacturer supplies a driver for each OS it supports (e.g., drivers for Windows XP, Longhorn, UNIX).

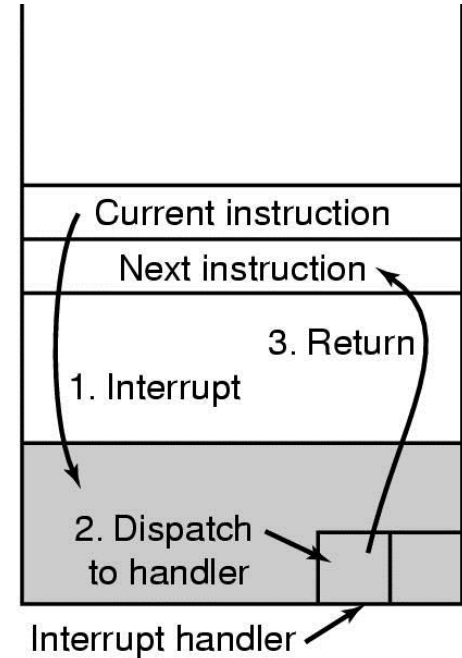
Methods for I/O (1)

- How device driver talks to controller.
 - Busy waiting
 - CPU executes a program that communicates with controllers.
 - Read/write commands
 - Sensing status
 - Transferring data
 - CPU waits for controllers to complete operation.
 - Disadvantage is that it wastes CPU time.
 - Interrupt
 - DMA

Methods for I/O (2)



(a)



(b)

Figure 1-11.

- (a) The steps in starting an I/O device and getting interrupt.
- (b) Interrupt processing involves taking the interrupt, running the interrupt handler, and returning to the user program.

Buses

- The OS must be aware of all of buses for configuration and management.

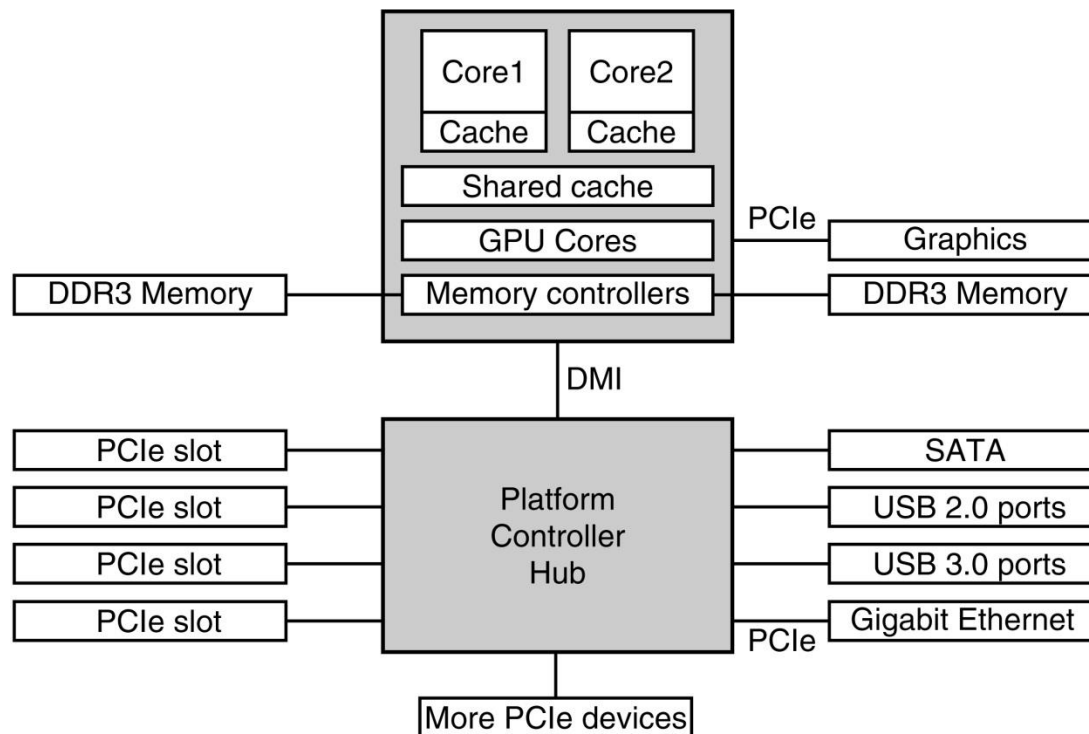



Figure 1-12. The structure of a large x86 system

- 
- 1.1 What is an Operating System?
 - 1.2 History of Operating Systems
 - 1.3 Computer Hardware Review
 - **1.4 The Operating System Zoo**
 - 1.5 Operating System Concepts
 - 1.6 System Calls
 - 1.7 Operating System Structure
 - Summary



The Operating System Zoo (1)

■ Mainframe Operating Systems

- Very powerful systems used on large computers (e.g. in insurance companies, banks)
- Support batch processing, transaction processing, and timesharing
- OS/390, Linux

■ Server Operating Systems

- Large PC's that provide services (usually over a network), e.g. print service, file service or Web service
- Solaris, FreeBSD, Linux or Windows Server 200x

The Operating System Zoo (2)

- Multiprocessor Operating Systems
 - Special versions of server OS for multiprocessor hardware.
 - HYDRA and StarOS developed by CMU.
 - Object-oriented Operating System
 - Unix-like Operating Systems
 - DYNIX (Dynamic Unix)
 - Developed by Sequent Computer Systems.
 - Based on 4.2BSD and modified to run on Intel-based symmetric multiprocessor hardware.
 - UMAX, RP3
 - Popular OS
 - Windows
 - Linux
 - ...



The Operating System Zoo (3)

- Personal Computer Operating Systems

- Windows, Linux, FreeBSD, MacOS

- Handheld Computer Operating Systems

- Symbian OS, Palm OS, Android, iOS

- Embedded Operating Systems


- No untrusted software will run on embedded systems.

- QNX, Vxworks



The Operating System Zoo (4)

- Sensor Node Operating Systems
 - TinyOS
- Real-Time Operating Systems
 - Special OS that operates in real time
 - Response of hardware must meet strict time requirements
 - e-Cos
- Smart Card Operating Systems
 - The OS is the hardware-specific firmware that provides basic functionality as secure access to on-card storage, authentication and encryption.
 - Only a few cards allow writing programs that are loaded onto the smart card.

- 
- 1.1 What is an Operating System?
 - 1.2 History of Operating Systems
 - 1.3 Computer Hardware Review
 - 1.4 The Operating System Zoo
 - 1.5 Operating System Concepts
 - 1.6 System Calls
 - 1.7 Operating System Structure
 - Summary



OS Concepts

- Every modern operating systems provides support for:
 - Process Management
 - Memory Management
 - Management of Persistent Storage (File)
 - Communicating with peripherals (Input/Output)
 - Protection
 - User Interface (Shell)

Processes (1)

- Process

- A process is a program in execution.
 - A process is a running instance of a program.

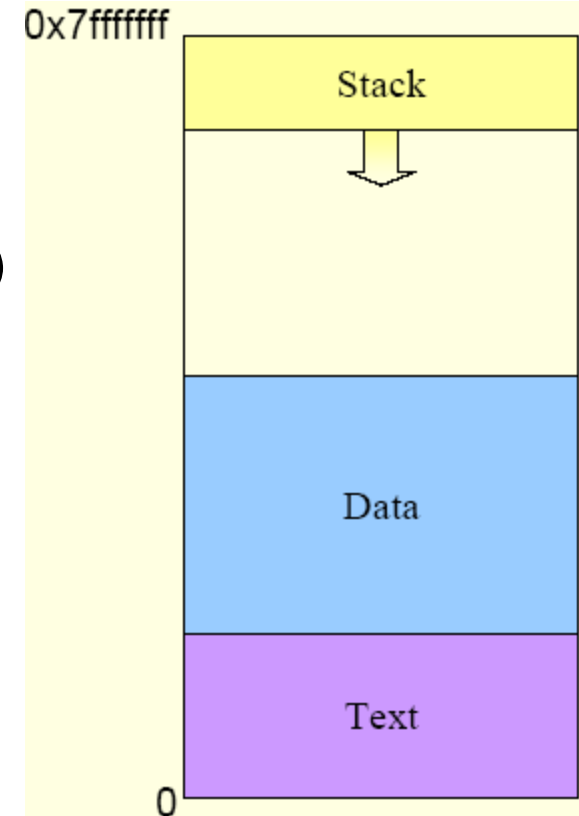
- Associated with each process are:

- The process's address space (instructions, data, stack)
 - The values stored in a set of registers including program counter, stack pointer, and other hardware registers.
 - Other information needed to run the program.

- Each person authorized to use a system is assigned a UID (User IDentification). Each process is uniquely identified with a process id.

Processes (2)

- Memory Image of a Unix Process
 - Text: program code
 - Data: program data
 - Statically declared variables
 - Areas allocated by malloc() or new (heap)
 - Stack
 - Automatic variables
 - Procedure call information
 - Address space growth
 - Text: doesn't grow
 - Data: grows “up”
 - Stack: grows “down”



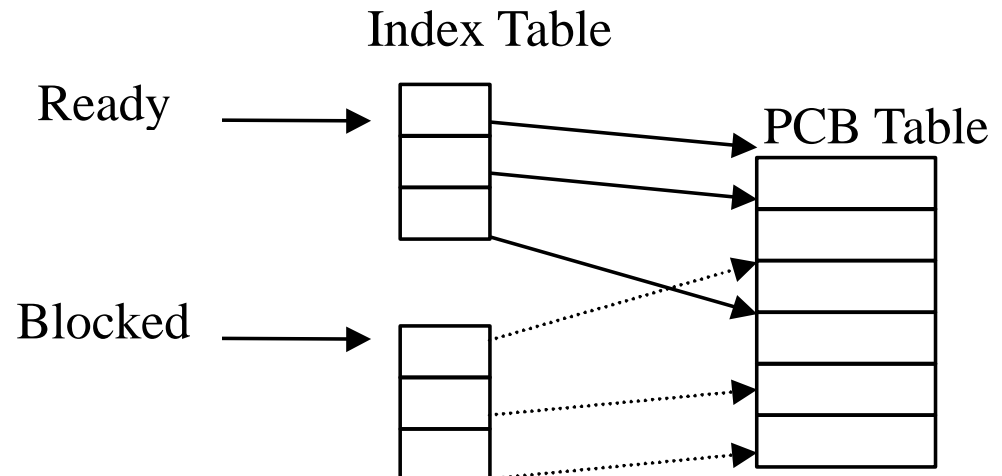
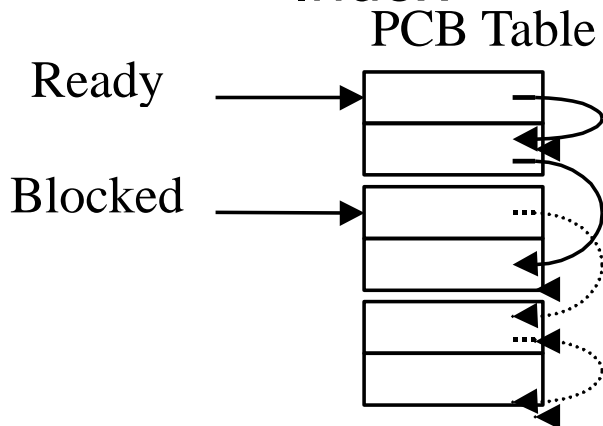
Processes (3)

■ Process Table or PCB Table

- Process Control Block is a data structure in the operating system kernel containing the information needed to manage a particular process.
- OS maintains a PCB for each process.
- OS keeps track of all processes in a *process table*.
- Two Organization Form

