MODERN OPERATING SYSTEMS

Chapter 1
Introduction

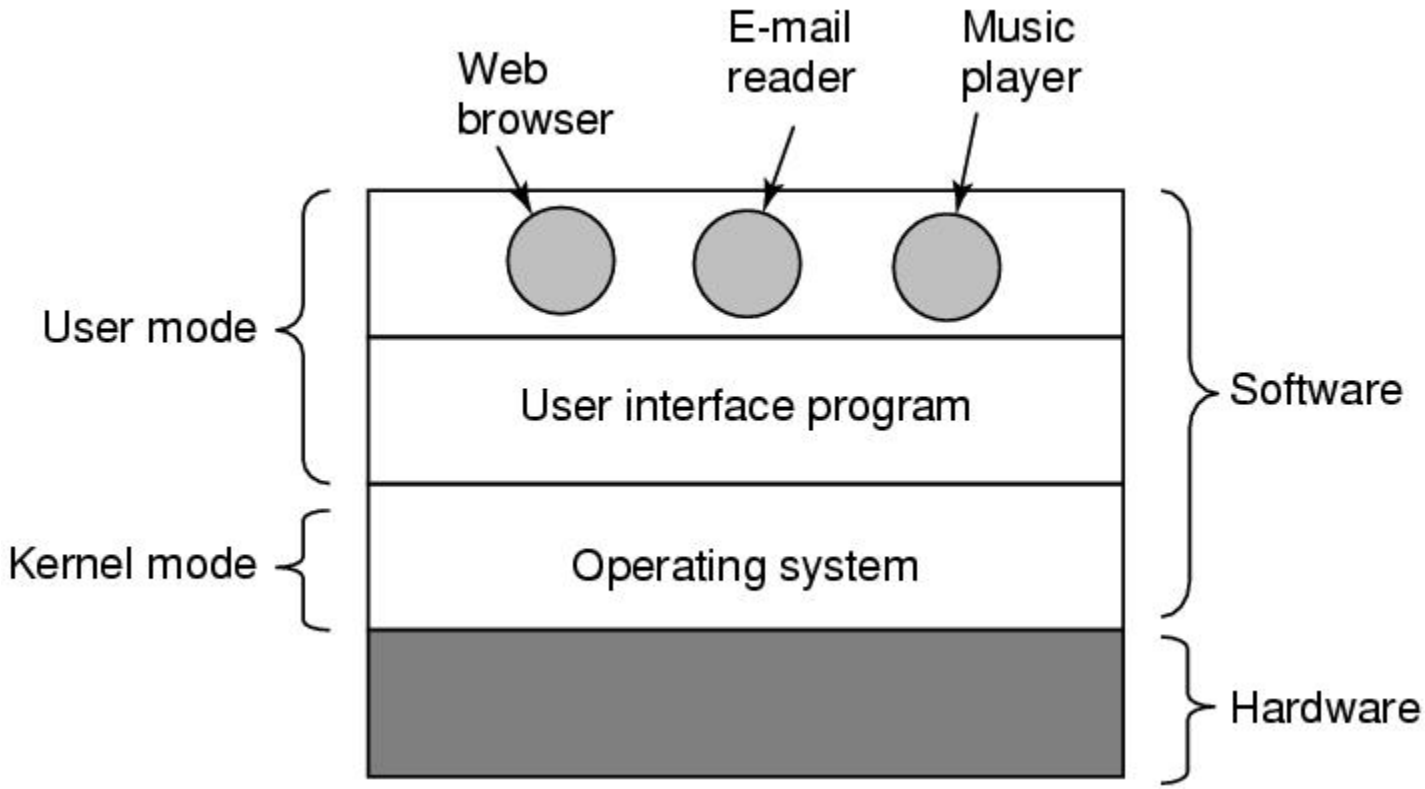
What Is An Operating System?

Lots of hardware !!

- One or more processors
- Main memory
- Disks
- Printers
- Various input/output devices

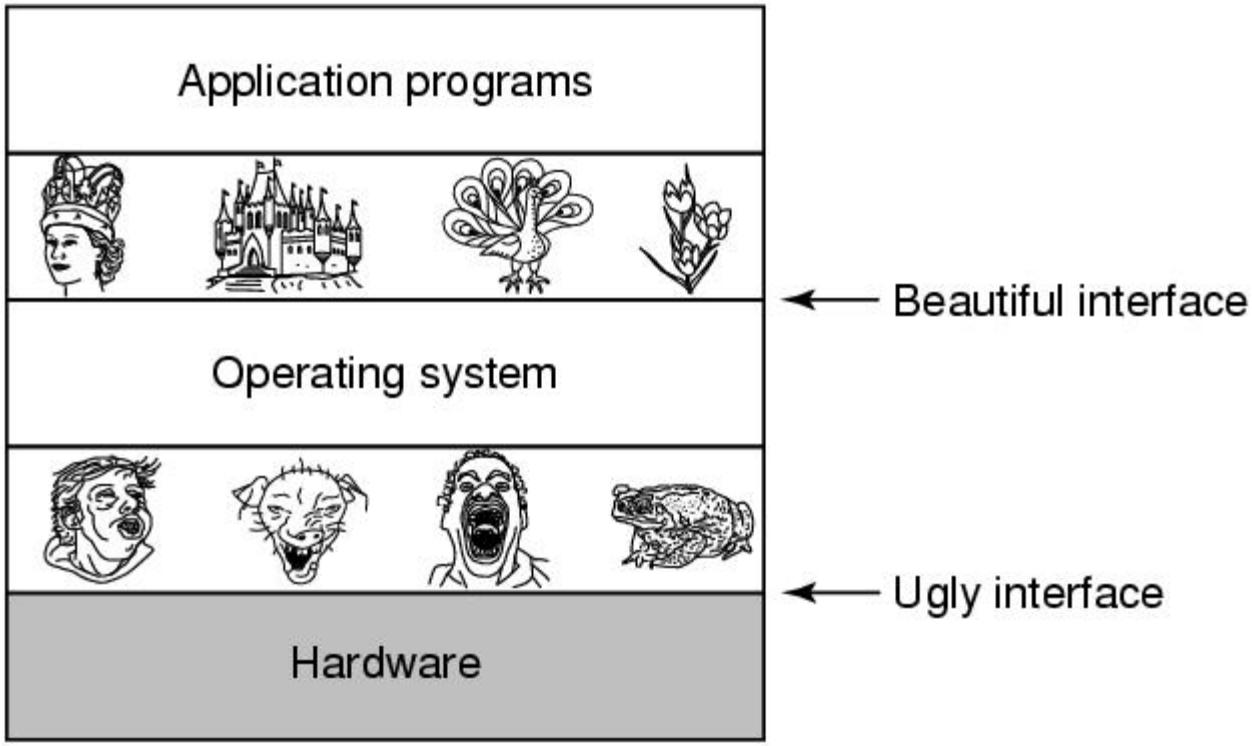
Managing all these components requires a layer of software – the **operating system**

Where is the software?



Where the operating system fits in.

The Operating System as an Extended Machine



The Operating System as a Resource Manager

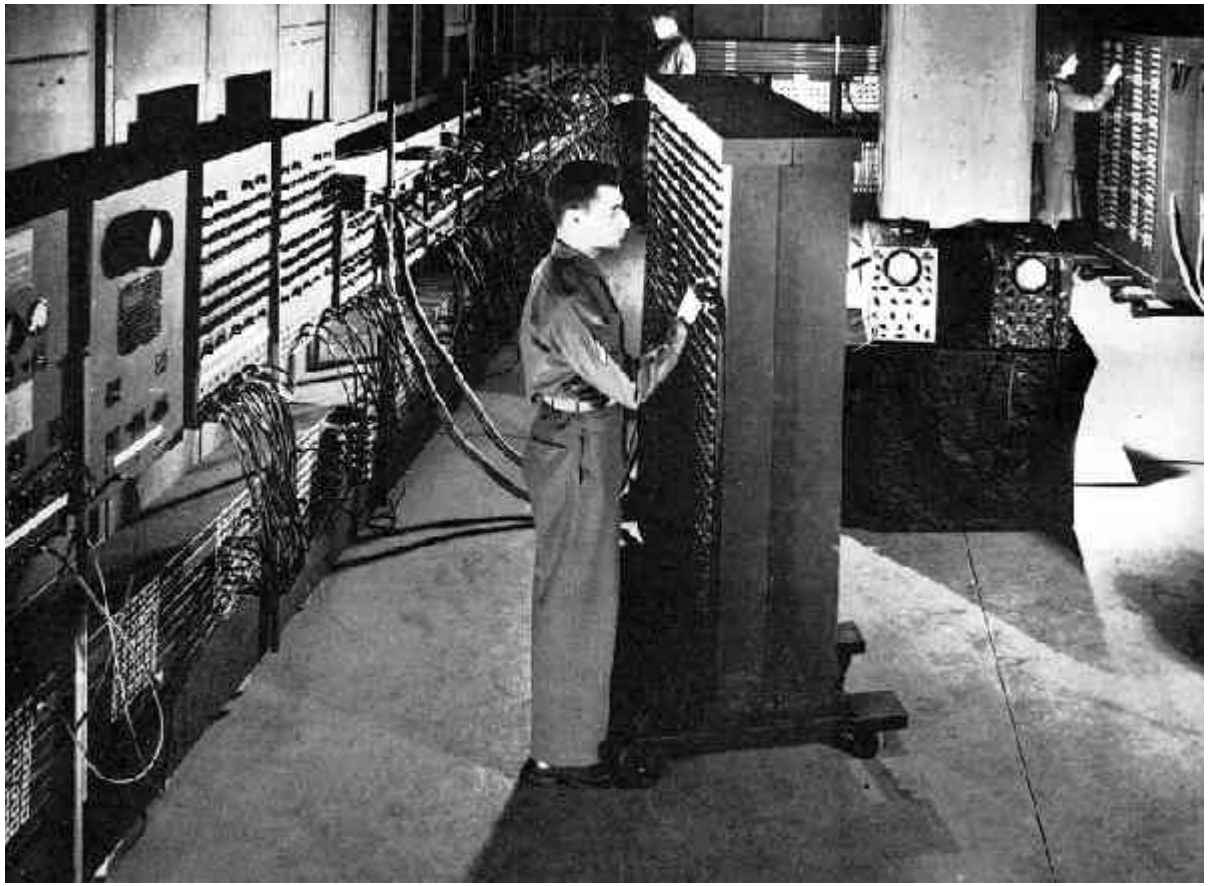
- Allow multiple programs to run at the same time
- Manage and protect memory, I/O devices, and other resources
- Multiplexes (shares) resources in two different ways:
  - In time
  - In space