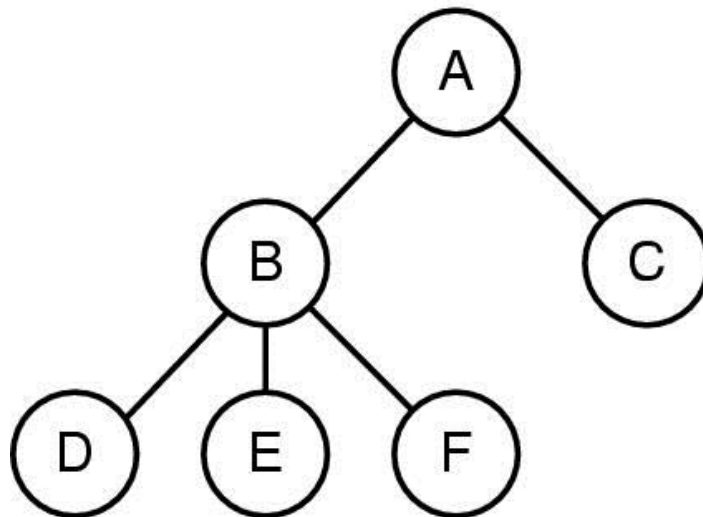
■ Link

■ Index



Processes (4)

- Processes can create other processes
 - A process tree
 - A created two child processes, B and C
 - B created three child processes, D, E, and F



Processes (5)

- Process Management Activities

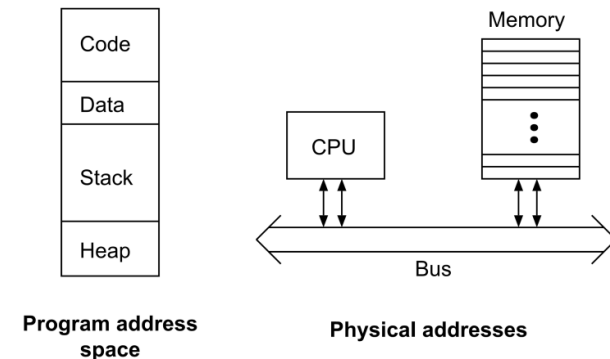
- ☐ Creating and deleting both user and system processes.
- ☐ Suspending and resuming processes.
- ☐ Providing mechanisms for process synchronization.
- ☐ Providing mechanisms for process communication.
- ☐ Providing mechanisms for deadlock handling.

Memory Management (1)

- Memory management determines what is in memory and when.
 - Optimizing CPU utilization and computer response to users.
- Managing and Protecting of Main Memory
 - Keeping track of which parts of memory are currently being used and by whom.
 - Deciding which processes (or parts thereof) and data to move into and out of memory.
 - Allocating and deallocating memory space as needed.

Memory Management (2)

- Managing Process Address Space
 - Virtual Memory
 - The address space is how the process sees its own memory.
 - Not necessarily how the memory is laid out in physical RAM.
 - Every process has its own **separate** address space.
 - Memory addresses are “private” to each process.
 - If Process A accesses address 0x3000, and Process B accesses 0x3000, they are accessing **different** physical memory locations.
 - *Unless, of course, the processes have been set up to share memory somehow.*



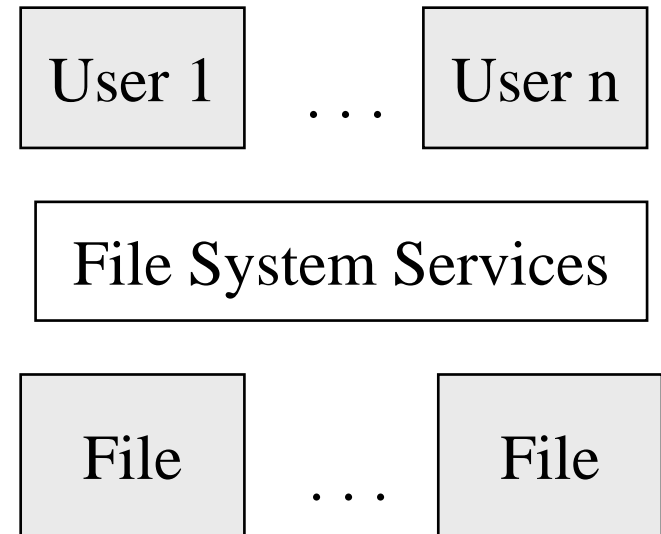


File Management (1)

- Files, Directories and File Systems
 - Secondary storage provides persistent storage in a computer system.
 - OS implements an abstraction that is called a file to represent bytes stored on secondary storage.
 - Files are organized into directories.
 - Directories make up file systems.

File System (2)

- The OS provides services for managing files, directories, and file systems on secondary storage.
- OS activities include
 - Creating and deleting files and directories.
 - Primitives to manipulate files and directories.
 - Mapping files onto secondary storage.
 - Backup files onto stable (non-volatile) storage media.



File System (3)

■ File system for a university department

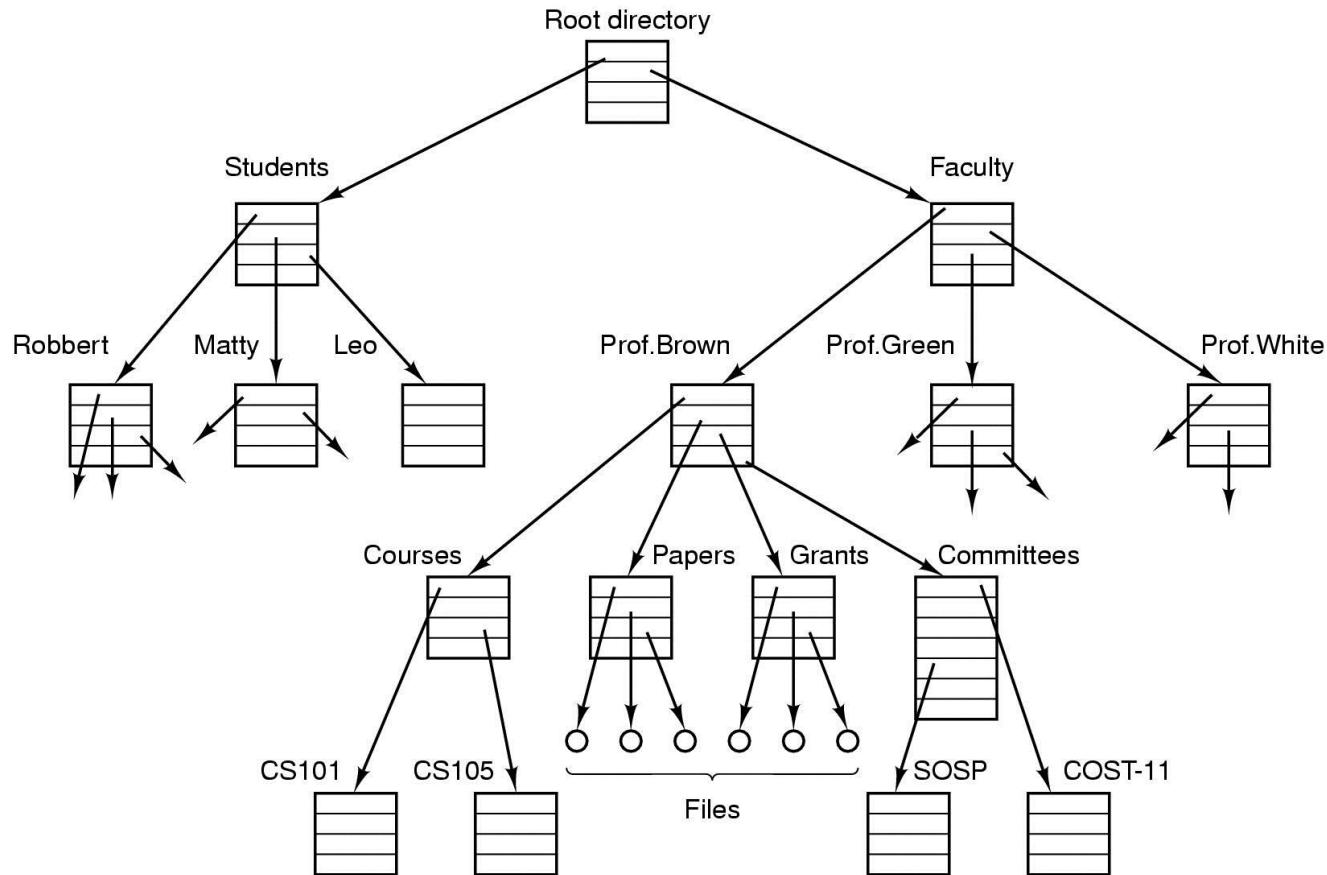
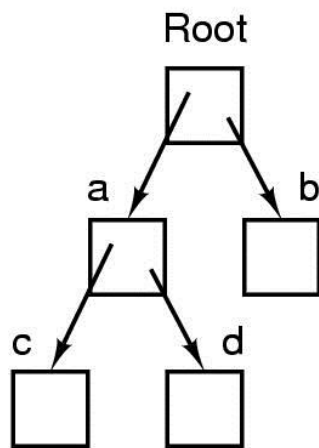


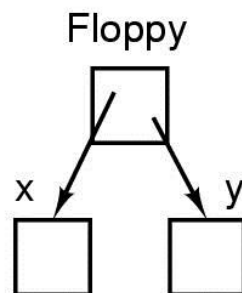
Figure 1-14. A file system for a university department.

File System (4)

- Unix file systems can be extended to include files from additional secondary storage devices.
 - Before mounting
 - Files on floppy are inaccessible.
 - After mounting floppy on b,
 - Files on floppy are part of file hierarchy.



(a)



(b)

File System (5)

■ Unix Special Files

□ Special Files (Device Files)

- Represent hardware or logical devices.
- An interface for a device driver that appears in a file system as if it was an ordinary file.
- Found in directory called /dev.

□ Block Special File (Block Device)

- Talks to devices 1 block at a time (1 block = 512 bytes to 32KB)
- Hard disk, DVD, CD ROM, ...
- e.g. /dev/hd1



File System (6)

■ Unix Special Files (ctd.)

□ Character Special File (Character Device)

- Used to model serial I/O devices such as keyboards and printers.
- Talks to devices in a character by character (1 byte at a time)
- e.g. /dev/lp

File System (7)

■ Pipe

- A pipe provides a communication channel between two independent processes.

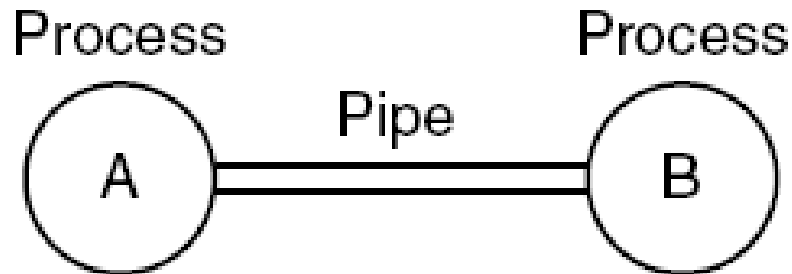


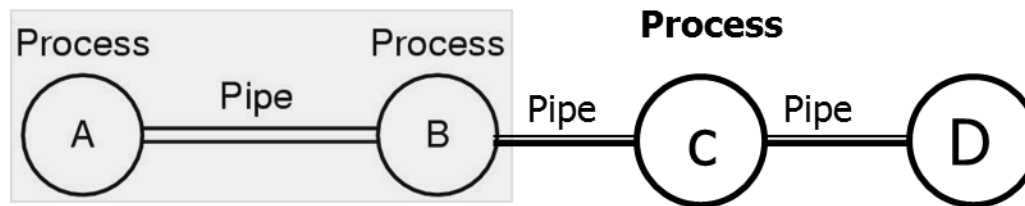
Figure 1-16. Two processes connected by a pipe.

- One process writes to the pipe, the other reads from the pipe.
- Looks exactly the same as reading from/to a file.

File System (8)

■ Pipe (ctd.)

- Sometimes useful to connect a set of processes in a pipeline.



- Process A writes to pipe AB, Process B reads from AB and writes to BC
- Process C reads from BC and writes to CD



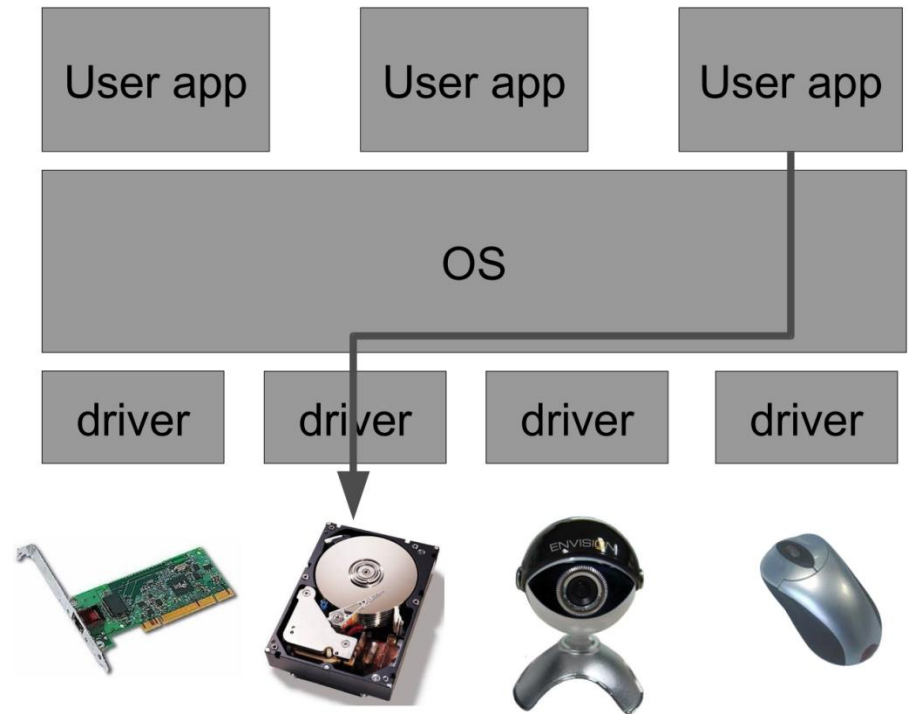
Input/Output (1)

- Every OS has an I/O sub-system.
- OS must understand how to communicate with many types of peripherals (keyboard, mouse, monitor, printer,...)
- The OS must provide an interface to I/O devices that enables programmers to perform I/O without being concerned with H/W details.
- Modern operating systems interface with the hardware through **device drivers**.

Input/Output (2)

■ Device Driver

- Black boxes to hide details of hardware devices.
- Main Role
 - Map standard calls to device-specific operations.
- Can be developed separately from the rest of the kernel.
 - Plugged in at runtime when needed.



Protection (1)

- OS must prevent unauthorized access too resources.
- Must also prevent one user from interfering with another user's resources (e.g. files)
- OS provides mechanisms for
 - Determining who can gain access to the computer system.
 - What resources a user can access.
 - What action(s) the user can carry out on resource.

Protection (2)

■ Protection Example

- Files in UNIX are protected by a 9-bit binary protection code.
 - The protection code consists of three 3-bit fields, one for owner, one for group, and one for others.
 - Each field has a bit for read, a bit for write, and a bit for execute. They are known as rwx bits.
 - e.g. `rwxr-x--x`

Kernel & Utility

■ Kernel

- Usually refers to that part of the OS that implements basic functionality and is always present in memory.
- In some cases the entire OS is created as one monolithic entity and this entire unit is called the kernel.

■ Utility

- Programs that are not part of the OS kernel, but work closely with the kernel to provide ease of use or access to system information.
- E.g., Shell

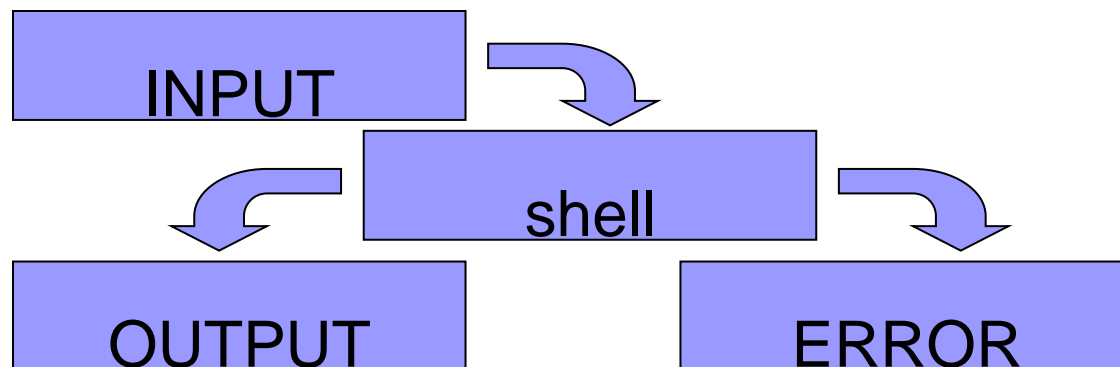
Shell (1)

■ What is Shell?

- A shell is software that provides an interface for an operating system's users to provide access to the kernel's services.

■ Uses of a Shell

- One of the major uses of a shell is to interpret commands entered at the prompt.



Shell (2)

■ Uses of a Shell (ctd.)

- Another important function of the shell is to customize the user's environment, normally done in shell initialization files.
 - Definitions for setting terminal keys and window characteristics.
 - Setting variables that define the search path, permissions, prompts, and the terminal type.
 - Setting variables that are required for specific applications such as windows, text-processing programs, and libraries for programming languages.

Shell (3)

■ Uses of a Shell (ctd.)

- The shell can also be used as an interpreted programming language.

- Shell programs, also called scripts, consist of commands listed in a file.
- Consist of Linux commands interspersed with fundamental programming constructs, such as variable assignment, conditional tests, and loops.

Shell (4)

■ Three Major Unix Shells


- Bourne Shell (or Standard Shell): sh, bash, ksh
 - Fast
 - \$ for command prompt
- C Shell
 - Shell with C-like syntax
 - Better for user customization and scripting
 - %, > for command prompt
- Korn shell: ksh
 - Superset of sh

Shell (5)

■ An Example of Bourne Shell

```

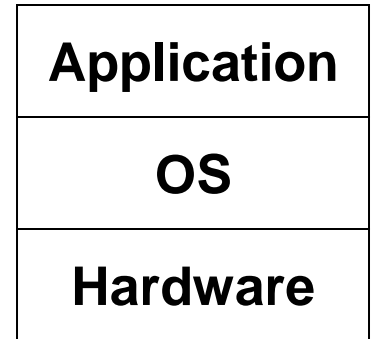
PBG-Mac-Pro:~ pbg$ w
15:24 up 56 mins, 2 users, load averages: 1.51 1.53 1.65
USER      TTY      FROM          LOGIN@  IDLE WHAT
pbg       console  -             14:34   50  -
pbg       s000    -             15:05   -  w
PBG-Mac-Pro:~ pbg$ iostat 5
            disk0            disk1            disk10            cpu            load average
      KB/t tps MB/s      KB/t tps MB/s      KB/t tps MB/s  us sy id  1m  5m  15m
      33.75 343 11.30      64.31 14  0.88      39.67  0  0.02  11  5 84  1.51 1.53 1.65
       5.27 320  1.65        0.00  0  0.00        0.00  0  0.00   4  2 94  1.39 1.51 1.65
       4.28 329  1.37        0.00  0  0.00        0.00  0  0.00   5  3 92  1.44 1.51 1.65
^C
PBG-Mac-Pro:~ pbg$ ls
Applications                Music                        WebEx
Applications (Parallels)    Pando Packages             config.log
Desktop                     Pictures                    getsmartdata.txt
Documents                   Public                      imp
Downloads                   Sites                       log
Dropbox                     Thumbs.db                  panda-dist
Library                     Virtual Machines           prob.txt
Movies                      Volumes                    scripts
PBG-Mac-Pro:~ pbg$ pwd
/Users/pbg
PBG-Mac-Pro:~ pbg$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=2.257 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.262 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.262/1.760/2.257/0.498 ms
PBG-Mac-Pro:~ pbg$ 
```

- 
- 1.1 What is an Operating System?
 - 1.2 History of Operating Systems
 - 1.3 Computer Hardware Review
 - 1.4 The Operating System Zoo
 - 1.5 Operating System Concepts
 - **1.6 System Calls**
 - 1.7 Operating System Structure
 - Summary

From Slide What is OS

■ Code that:

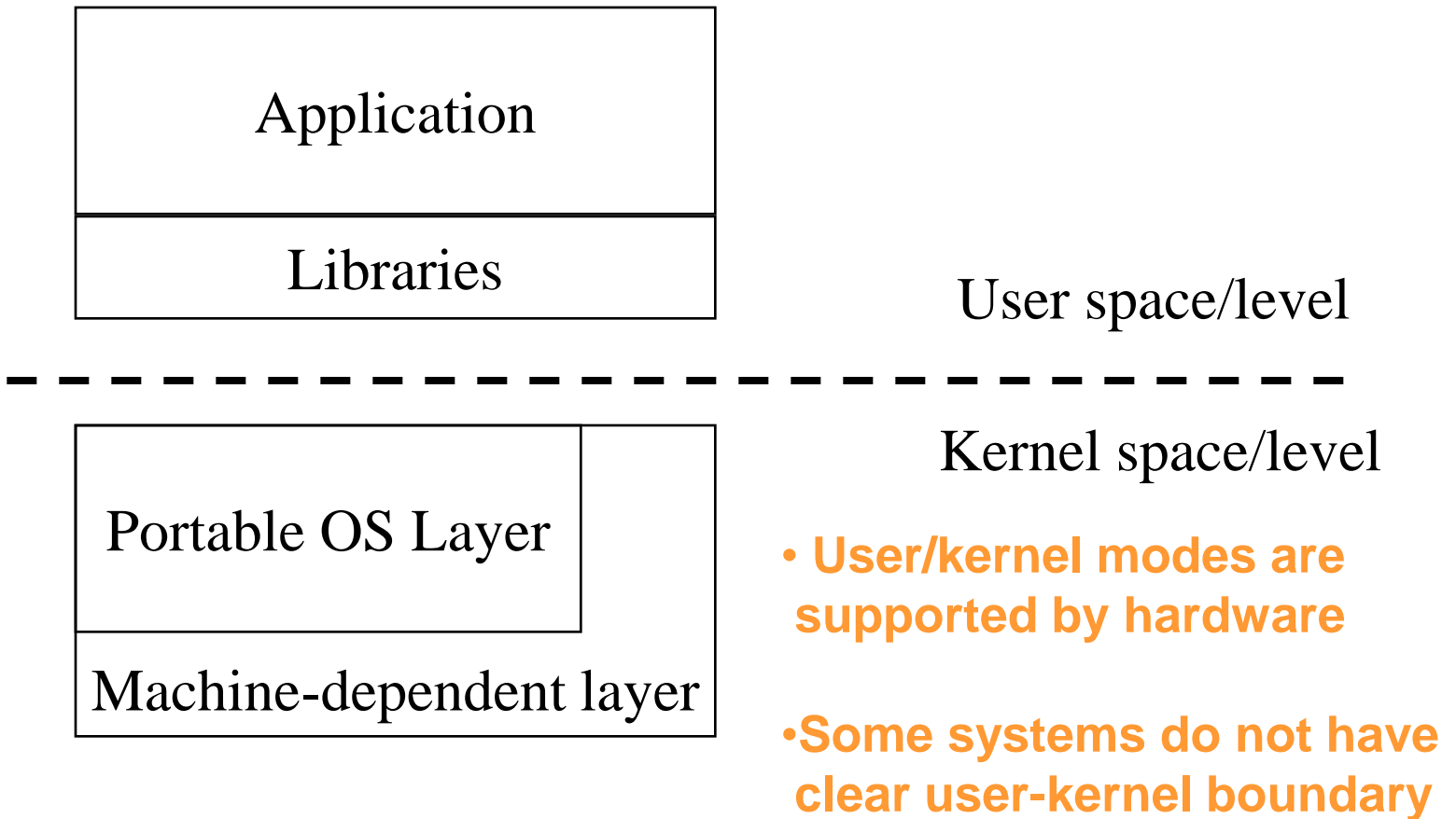
- Sits between programs & hardware
- Sits between different programs
- Sits between different users