History of Operating Systems

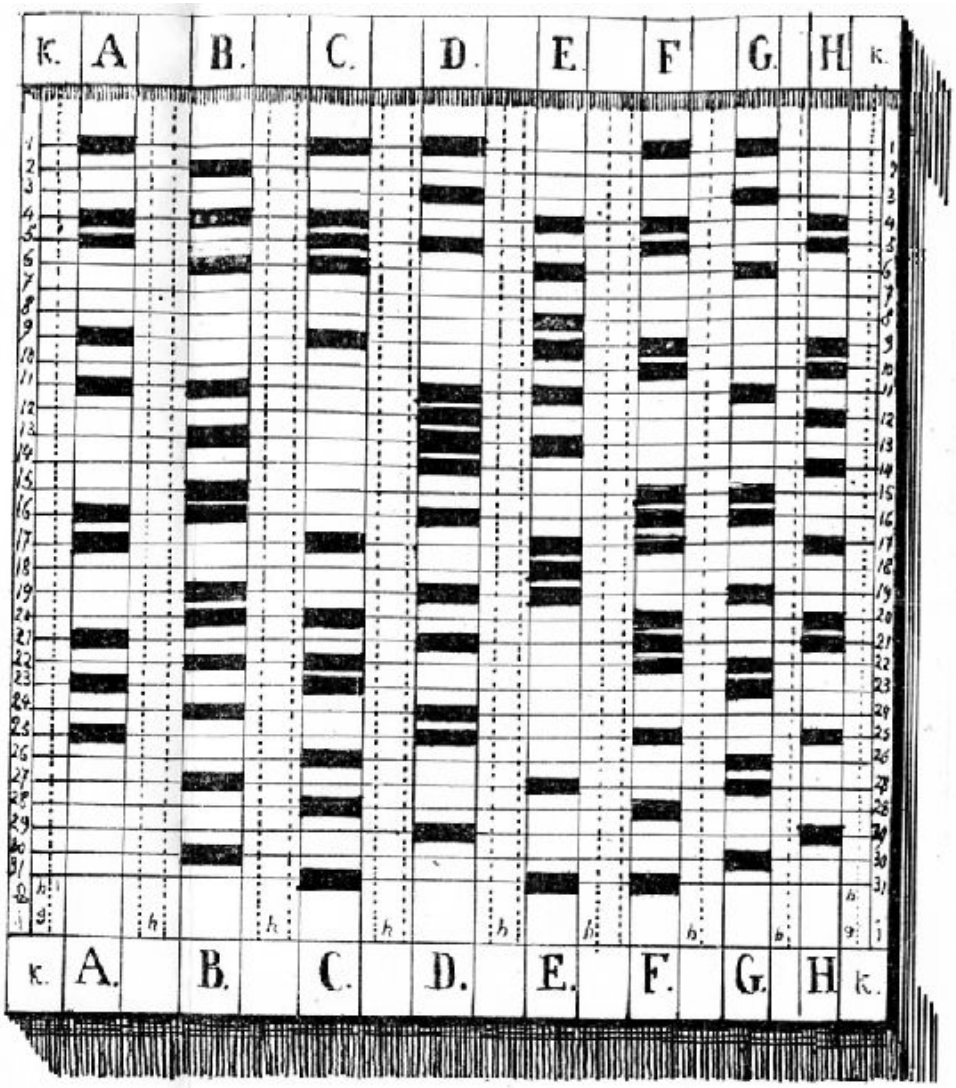
Generations:

- (1945–55) Vacuum Tubes
- (1955–65) Transistors and Batch Systems
- (1965–1980) ICs and Multiprogramming
- (1980–Present) Personal Computers, Tablets, Phones

Vacuum Tubes



Punch card



Transistors and Batch Systems (1)

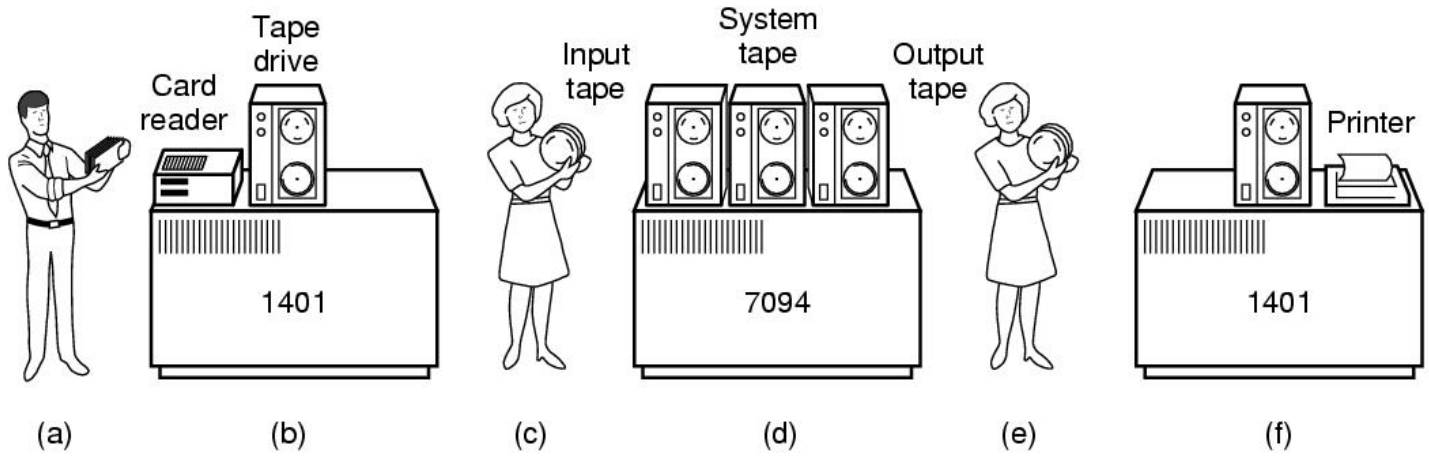


Figure 1-3. An early batch system.
(a) Programmers bring cards to 1401.
(b)1401 reads batch of jobs onto tape.

Transistors and Batch Systems (2)

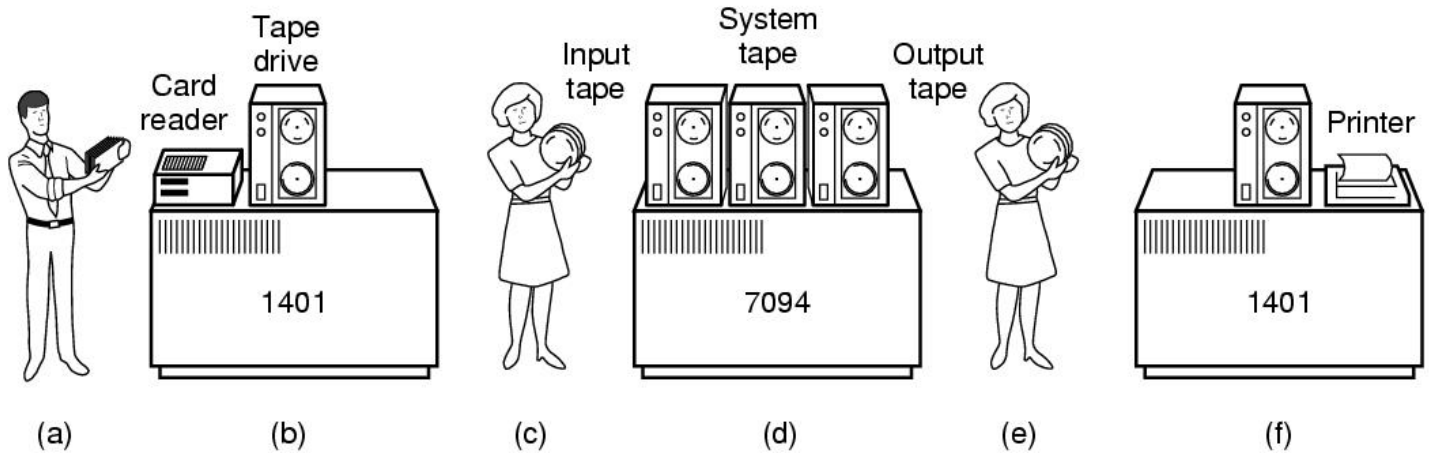


Figure 1-3. (c) Operator carries input tape to 7094. (d) 7094 does computing. (e) Operator carries output tape to 1401. (f) 1401 prints output.

Transistors and Batch Systems (4)

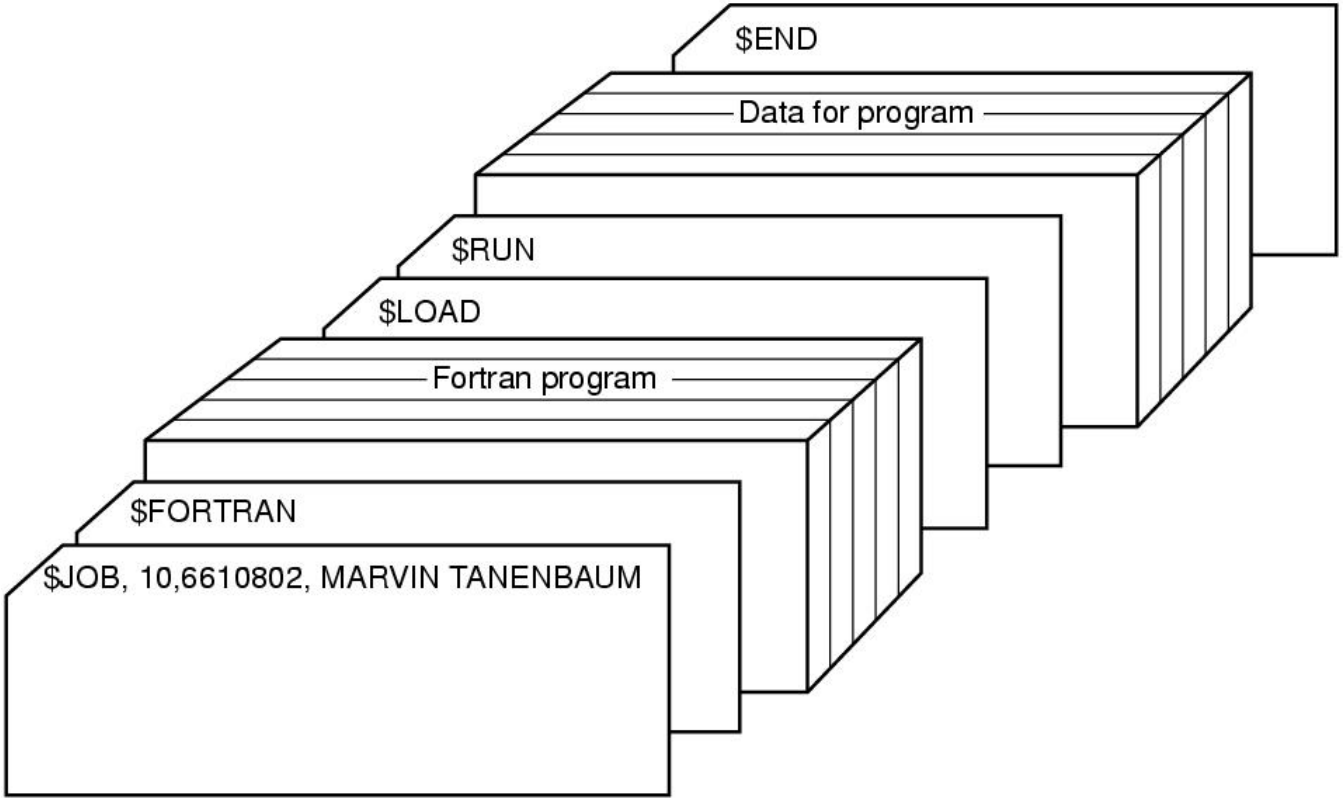


Figure 1-4. Structure of a typical FMS job.