■ Job of OS:

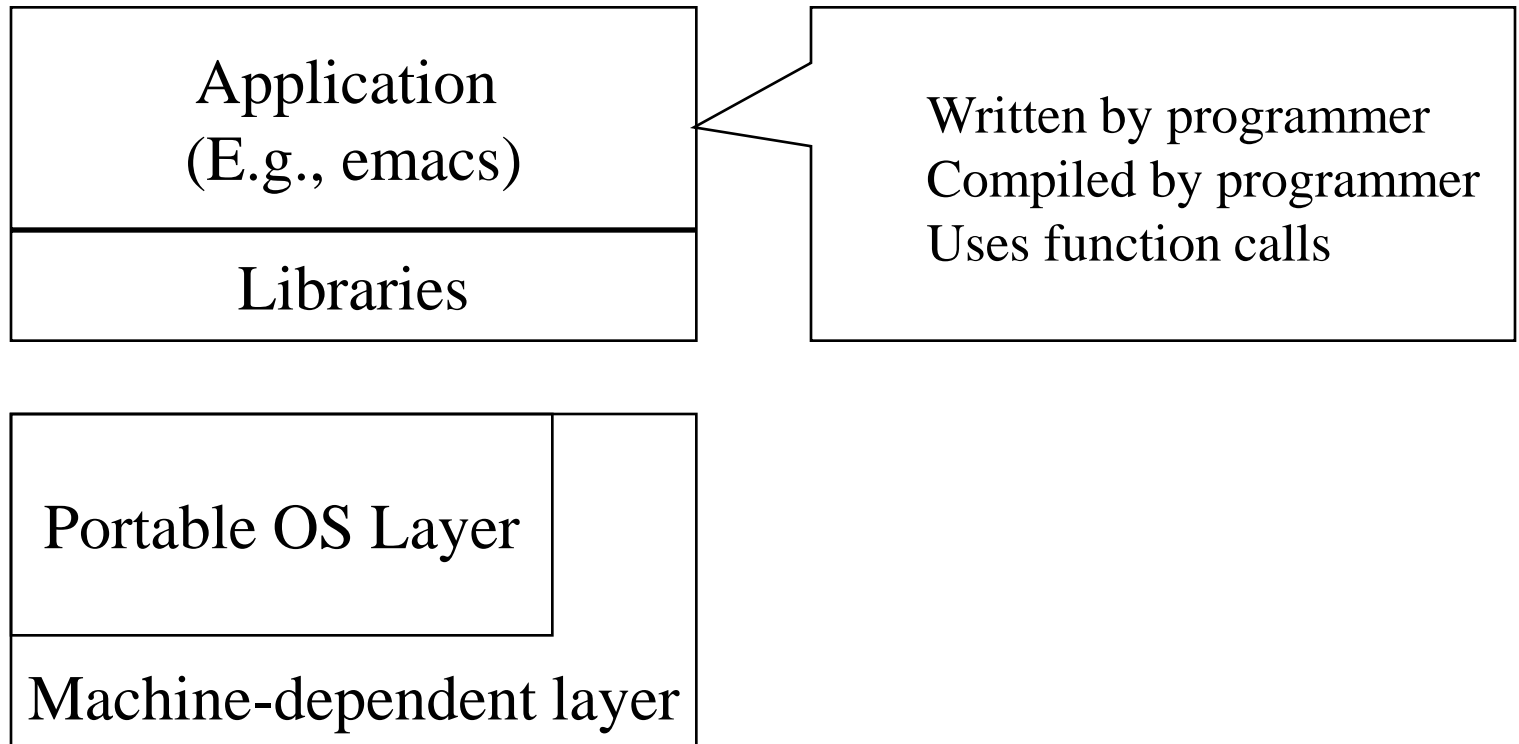
- Manage hardware resources
 - Allocation, protection, reclamation, virtualization
- Provide services to app. How? -- System Call
 - Services are functions that the OS kernel provides to users.
 - Abstraction, simplification, standardization

A Peek into Unix/Linux (1)



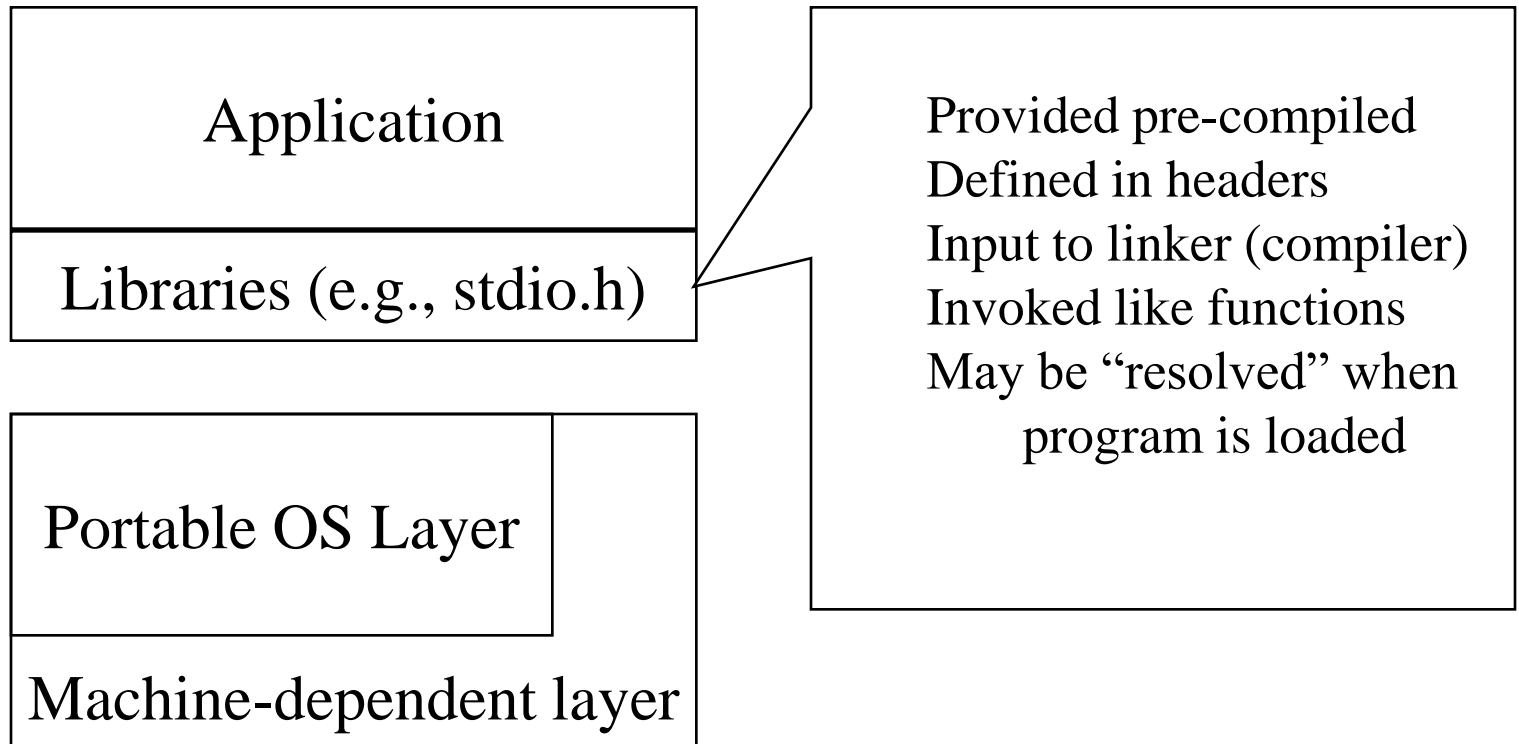
A Peek into Unix/Linux (2)

■ Unix: Application



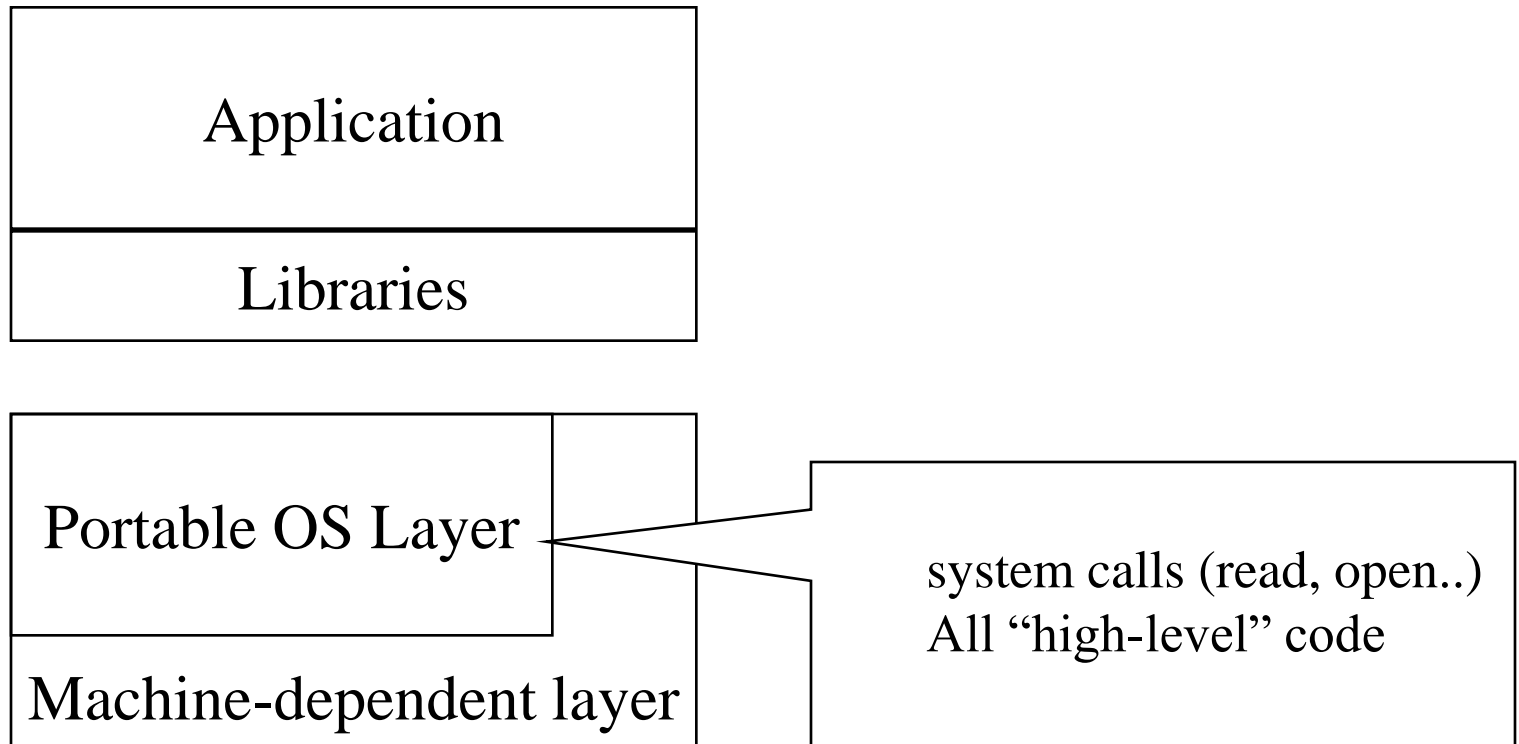
A Peek into Unix/Linux (3)