ICs and Multiprogramming

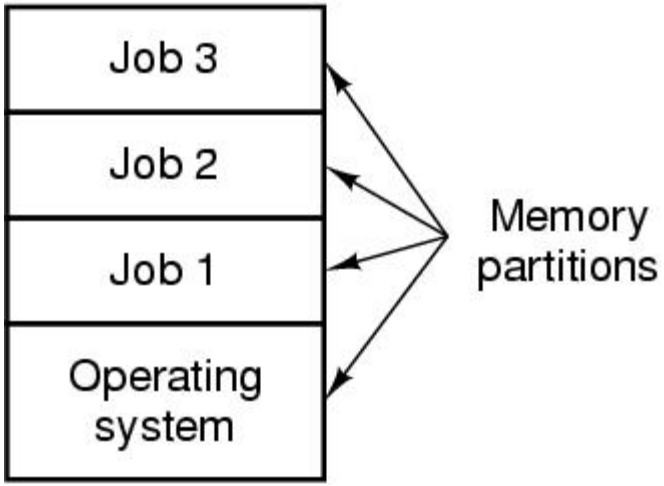


Figure 1-5. A multiprogramming system with three jobs in memory.

More third generation

- Time Sharing (Compatible Time Sharing System)
- MULTICS (MULTiplexed Information and Computing Service)
- UNIX – POSIX and MINIX
- LINUX

.

Fourth generation

- PCs
- Network Operating Systems
- Distributed Operating Systems

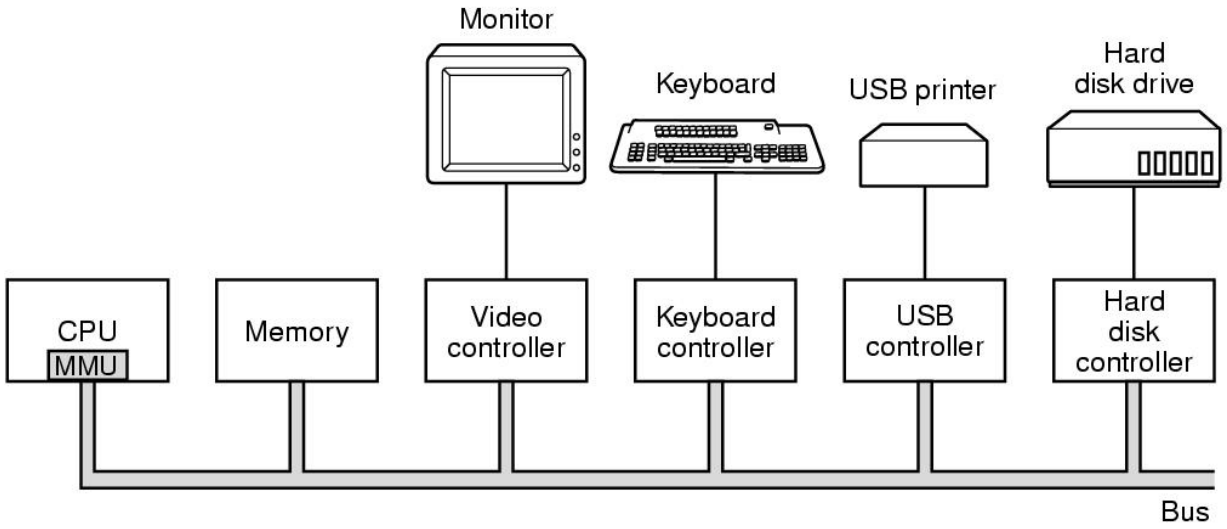
.

Fifth generation – mobile computer

- PDA (Personal Digital Assistant)
- iPhone
- Smart phone

.

Computer Hardware Review



Some of the components of a simple personal computer.

CPU Pipelining

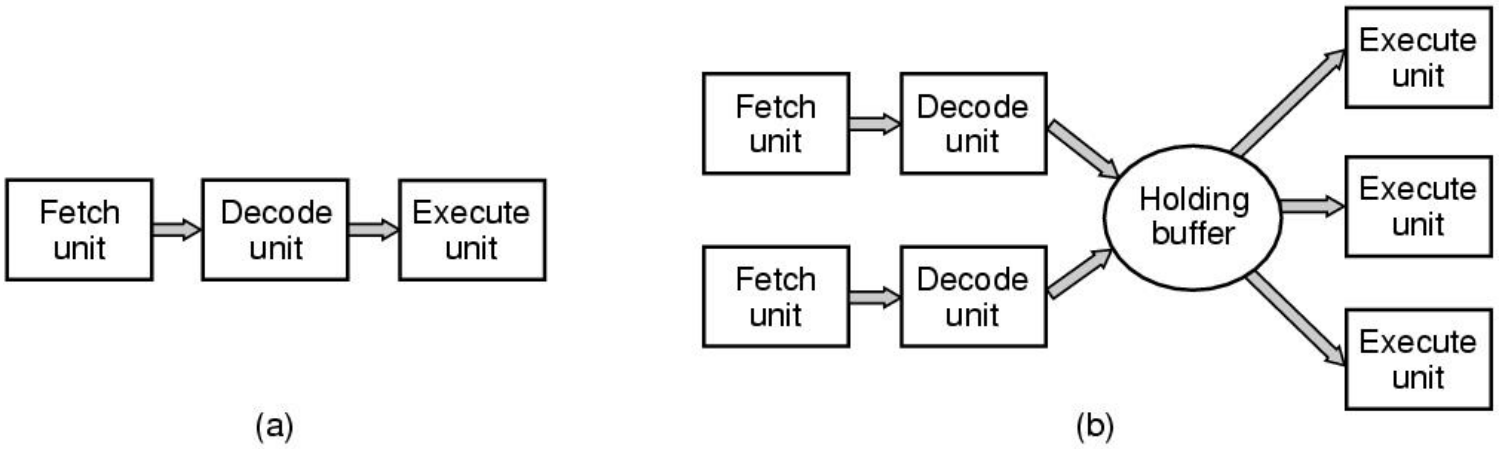
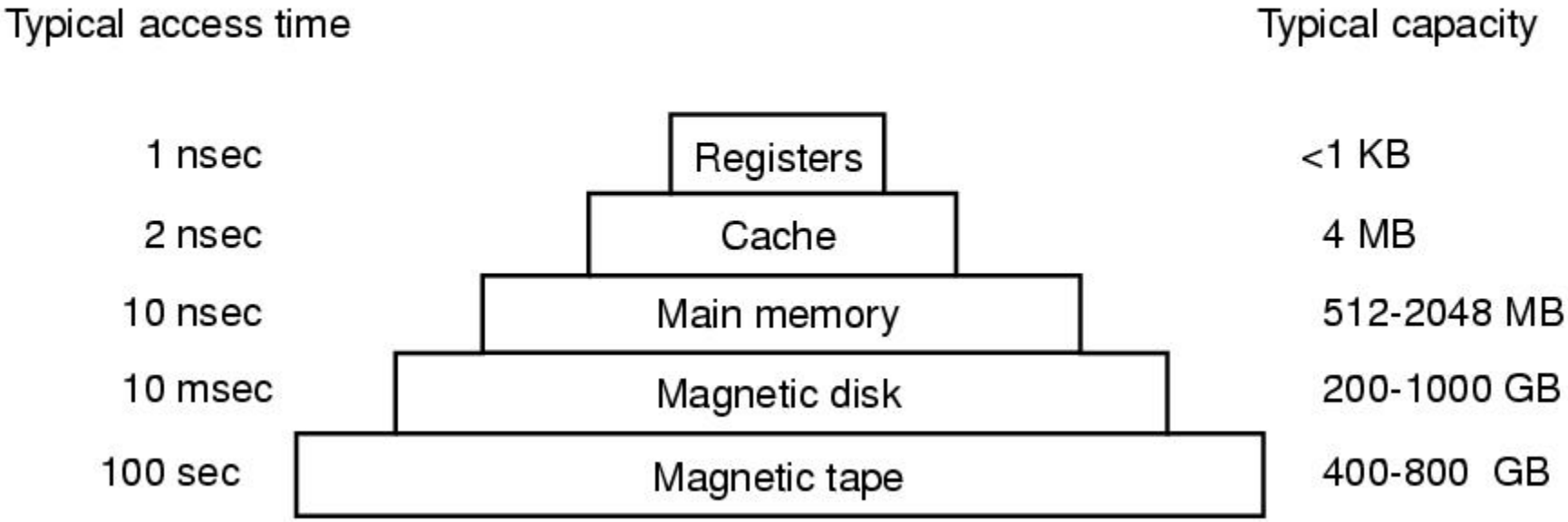


Figure 1-7. (a) A three-stage pipeline. (b) A superscalar CPU.

Memory Hierarchy



A typical memory hierarchy.
The numbers are very rough approximations.

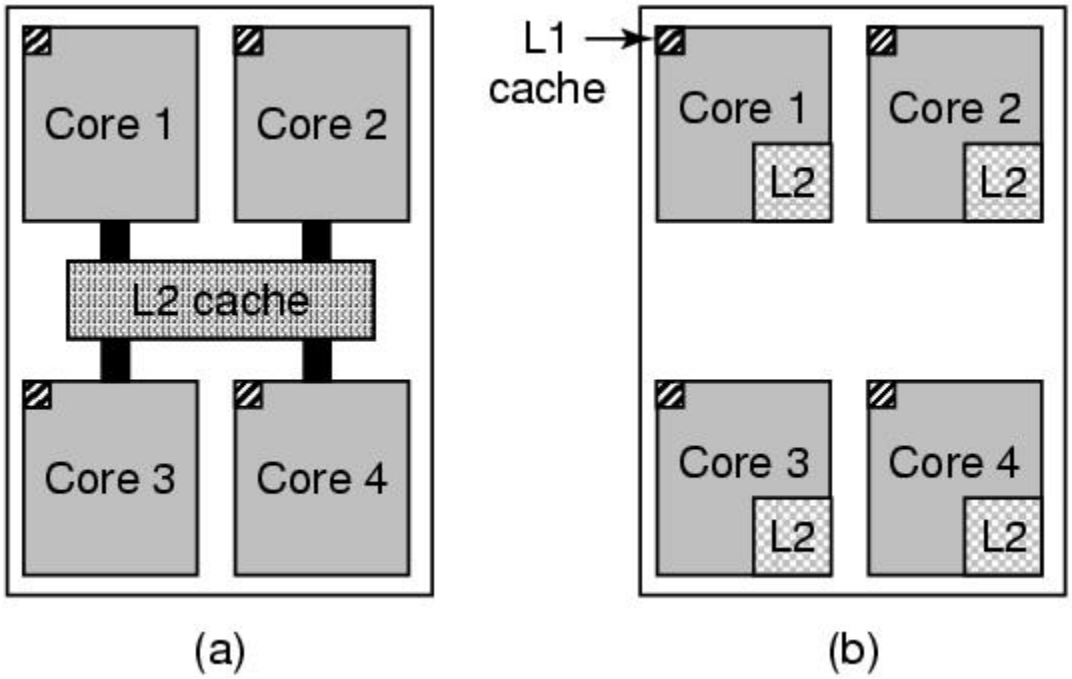
Caches

- Main memory is divided into cache lines (64 bytes)
  - 0-63 in line 1, 64-127 in line 2
- When program reads a word-cache hardware checks to see if in cache.
  - If so, then have a cache hit (2 cycles).
  - Otherwise, make request of main memory over the bus (expensive)
- Cache is expensive and is therefore limited in size
- Can have cache hierarchies
- Cache other things, like URL addresses

The 4 Cache Questions

- When to put a new item into the cache? (on a cache miss)
- Which cache line to put the new item in? (memory word determines which line)
- Which item to remove from the cache when a slot is needed? (same line new data goes into)
- Where to put a newly evicted item in main memory? (memory address determines this)

Multithreaded and Multicore Chips

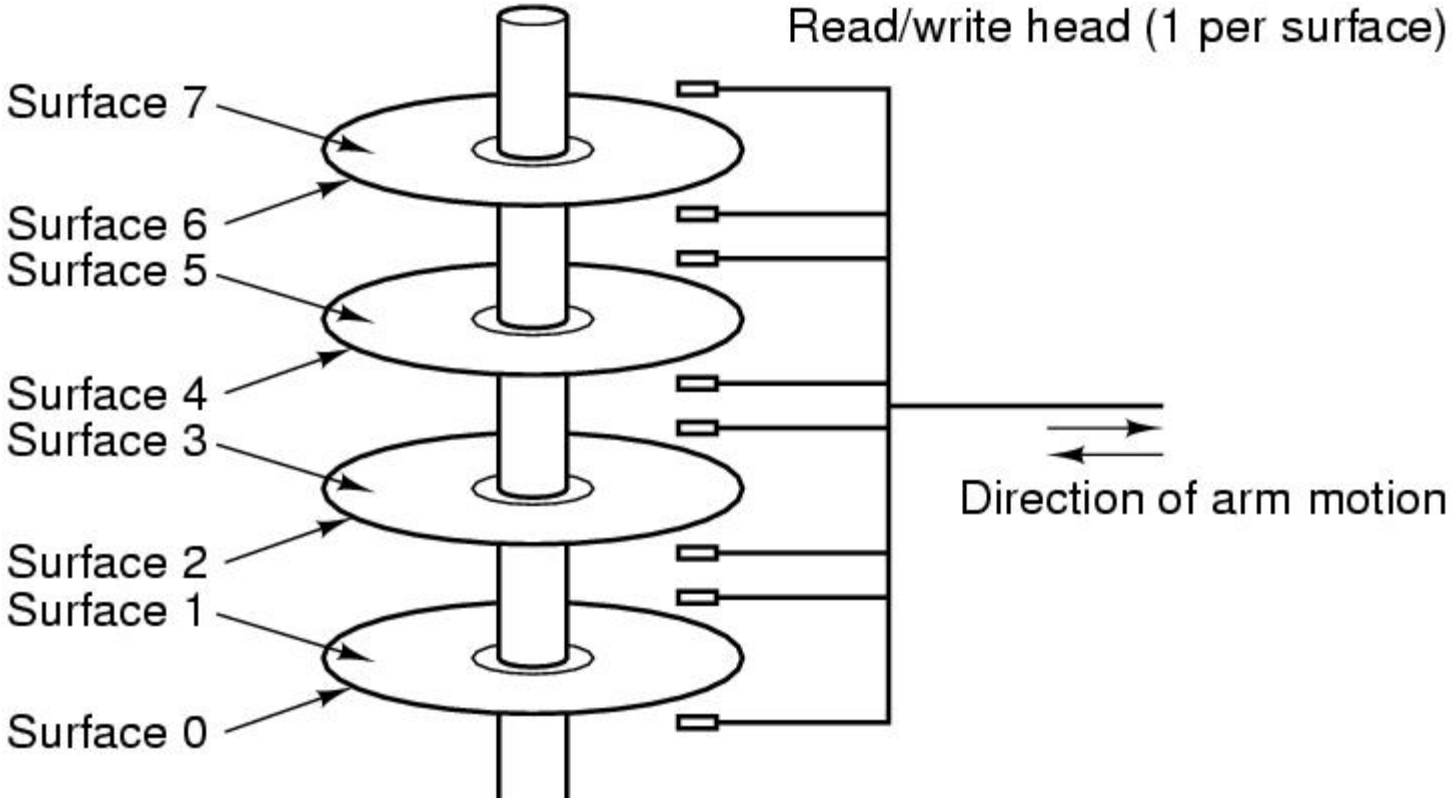


(a) A quad-core chip with a shared L2 cache.
(b) A quad-core chip with separate L2 caches.

Main Memory

- RAM
- ROM-can’t be changed. Fast, cheap
  - eg.-keeps bootstrap OS loader
- EEPROM (Electrically Erasable PROM) Can be re-written, but slowly
  - E.g.- Memory stick serves as films in digital cameras, as disk in portable music players

Disks



- Tracks are divided into sectors (512 bytes)
- Multiple tracks form a cylinder.

I/O Devices

- Controller runs a device-accepts commands from the OS and executes them
- Complicated business
  - Eg. Gets command to read sector x on disk y. Must convert to (cylinder, sector, head) address, move arm to correct cylinder, wait for sector to rotate under the head, read and store bits coming off the drive, compute checksum, store bits as words in memory
  - Controller contains a computer to run its device

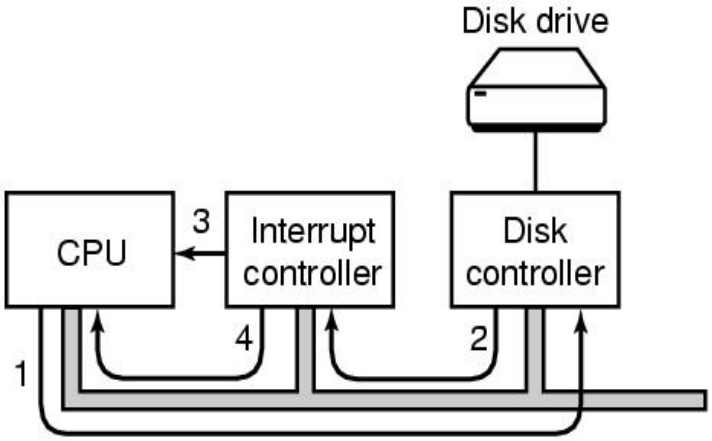
Device Driver

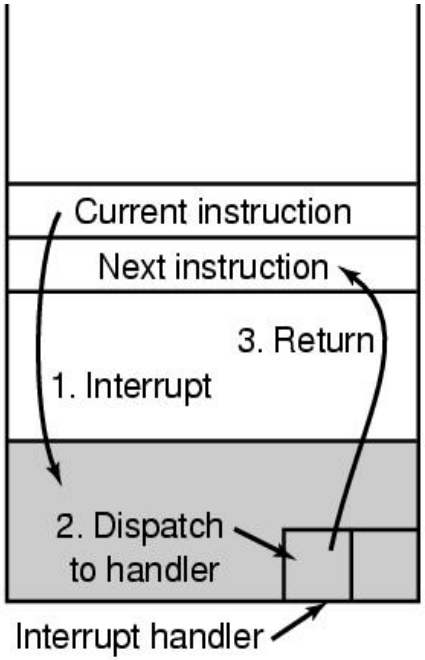
- OS software that talks to controller-gives commands, accepts responses
- Each controller manufacturer supplies a driver for each OS
- Driver runs in kernel mode
- Controller has registers which are used to communicate with the driver
- Three modes of communication
  - Polling
  - Interrupts
  - DMA

I/O by polling device

- Driver issues command to controller
- Driver polls device until it is ready
- Eg Send character to printer controller and poll until it is ready to accept the next character
- Big use of CPU
- Called programmed I/O-not really used any more

I/O by Interrupts





Generate an interrupt when I/O is finished.

Eg When character is finished being printed, interrupt CPU. Allows CPU to do something else while character is being printed

I/O by DMA

- Special (controller) chip
- Avoids using the CPU as part of the transfer to/from memory
- CPU tells chip to set up transfer and take care of it
- Chip does as it it told and interrupts CPU when it is finished

The bus hierarchy

- In the beginning there was one bus-couldn't handle the traffic when cpu and memories got faster and bigger
- Create a hierarchy of faster buses (PCI) and specialized buses (SCSI, USB)

Pentium System Buses

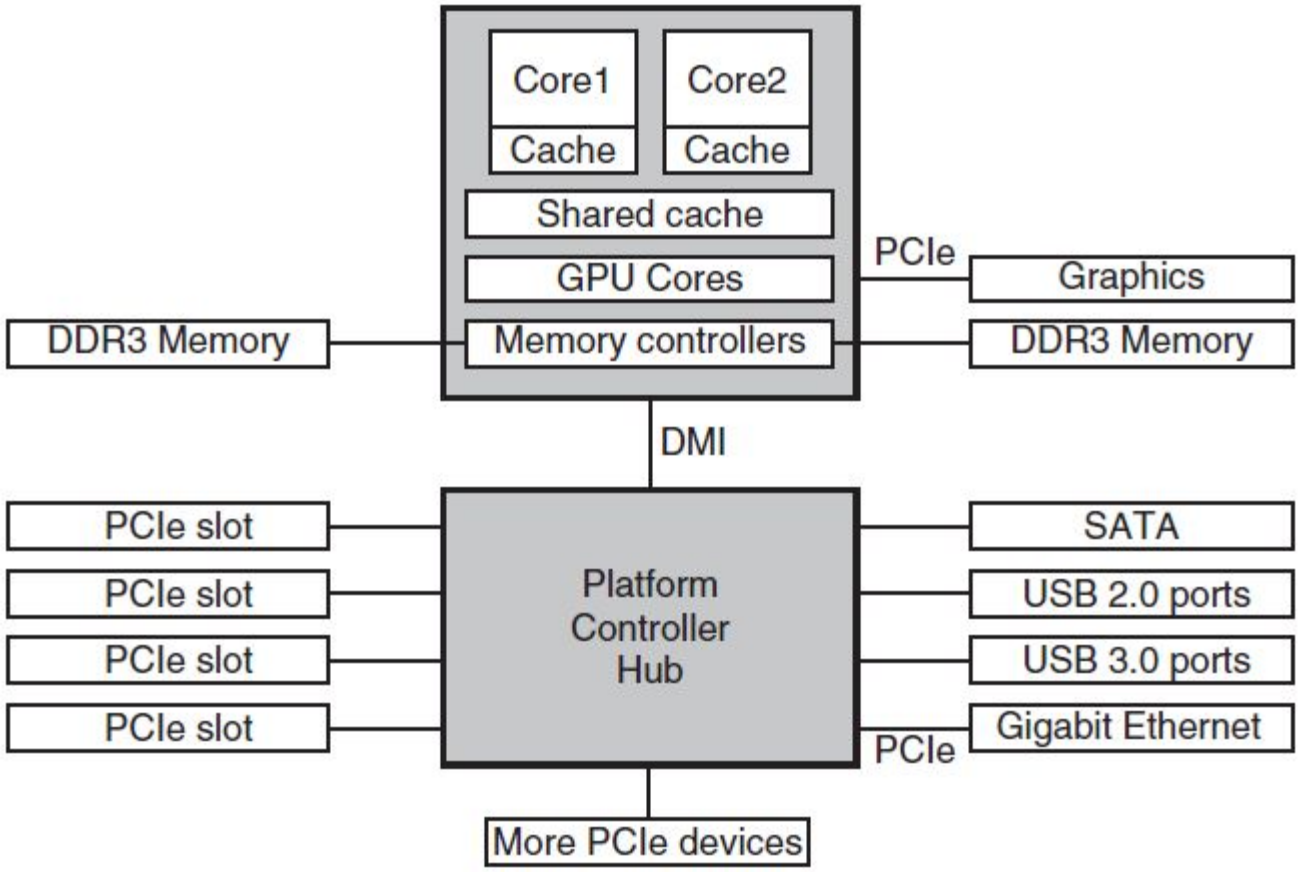


Figure 1-12. The structure of a large x86 system.