■ Unix: Libraries



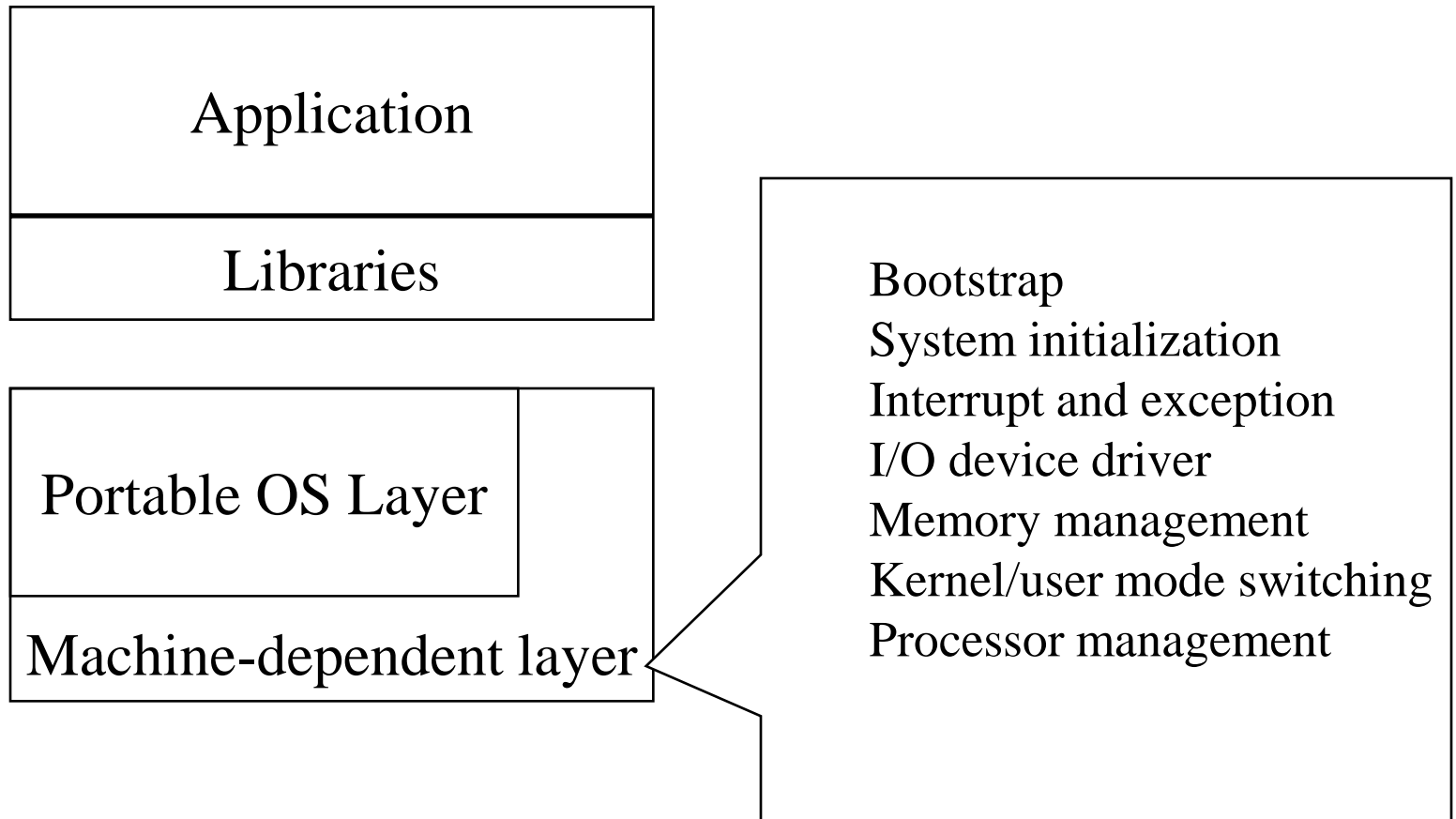
A Peek into Unix/Linux (4)

■ Typical Unix OS Structure



A Peek into Unix/Linux (5)

■ Typical Unix OS Structure (ctd.)



System Calls (1)

■ System Calls

- The mechanism used by an application program to request service from the operating system.
- Mostly accessed by programs via a high-level Application Programming Interface (API) rather than direct system call use.
- System calls often use a special machine code instruction which causes the processor to change mode (e.g. to "kernel mode" or "protected mode").
- Three most common APIs
 - **Win32 API** for Windows
 - **POSIX API** for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X)
 - **Java API** for the Java virtual machine (JVM)

System Calls (2)

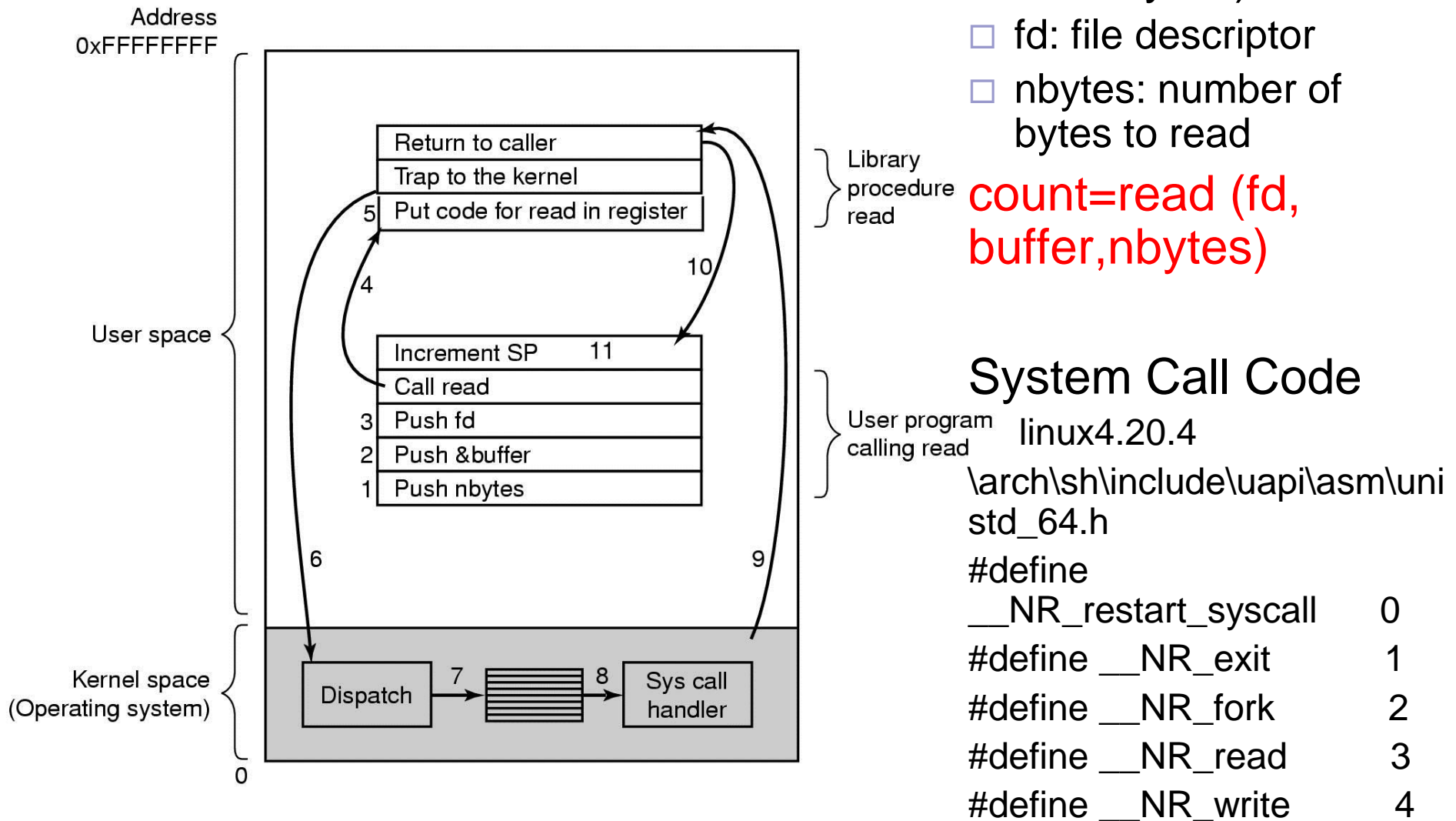


Figure 1-17. Steps in Making a System Call

System Calls (3)

■ POSIX

- Portable Operating System Interface
- Set of IEEE standards
- Mandatory + Optional Parts
- Objective: Source code portability of applications across multiple OS.

- Standard way for applications to interface to OS.
- Mostly but not exclusively Unix type OS.
- *Total portability is not achievable.*

- The mapping of POSIX procedure calls onto system calls is not one-to-one.

System Calls (4)

■ System Calls for Process Management

Process management

Call	Description
<code>pid = fork()</code>	Create a child process identical to the parent
<code>pid = waitpid(pid, &statloc, options)</code>	Wait for a child to terminate
<code>s = execve(name, argv, environp)</code>	Replace a process' core image
<code>exit(status)</code>	Terminate process execution and return status

□ `fork ()`

- `fork` is the only way to create a new process in POSIX.
- The new process created by *fork* is called the *child process*.
- This function is called once but returns twice.
 - Returns: 0 in child, process ID of child in parent, -1 on error
- Note: Both the child and parent continue executing with the instruction that follows the call to *fork*.

System Calls (5)

■ System Calls for Process Management (ctd.)

□ waitpid()

- Return: process ID if OK, 0 or -1 on error
- The interpretation of the *pid* argument depends on its value:
 - *pid* == -1 waits for any child process. In this respect, *waitpid* is equivalent to *wait*.
 - *pid* > 0 waits for the child whose process ID equals *pid*.

□ execve()

- *execve* function allows child process to execute code that is different from that of parent.
- *exec* family of functions (*execl*, *execv*, *execle*, *execve*) provides a facility for overlaying the process image of the calling process with a new image.

System Calls (6)

■ System Calls for Process Management (ctd.)

□ Example of fork used in simplified shell program

```
#define TRUE 1
while (TRUE) {                                /* repeat forever */
    type_prompt( );                            /* display prompt */
    read_command (command, parameters)        /* input from terminal */
    if (fork() != 0) {                          /* fork off child process*/
        /* Parent code */
        waitpid( -1, &status, 0);              /* wait for child to exit */
    }
    else {
        /* Child code */
        execve (command, parameters, 0);      /* execute command */
    }
}
```

■ Consider the case of a command such as

cp file1 file2

System Calls (7)

■ Some System Calls for File Management

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

System Calls (8)

■ Some System Calls for Directory and File System Management

Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

System Calls (9)

- Some System Calls for Directory and File System Management (ctd.)
 - Example: `link("/usr/jim/memo", "/usr/ast/note")`

/usr/ast		/usr/jim	
16	mail	31	bin
81	games	70	memo
40	test	59	f.c.
		38	prog1

(a)

(a) Two directories before linking
/usr/jim/memo to ast's directory

/usr/ast		/usr/jim	
16	mail	31	bin
81	games	70	memo
40	test	59	f.c.
70	note	38	prog1

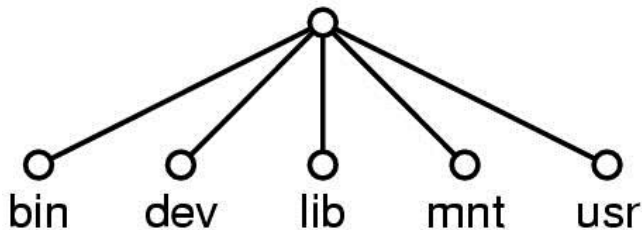
(b)

(b) The same directories after linking

System Calls (10)

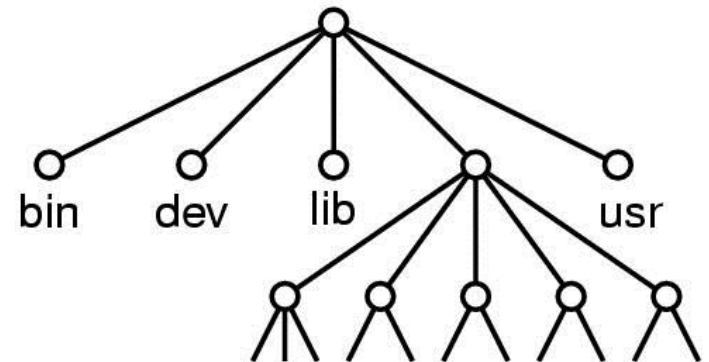
■ Some System Calls for Directory and File System Management (ctd.)