The structure of a large Pentium system

The Operating System Zoo

- Mainframe operating systems
  - Big-thousands of disks….
  - Lots of jobs with lots of I/O
  - Services-batch (payroll) transactions (airline reservations, timesharing (query database)
  - Elderly-Unix, Linux replacing them

The Operating System Zoo

- Server operating systems
  - Workstations
  - File, print, web servers
  - BSD, Linux, Windows
- Multiprocessor operating systems
  - Use multiple cores

The Operating System Zoo

- PC operating systems-Linux, Mac, Windows
- Smart phone operating systems- Android, iPhone, Blackberry
  - No hard disk
  - Palm, Symbian popular OS's

The Operating System Zoo

- Embedded operating systems-TV sets, cars, DVDs, MP3s
  - Everything is in ROM (no apps can run on it)
  - QNx, Vxworks
- Real time operating systems
  - Hard (eg. factory) deadline
  - Soft (eg. multi-media) deadline
- Smart card OS (eg border crossing cards)
  - Java in ROM

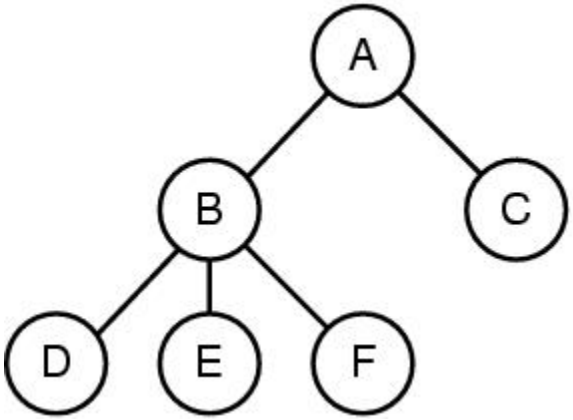
Operating System Concepts

- Processes
- Address spaces
- Files
- Input/Output
- Protection
- The shell

Processes

- Program in execution
  - Container that holds all the information needed to run a program
- Lives in address space
  - Contain executable program, data and stack
- Process table
  - Keeps info about process
  - Used to re-start process
- Shell (command interpreter) reads commands from terminal

A Process Tree

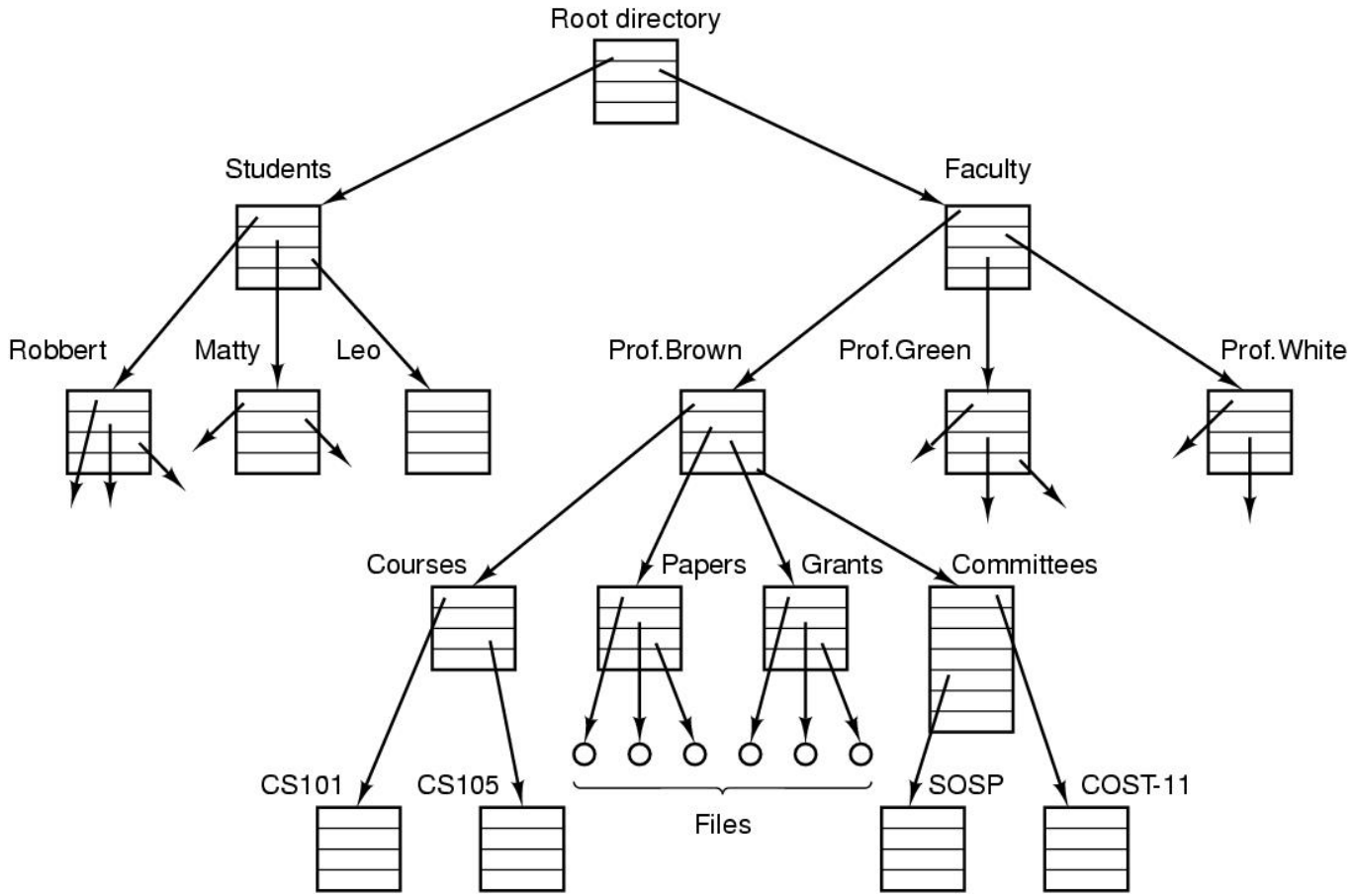


Process creates child processes
Tree structure

Process

- Have UID's and group ID's (GID)
- Can communicate with one another (IPC)

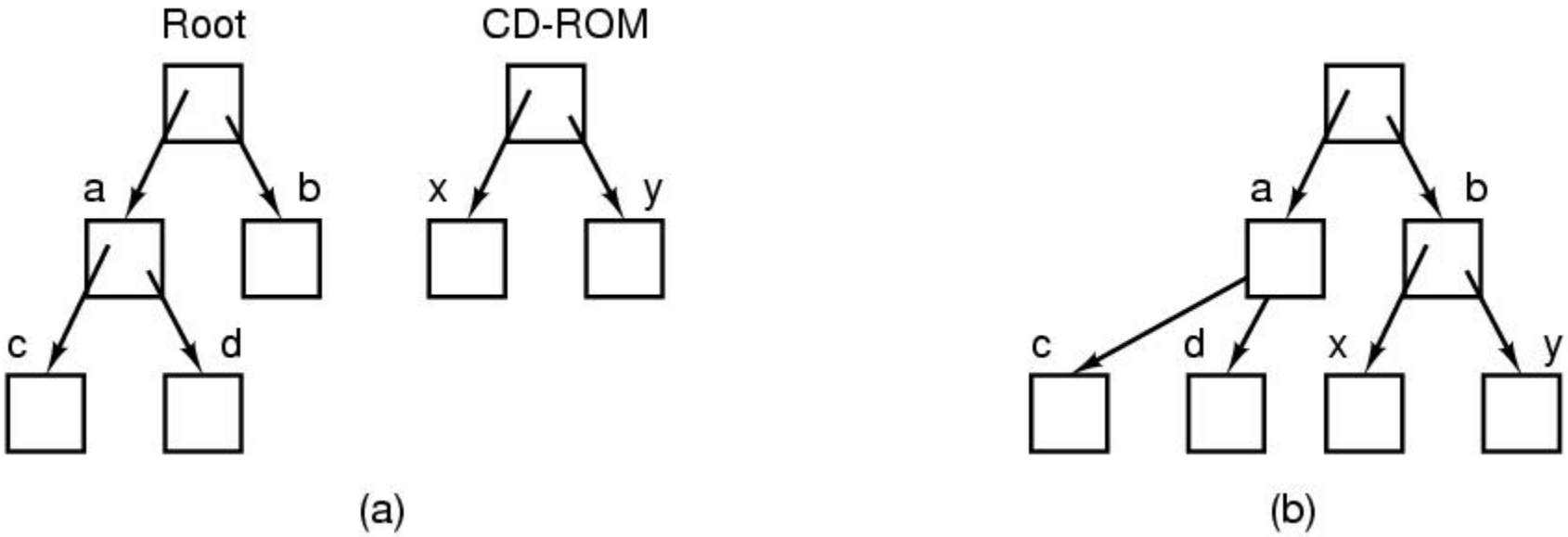
File directory



The directory is organized as a tree

Root directory at the top. Path proceeds from the root (e.g. faculty/prof brown/courses)

Mounting Files in UNIX



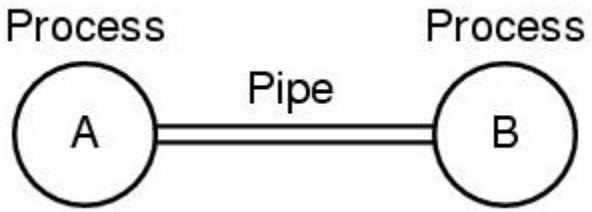
A CD-ROM is mounted on directory b.

Special files

- Special files-can use same calls for I/O as for files. OS treats them as files.
  - Block special files (disks)
  - Character special files (line printers, modems)
  - Kept in /dev directory, e.g. /dev/lp is line printer

Unix Pipe

Processes communicate by writing into/reading from a file in Unix



A and B write into the pipe and read from the pipe.