□ Example: `mount("/dev/sdb0", "/mnt", 0)`



(a)

(a) File system before the mount



(b)

(b) File system after the mount

System Calls (11)

■ Some System Calls for Miscellaneous

Miscellaneous

Call	Description
<code>s = chdir(dirname)</code>	Change the working directory
<code>s = chmod(name, mode)</code>	Change a file's protection bits
<code>s = kill(pid, signal)</code>	Send a signal to a process
<code>seconds = time(&seconds)</code>	Get the elapsed time since Jan. 1, 1970

- E.g. `chdir("/usr/ast/test");`
- E.g. `chmod("file",0644);` means to make file read only by everyone except the owner.



System Calls (12)


■ The Windows Win32 API

- In windows, the library calls and the actual system calls are highly decoupled.
- Win32 API for programmers to use to get OS services.
- The number of Win32 API calls is extremely large, numbering in the thousands.
- While many of Win32 APIs do invoke system calls, a substantial number are carried out entirely in user space.

System Calls (13)

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

Some Win32 API calls

- 
- 1.1 What is an Operating System?
 - 1.2 History of Operating Systems
 - 1.3 Computer Hardware Review
 - 1.4 The Operating System Zoo
 - 1.5 Operating System Concepts
 - 1.6 System Calls
 - 1.7 Operating System Structure
 - Summary



Operating System Structure (1)

■ Monolithic System - "the big mess"

- All operating system operations are put into a single file. The operating system is a collection of procedures, each of which can call any of the others.
- Basic Structure
 - A main program that invokes the requested service procedure.
 - A set of service procedures that carry out the system calls.
 - A set of utility procedures that help the service procedures.

Operating System Structure (2)

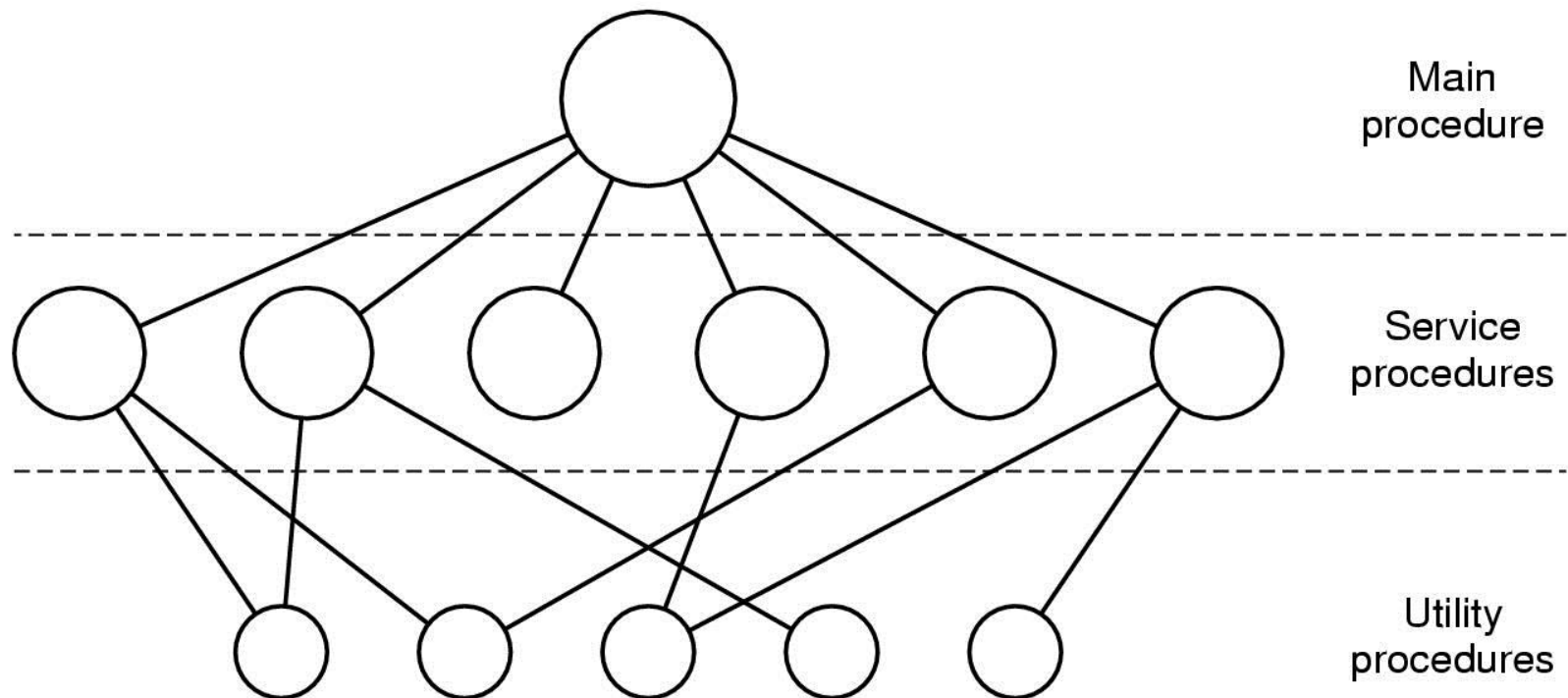


Figure 1-24. Simple structuring model for a monolithic system

Operating System Structure (3)

■ Monolithic System - "the big mess" (ctd.)

□ Examples

- BSD(Berkeley Software Distribution), FreeBSD, NetBSD, OpenBSD, MirOS BSD, SunOS.
- UNIX System V, AIX, HP-UX, Solaris, OpenSolaris / illumos.

□ Problems:

- Kernel components aren't protected from each other.
- Not easily extended/modified.
- Structure may be unclear.

Operating System Structure (4)

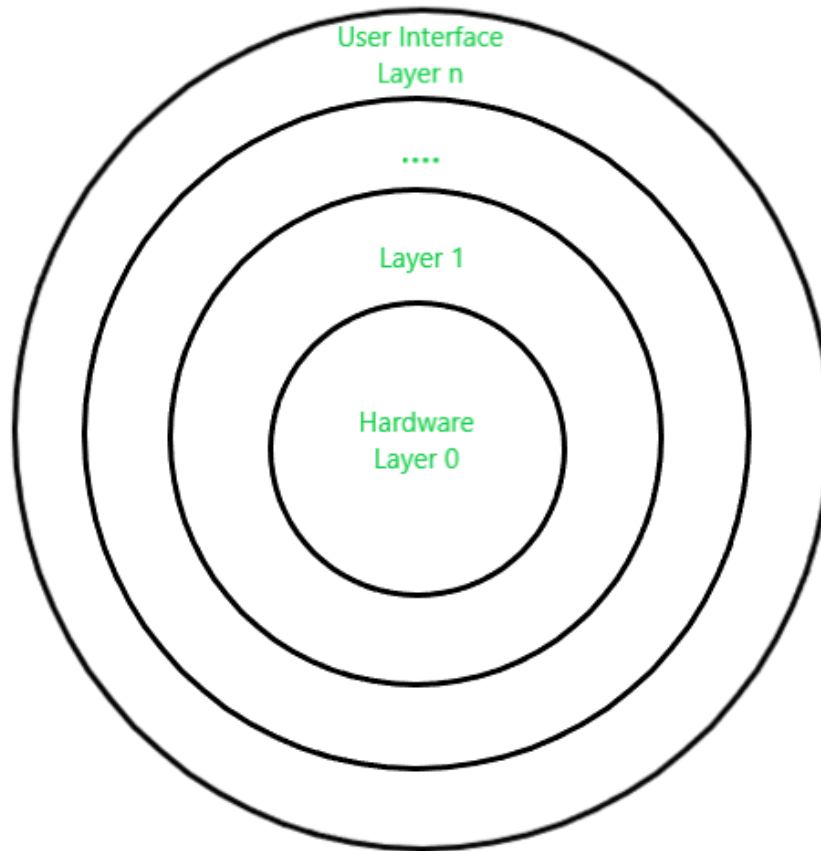
■ Layered System

- The operating system is divided into a number of layers (levels), each built on top of lower layers.
- The different services of the operating system are split into various layers, where each layer has a specific well-defined task to perform.
- The bottom layer (layer 0) is the hardware; the highest (layer N) is application with user interface.
- A particular layer can access all the layers present **below** it but it **cannot** access the layers present **above** it. That is layer $n-1$ can access all the layers from $n-2$ to 0 but it cannot access the n th layer.
- Hiding information at each layer.
- Develop a layer at a time.

Operating System Structure (5)

■ Layered System (ctd.)

□ Layered OS Design



Operating System Structure (6)

■ Layered System (ctd.)

□ THE Operating System (Dijkstra, 1968)

- A simple batch system for Electrologica X8
- All the parts of the system were ultimately linked together into a single exe.

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Figure 1-25. Structure of the THE operating system

Operating System Structure (7)

■ Layered System (ctd.)

□ Advantages

■ Modularity

- Promote modularity as each layer performs only the tasks it is scheduled to perform.

■ Easy debugging

- Debug the particular layer that has error.

■ Easy update

- A modification made in a particular layer will not affect the other layers.

■ No direct access to hardware

■ Abstraction

Operating System Structure (8)

■ Layered System (ctd.)

□ Disadvantages

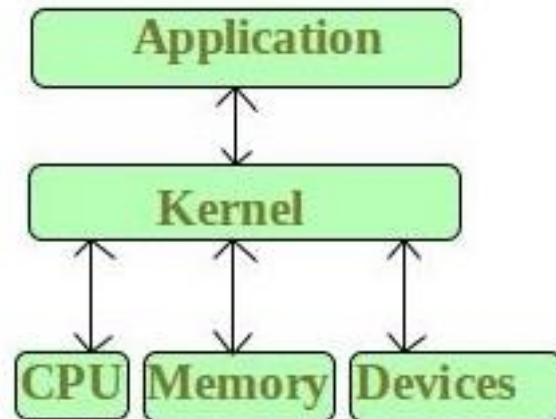
- Complex and careful implementation.
 - The arrangement of the layers must be done carefully.
 - With great modularity comes complex implementation.
- Slower in execution.

Operating System Structure (9)

■ Microkernels

□ Kernel

- The core part of an operating system which manages system resources.
- It also acts like a bridge between application and hardware of the computer.
- It is one of the first programs loaded on start-up (after the Bootloader).



Operating System Structure (10)

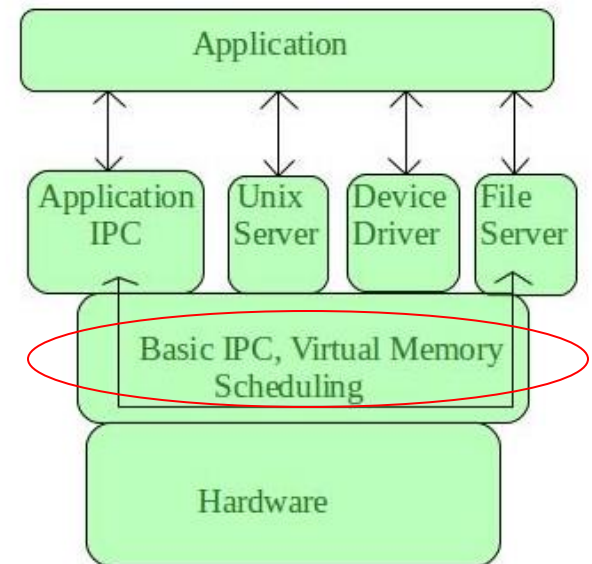
■ Microkernels (ctd.)

□ Microkernel

- Move as much functionality as possible from the kernel into “user” space.
- OS kernel is very small – minimal functionality.
 - Primitive memory management (address space)
 - I/O and interrupt management
 - Inter-Process Communication (IPC)
 - Basic scheduling
- Other OS functions provided at user level by **trusted** servers.
 - User process, trusted by kernel.
 - Device drivers, file system, virtual memory.

□ Better Known Microkernels:

- Integrity, K42, L4, MINIX 3



Operating System Structure (11)

■ Microkernels (ctd.)

□ MINIX 3

- Reincarnation Server: check if the other servers and drivers are functioning correctly. If a faulty one is detected, it is automatically replaced.

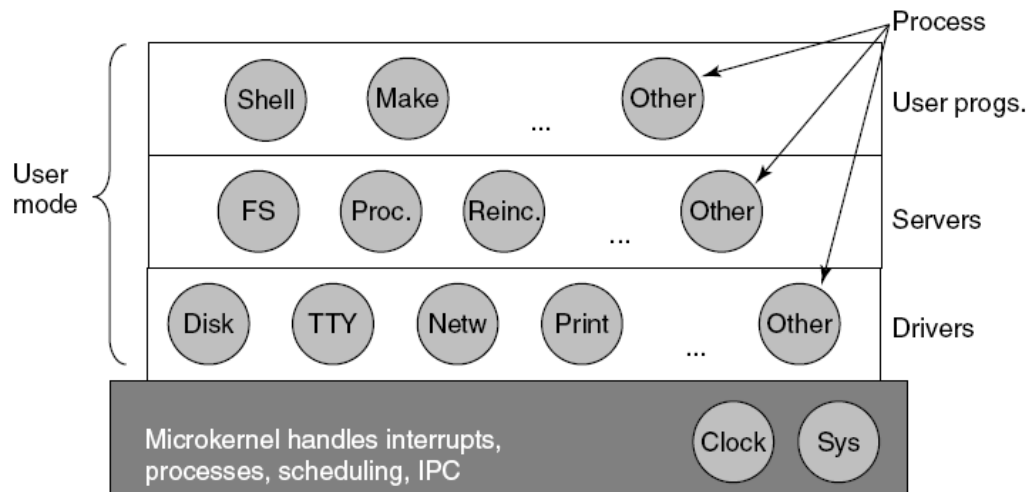


Figure 1-26. Structure of the MINIX 3 system.

- Put the mechanism for doing something in the kernel but not the policy.

Operating System Structure (12)

■ Microkernels

□ Advantages

- Expansion of the system is easier, it is simply added in the system application without disturbing the kernel.
- Reliability
- Portability

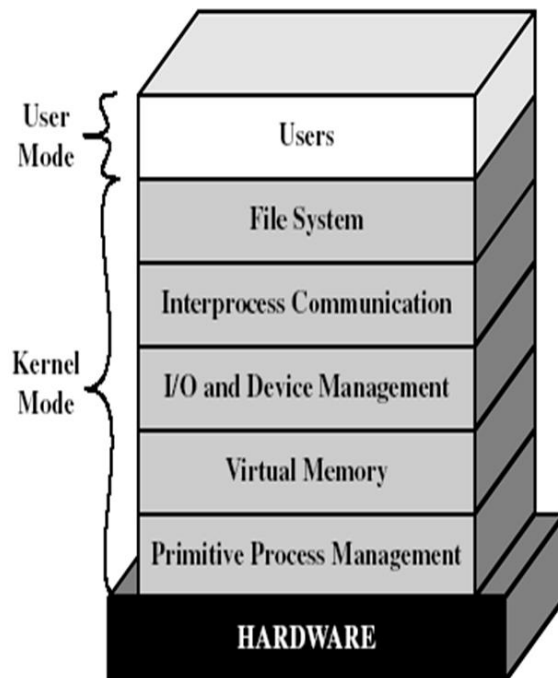
□ Disadvantage

- But performance overhead caused by replacing service calls with message exchanges between processes.

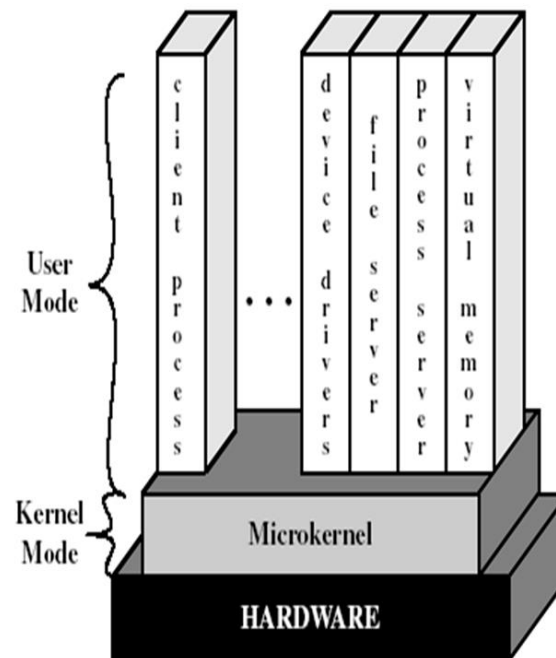
Operating System Structure (13)

■ Layered vs. Microkernels

- Communication takes place between user modules using message passing.



(a) Layered kernel

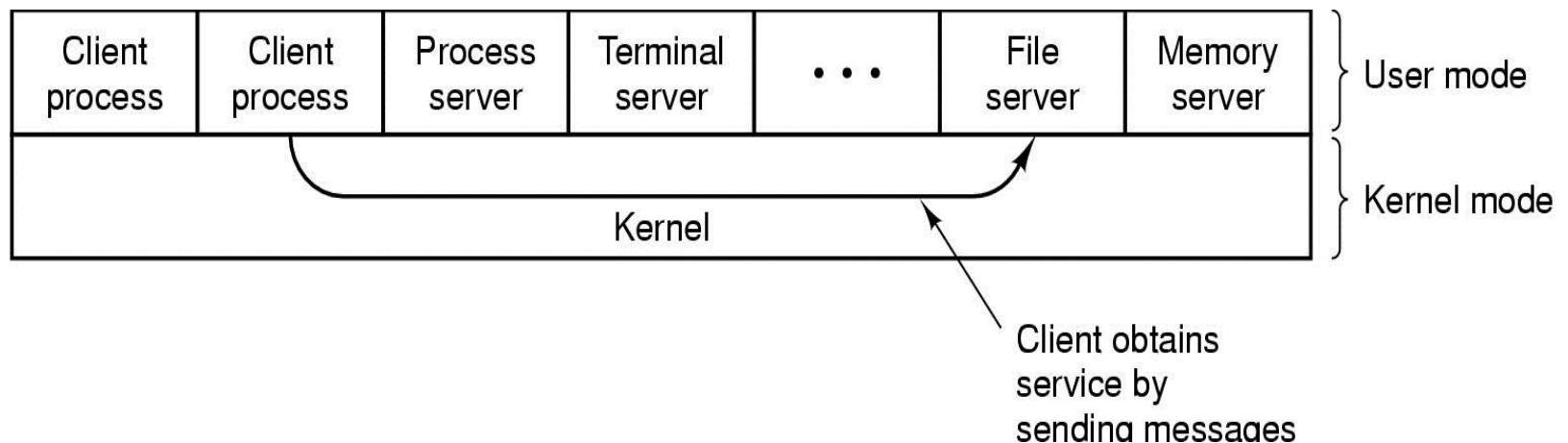


(b) Microkernel

Operating System Structure (14)

■ Client-Server Model

- A slight variation of microkernel.
- Distinguish 2 classes of processes: the servers and the clients.
- Moving most of the code up into the higher layers made the kernel to be minimal and only responsible for communication between clients and servers.



Operating System Structure (15)

■ Client-Server Model (ctd.)

- The operating system is collection of servers that provide specific services; e.g., file server, etc.

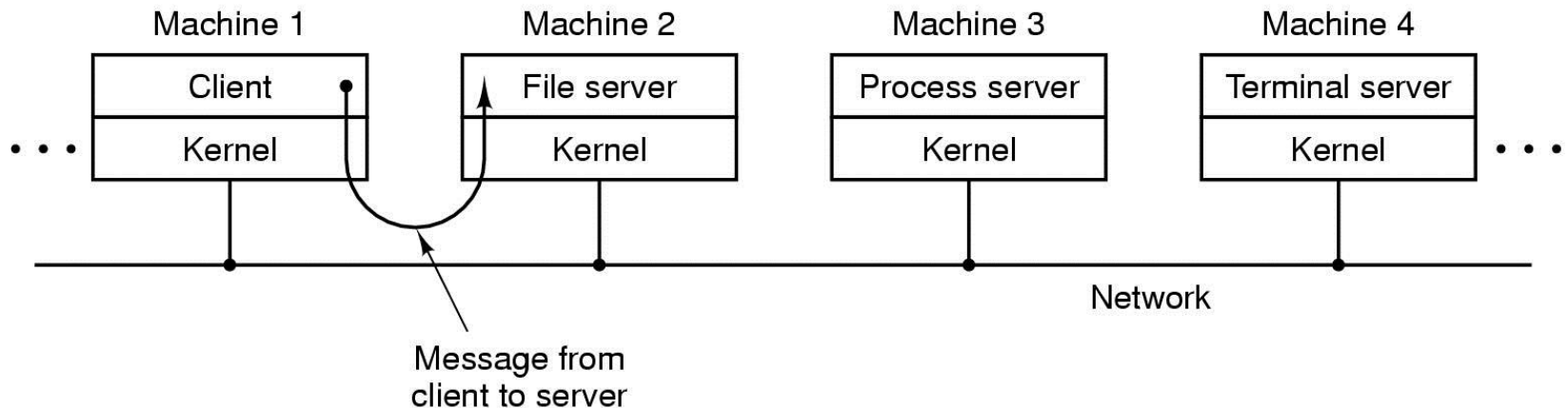


Figure 1-27. The client-server model in a distributed system

Operating System Structure (16)

■ Client-Server Model (ctd.)

□ Advantage

- Adaptability to use in distributed systems.

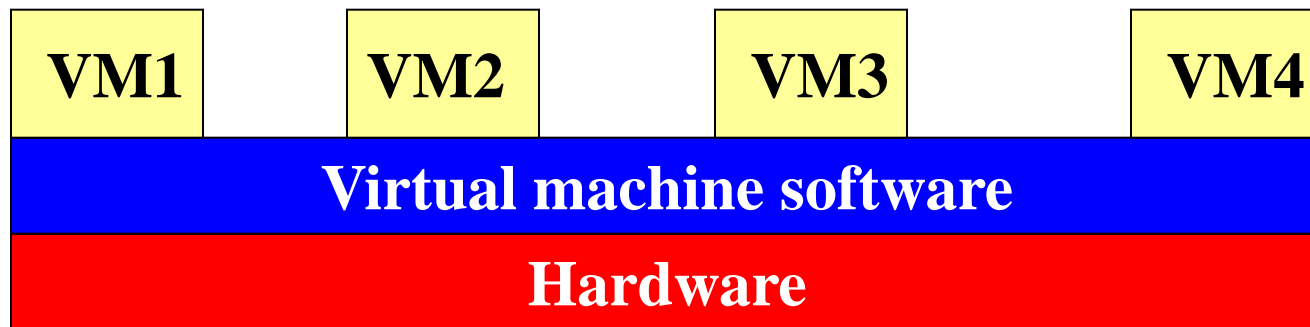
□ Disadvantage

- Doing OS functions (e.g. loading physical I/O device registers) in the user space is difficult.
- There are two ways to solve this problem
 - To have some critical server processes run in the kernel with complete access to hardware but still communicating with other normal processes.
 - Enhance the kernel to provides these tasks but server decides how to use it (Splitting policy and mechanism).