I/O, Protection/Shell

- I/O-big part of OS
- Protection-UNIX uses rwx bits for each file
  - 3 bits for owner, 3 for group, 3 for everyone else
- Shell (command interpreter)
  - UNIX
  - Has lots of flavors-sh, bash, csh, ssh…..
  - Sort<file1>file2
  - Cat file 1 file 2 file3 | sort > /dev/lp

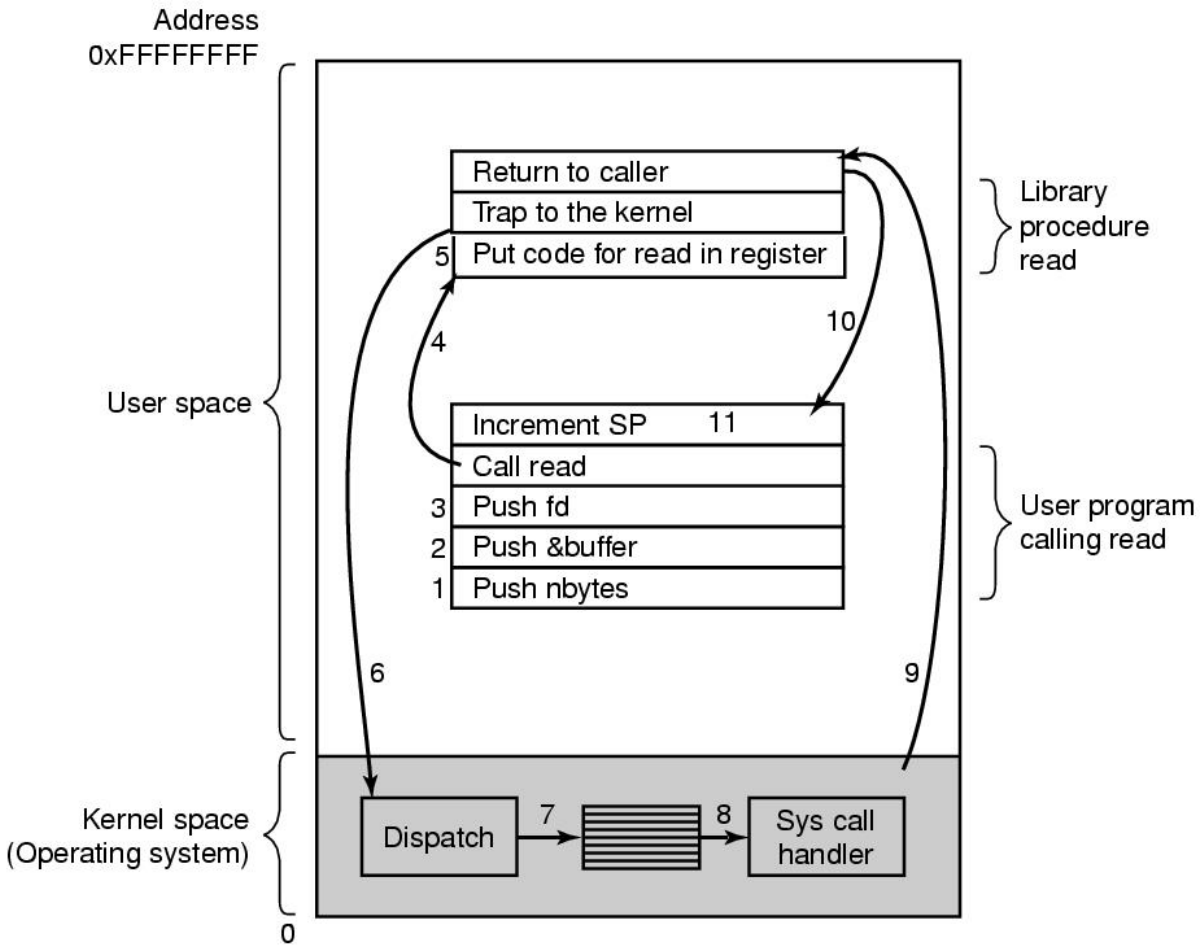
System Calls

- Interface between user programs and OS
- Varies from OS to OS
- System call issued by user program
- Call uses a library call of the same name
- Library routine puts machine into kernel modes (by issuing a special instruction)
- Finds actual routine for system call in a table
- Does the work involved in the call
- Returns to user program

Unix Read System Call

- Count=read(fd, buffer, nbytes)
  - fd is a file descriptor. When a file is opened, permissions are checked. If access is allowed, a number (fd) is returned. Then file can be read/written
  - nbytes is number of bytes in file
  - buffer is where read deposits the bytes

System Calls



read(fd, buffer, nbytes).

Look at System Calls

- Start with process management
  - See how they are used in shell
  - Then you write a shell
- Then do a brief summary of calls for
  - File management
  - Other useful calls

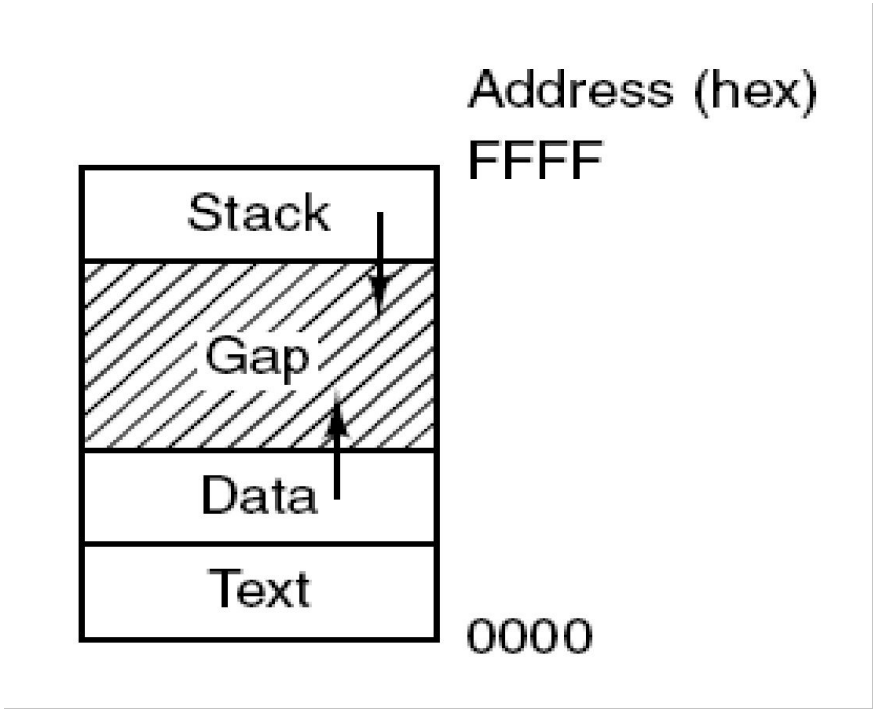
System Calls for Process Management

Process management

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

Memory Layout

Processes have three segments:
text, data, and stack.



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

File mode

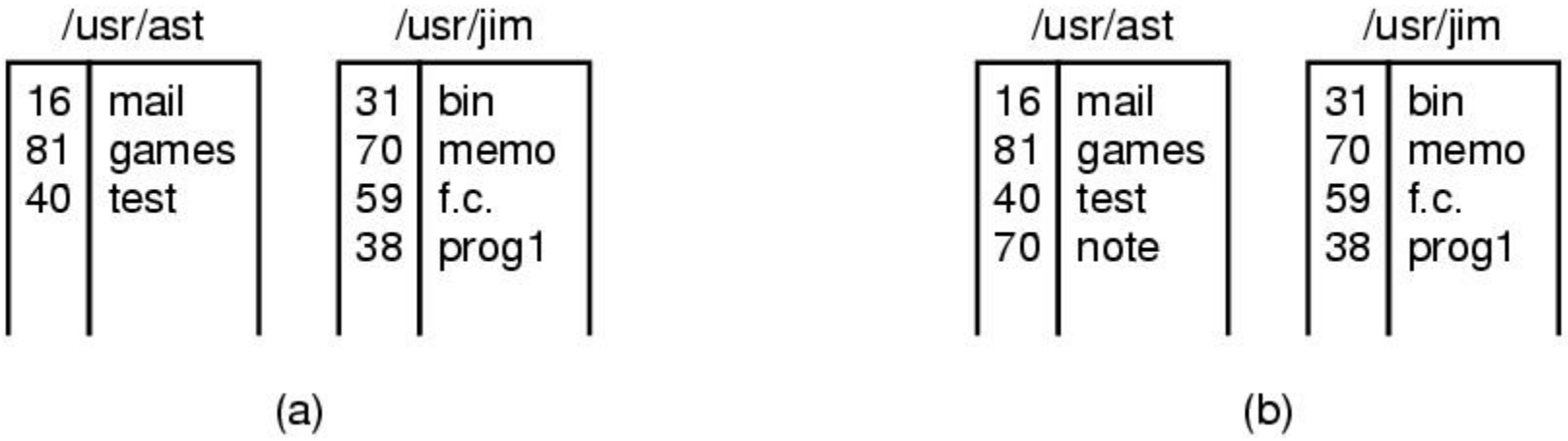
- System keeps track of it
  - Regular, special
  - Date of creation
  - Size
  - Access status of file via stat command

System Calls for Directory and File 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

Linking

link("/usr/jim/memo", "/usr/ast/note");



- In Unix, each file is identified by an i-number
  - i-number indexes into i-node table
- Link creates a new directory entry with same i-number
  - Has a new name (note instead of memo)

Mount System Call

- mount("/dev/fd0", "/mnt", 0) is system call
- File comes from drive 0 and is mounted on binary file /mnt.
- Third parameter tells if it is read or read-write
- Result is that file on drive zero can be accessed from a directory
- Saw this example before with CD-same mechanism for memory sticks and portions of hard drives

Mount System Call

Figure 1-22. (a) File system before the mount. (b) File system after the mount.

Other System Calls

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

Windows Win32 API

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

The Win32 API calls that roughly correspond to the UNIX calls

Operating Systems Structure

- Low level software system that
  – manages all applications
  – implements an interface between applications and resources
  – manages available resources
- Resource manager
- Interface
- Virtual Machine

Operating Systems Structure

Requirements

✔Portable

- Applications and OS should run on different systems

-Linux can be executed in different HW architectures

-Cross platform application can execute in different OS (Linux/Unix/MAC OS X)

Operating Systems Structure

Requirements

✔Performance

– the use of available resources should be efficient

✔Security

– users and processes should work independently, although sharing resources

✔Ease of use

– The programming interface should be simple but expressive

Operating Systems Structure

A general purpose OS is composed of:

Process manager

✔Multiplexes the CPU time between the multiple execution units (processes)

Memory manager

✔controls, manages and multiplexes the access to physical and virtual memory

Inter-process communication

✔Implements and handles mechanism for processes to communicate

Operating Systems Structure

A general purpose OS is composed of:

I/O manager

- ✔ manages communication with peripheral (keyboard/screen, disk, network)

User interface

- ✔ command line interpreter
- ✔ GUI

Operating Systems Structure

A general purpose OS is composed of:

File System manager

- ✔ manages and organizes data available on disks (file systems)

Function calls

- ✔ Programming functions that allow applications to use OS services (memory, disk, I/O)

Operating Systems Structure

A general purpose OS can be divided in:

– **OS kernel**

✔Code executed in privileged mode

– **User space**

✔Code executed in non privileged mode

– **Service /daemon**

✔Application that executed in the background (server)

– **Utility programs**

✔Application provided by the OS and executed by the user (editor, shell, compiler)

Operating Systems Structure

A general purpose OS can be divided in:

– **System calls**

✔functions that implement parts of the OS services or utilities

✔can be used inside the kernel

✔Manage and change internal structure.