Operating System Structure (17)

■ Virtual Machines

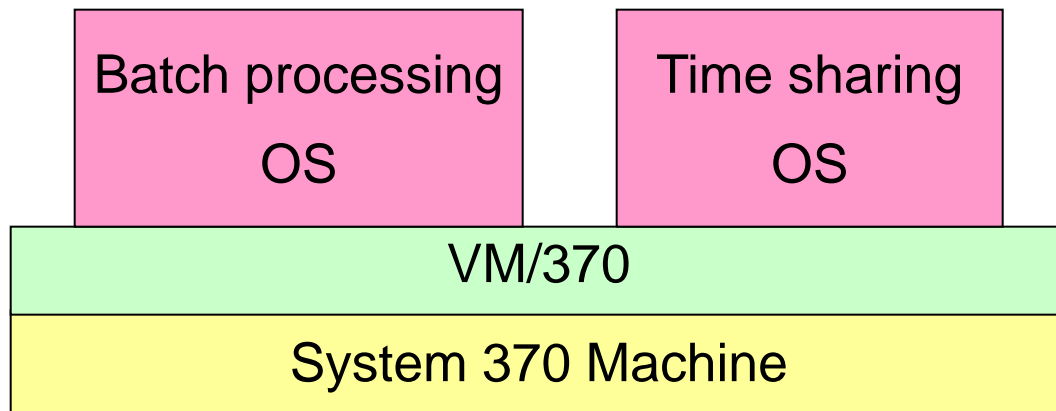
- Virtual software layer over hardware.
- Software emulation of an abstract machine.
- Illusion of multiple instances of hardware.
- Supports multiple instances of Oss.



Operating System Structure (18)

■ VM/370

- Conceived by IBM in the late 1960's
 - Popularized by VM/370 (1972)
- Used for OS debugging, time sharing, supporting multiple OS's



Structure of VM/370

Operating System Structure (19)

■ VM/370 (ctd.)

- Essence: separate multiprogramming & extended machine.
- Virtual Machine Monitor (VMM)
 - Multiprogramming, providing several virtual machines.
 - The VMM implements the complete hardware architecture in software.
- Conversational Monitor System (CMS): a single-user interactive system.

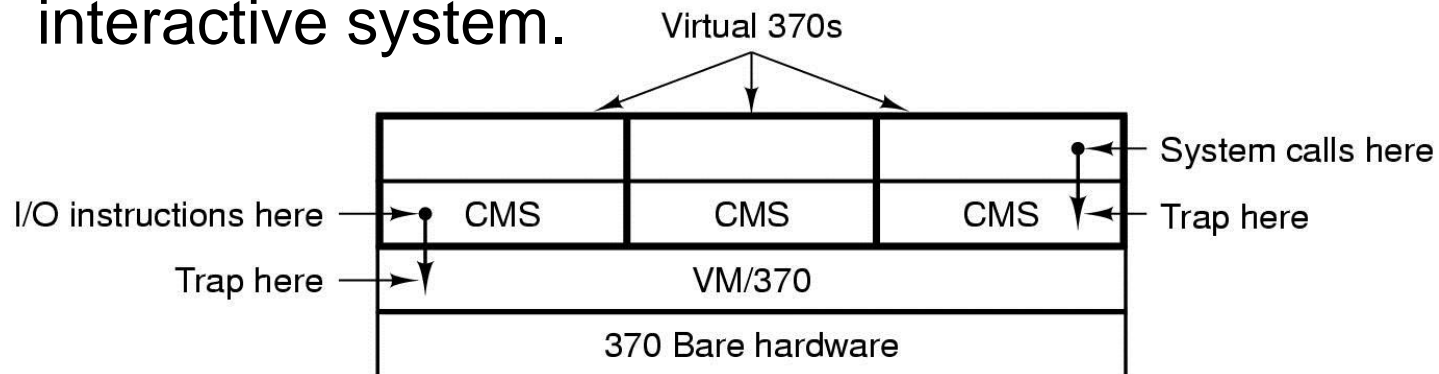


Figure 1-28 Structure of VM/370 with CMS



Operating System Structure (20)

■ Virtual Machine Rediscovered

□ Benefits of Virtual Machine

- Multiple OS environments can exist simultaneously on the same machine, isolated from each.
- Virtual machine can offer an instruction set architecture that differs from real computer's.
- Easy maintenance, application provisioning, availability and convenient recovery.

Operating System Structure (21)

■ Virtual Machine Rediscovered (ctd.)

□ Goldberg distinguished between two approaches to virtualization.

■ Type I Hypervisor

□ Runs on the raw hardware.

□ Like an operating system

□ Examples

- VMware Vsphere
- Microsoft Hyper-V
- Xen/XenServer

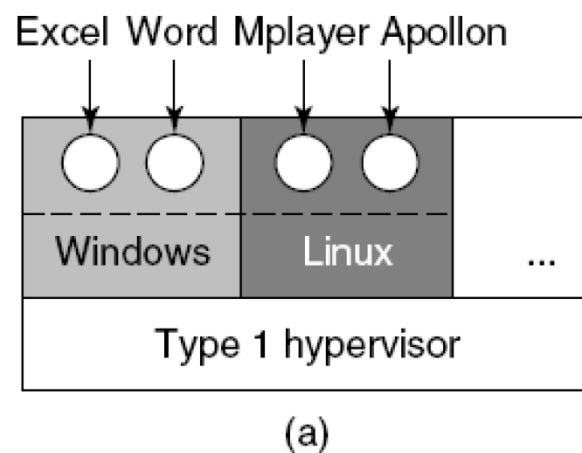


Figure 1-29. (a) A type 1 hypervisor

Operating System Structure (22)

■ Virtual Machine Rediscovered (ctd.)

□ Goldberg distinguished between two approaches to visualization. (ctd.)

■ Type II Hypervisor

□ Runs on host OS.

□ Examples

- VMware Workstation
- Oracle Virtual Box
- Microsoft Virtual PC
- Parallels
- KVM

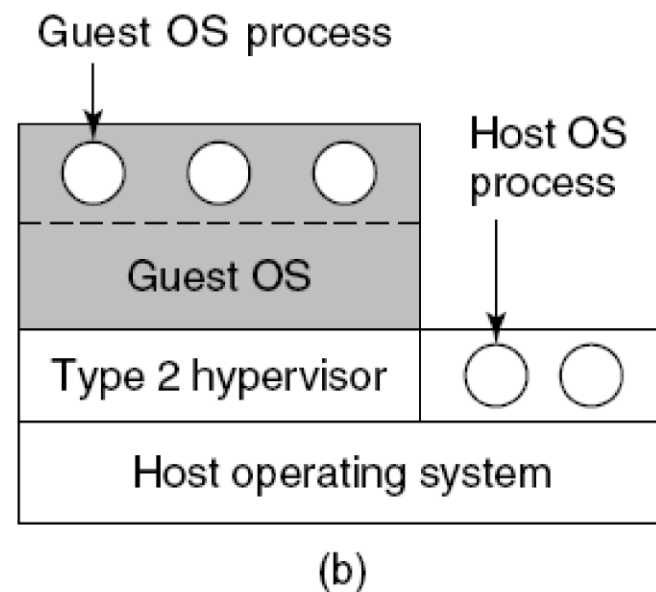
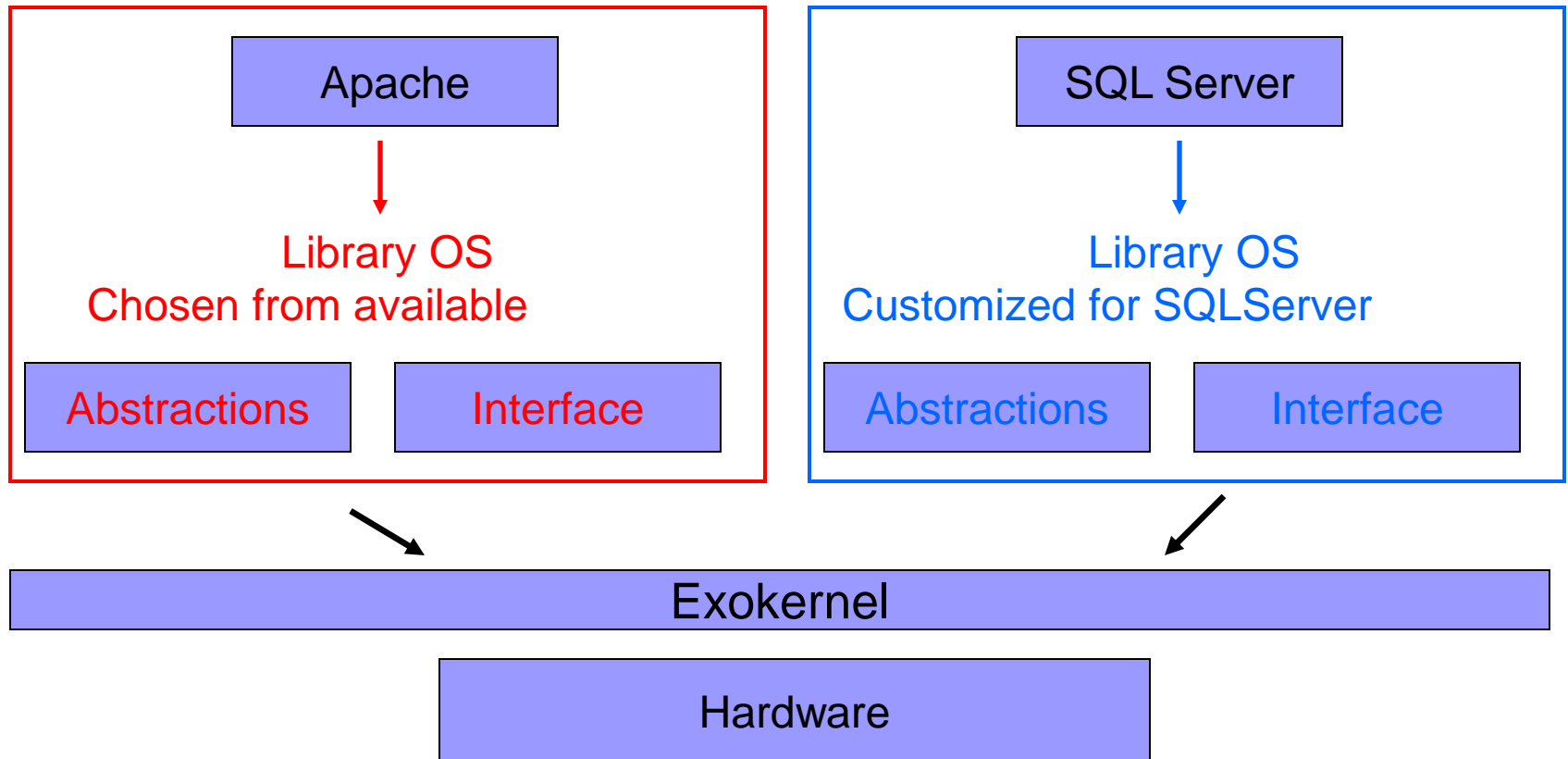


Figure 1-29. (c) A practical type 2 hypervisor.

Operating System Structure (23)

■ Exokernel

- Application level resource management





Operating System Structure (23)

- Exokernel (ctd.)

- Reference Reading

- Exokernel: An Operating System Architecture for Application-Level Resource Management

- Linux: Monolithic

- Windows NT(Windows XP Windows 7, Windows 8..), Mac OS: Hybrid Kernel(Micro & Monolithic)

Summary (1)

- What is OS
 - OS is an extended machine
 - OS is a resource manager
- OS History
 - Batch systems
 - Multiprogramming
 - Spooling
 - Time sharing
- Operating System Concepts
 - shell



Summary (2)

- System Calls
- Operating System Structure
 - Monolithic System
 - Layered System
 - Microkernel
 - Client-Server Model
 - Virtual Machines
 - Exokernel



Homework

- P81 1,3
- Reading Assignment: P713-P732 10.1, 10.2