– **C lib**

✔set of user level functions

Operating Systems Structure

Types:

- Monolithic
- Layered
- Micro kernel
- Distributed
- VM based
- The Java Virtual Machine
- Exokernels

Monolithic Operating Systems Structure

- OS composed of a single module
  -All kernel routines are together
  -All data and code use same memory space
- A system call interface
- Examples:
  – Linux, BSD Unix
  – Windows NT (hybrid)

Monolithic Operating Systems Structure

- Pros
  - – Performance is good if the program runs well.
  - – Shared kernel space
  - - Easy to implement
  - - Low overheads
- Cons
  - – low security mechanisms (one driver can mess other drivers)
  - – Difficult to evolve (reboot of system needed)

Monolithic Operating Systems Structure

How kernel works

- 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 Systems Structure

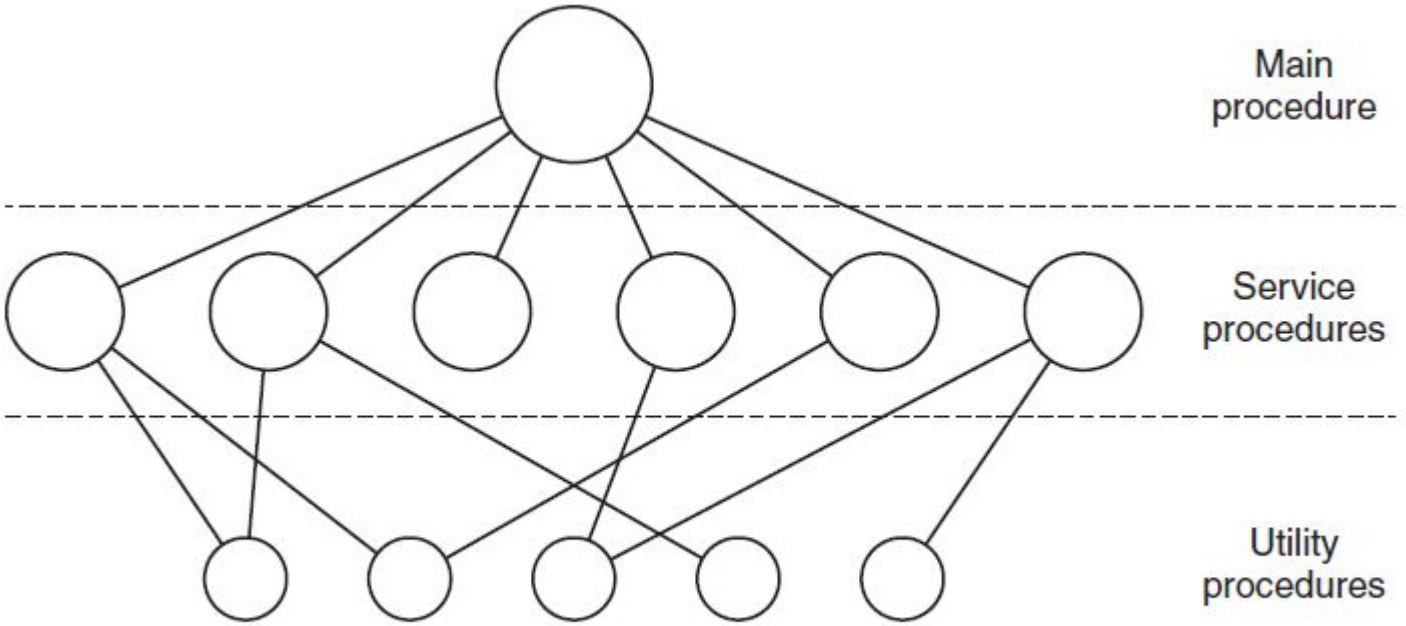
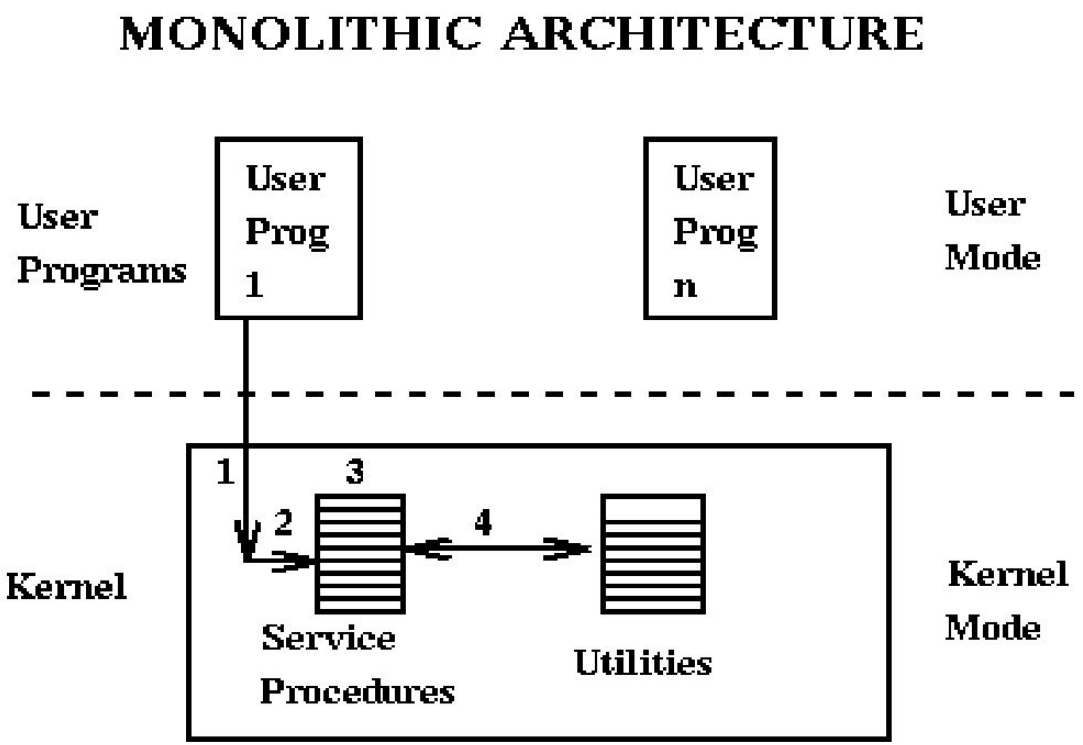


Figure 1-24. A simple structuring model for a monolithic system.

Monolithic Operating Systems Structure



1. System call (User->Kernel Mode)
2. Check parameters
3. Call service routine
4. Service Routine call utilities

Reschedule/Return to user

Monolithic Systems

```
#define FALSE  0
#define TRUE   1
#define N      2                        /* number of processes */

int turn;                               /* whose turn is it? */
int interested[N];                      /* all values initially 0 (FALSE) */

void enter_region(int process);         /* process is 0 or 1 */
{
     int other;                         /* number of the other process */

     other = 1 – process;               /* the opposite of process */
     interested[process] = TRUE;        /* show that you are interested */
     turn = process;                    /* set flag */
     while (turn == process && interested[other] == TRUE) /* null statement */ ;
}

void leave_region(int process)          /* process: who is leaving */
{
     interested[process] = FALSE;       /* indicate departure from critical region */
}
```

A simple structuring model for a monolithic system.

Layered Systems- operating system

- Components are divided into layers
  - grouping similar components
- Each layer only interacts with:
  - the bottom layer - requesting services
  - to top layer - answering requests
- Higher level layer
  - Applications

Layered Systems- operating system

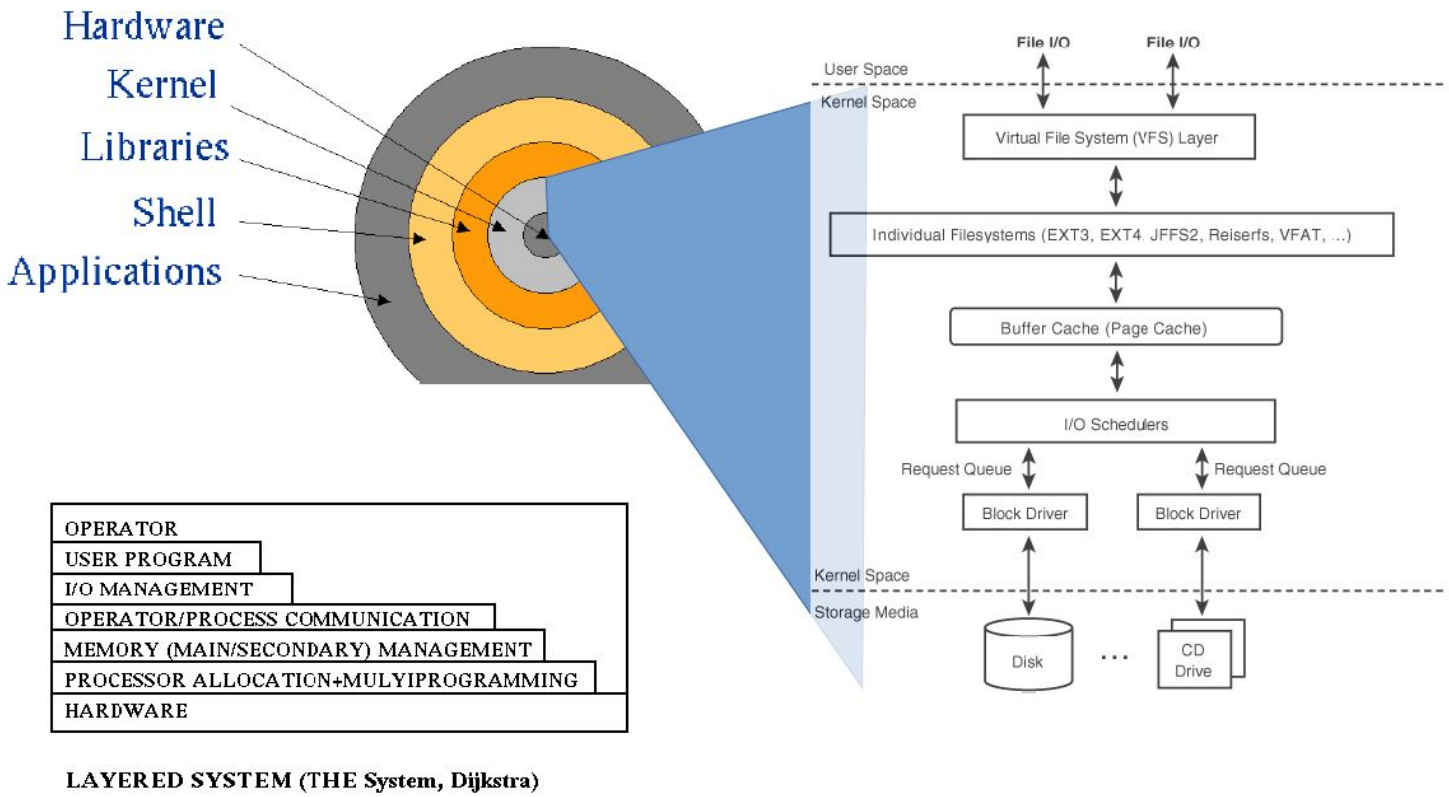
- lowest level layer
  - hardware
- Advantaged
  - good structure, well defined interface, ...
- Disadvantages
  - can be slow, may be difficult to define layers.

Layered Systems-THE operating system

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

Dijkstra's the author

Layered Systems-THE operating system



LAYERED SYSTEM (THE System, Dijkstra)

Microkernels

- Removes from kernel as much functionality as possible,
  - limiting the amount of code executed in privileged mode
  - allow easy modifications and extensions
- Most microkernels provide basic process and memory management, and message passing between other services
- Security and protection can be enhanced
  - most services are performed in user mode, not kernel mode.

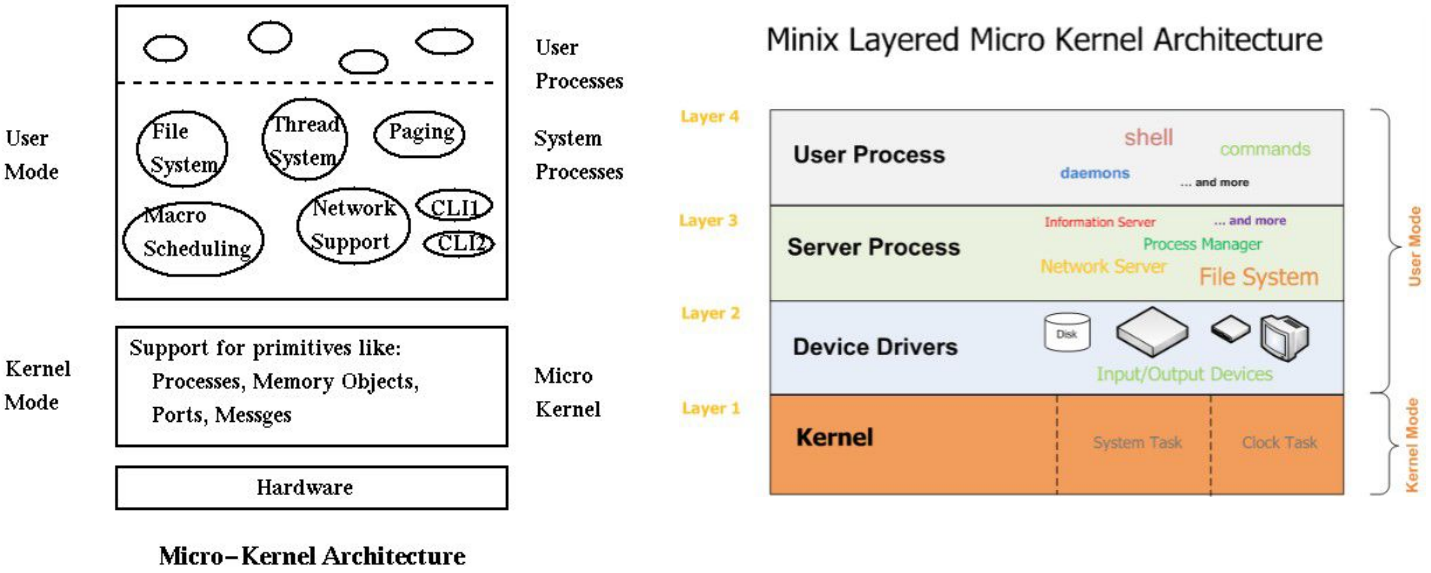
Microkernels

- System expansion can also be easier,
  - only involves adding more system applications, not rebuilding a new kernel.
- Windows NT was originally microkernel
  - but suffered from performance problems relative to Windows 95.
  - NT 4.0 improved performance by moving more services into the kernel
- Multiple OS can be built on top of a micro-kernel
  - Each operating system will make use of different system processes.

Microkernels

- Small number of processes are allowed in the kernel
- Minimizes effects of bugs
  - Don’t want bug in driver to crash system
- Put mechanism in kernel and policy outside the kernel
  - Mechanism- schedule processes by priority scheduling algorithm
  - Policy-assign process priorities in user space

Microkernels

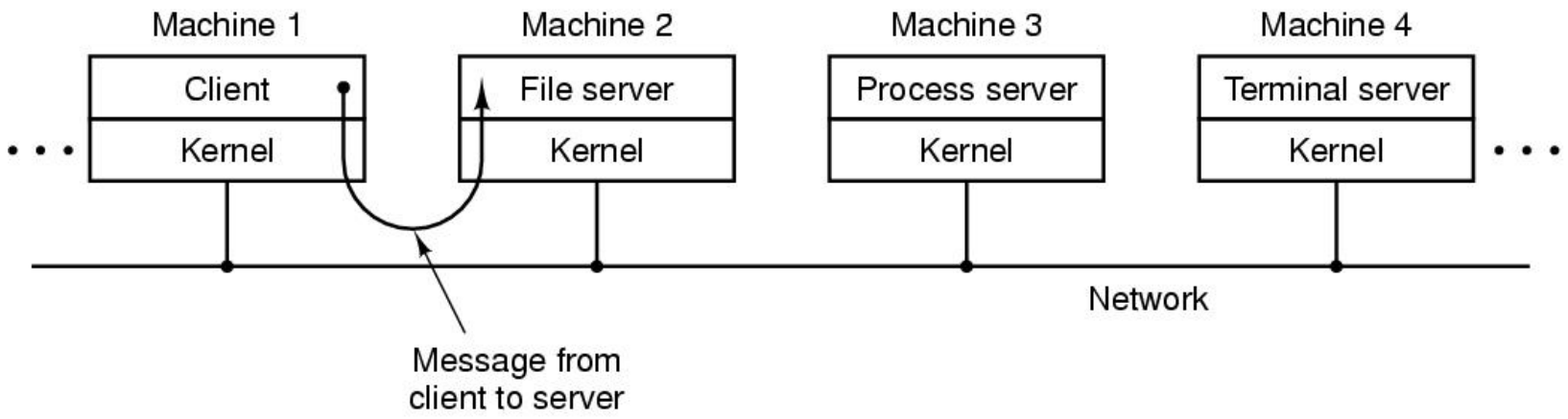


Structure of the MINIX 3 system.

Distributed OS

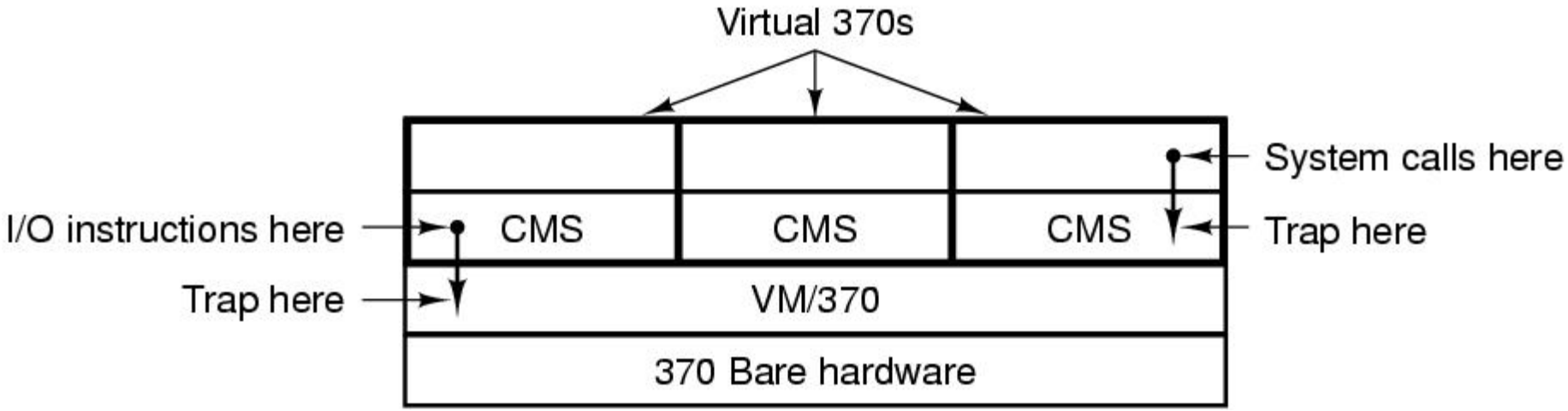
- Each component/service is a separated process
  - on the same machine
  - on different machines
- Components interactions
  - messages / Remote procedures
- Distributed File System
- Distributed memory
- Distributed processes

Client-Server Model



The client-server model over a network.

Virtual Machines (1)



The structure of VM/370 with CMS.

Virtual Machines (2)

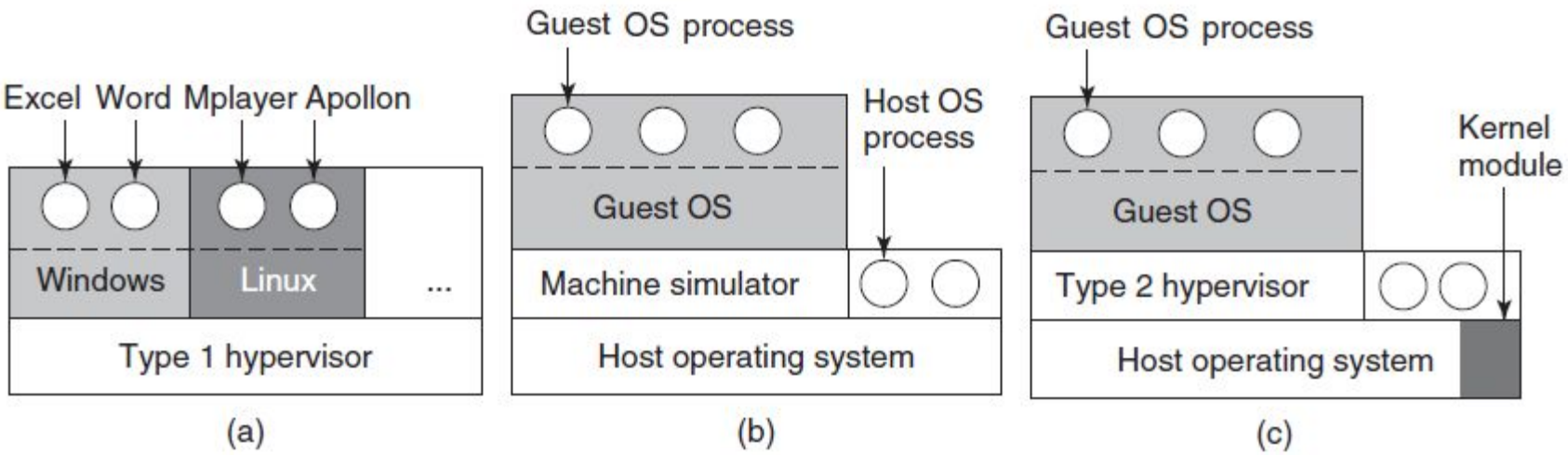
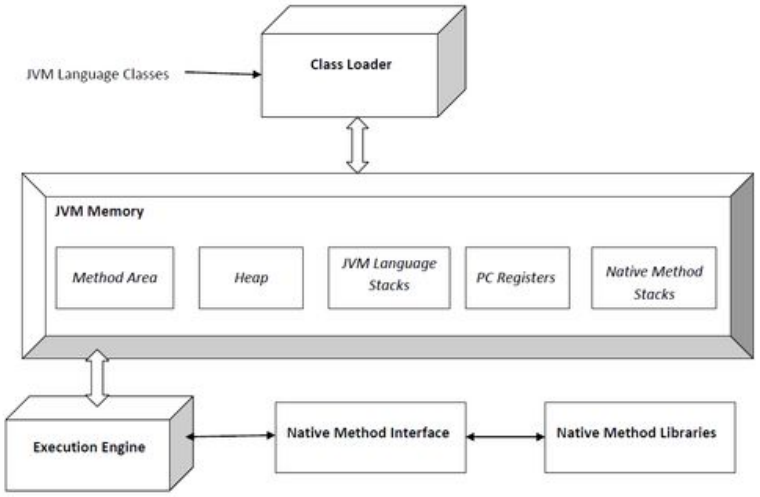


Figure 1-29. (a) A type 1 hypervisor. (b) A pure type 2 hypervisor. (c) A practical type 2 hypervisor.

The Java Virtual Machine



Java applications are called WORA (Write Once Run Everywhere). This means a programmer can develop Java code on one system and can expect it to run on any other Java enabled system without any adjustment. This is all possible because of JVM.

The Java Virtual Machine

Java responsible for three activities.

- Loading
- Linking
- Initialization

Loading : The Class loader reads the *.class* file, generate the corresponding binary data and save it in method area.

Linking : Performs verification, preparation, and (optionally) resolution.

Initialization : In this phase, all static variables are assigned with their values defined in the code and static block(if any). This is executed from top to bottom in a class and from parent to child in class hierarchy.

Exokernels

Rather than cloning the actual machine, as is done with virtual machines, another strategy is partitioning it, in other words, giving each user a subset of the resources.

At the bottom layer, running in kernel mode, is a program called the exokernel. Its job is to allocate resources to virtual machines and then check attempts to use them to make sure no machine is trying to use somebody else's resources.