METU - EE 442 - Operating Systems

Introduction

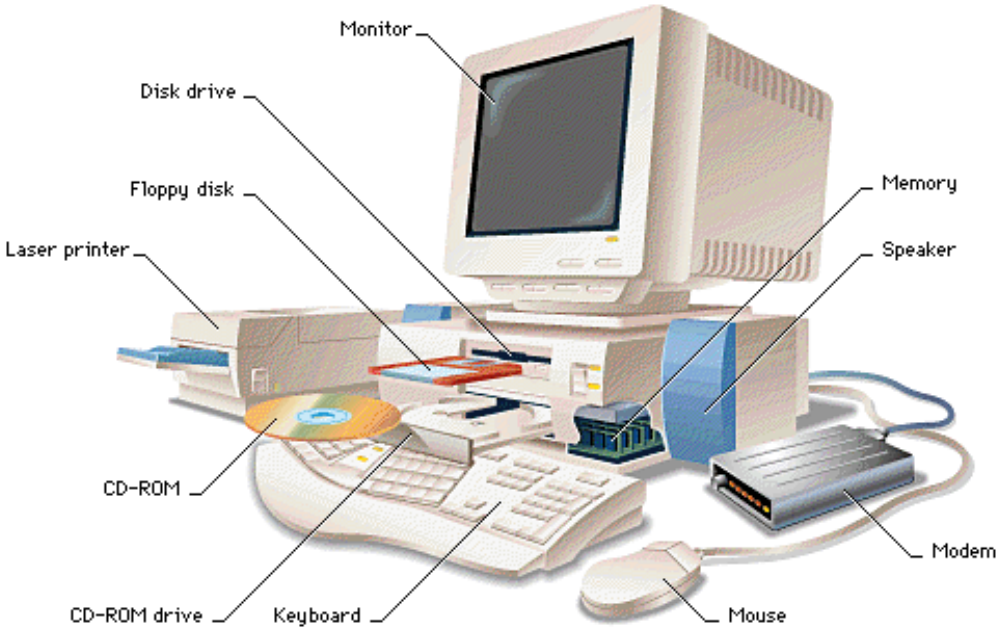
Chapter 1

Cüneyt F. Bazlamaçcı (METU-EEE)

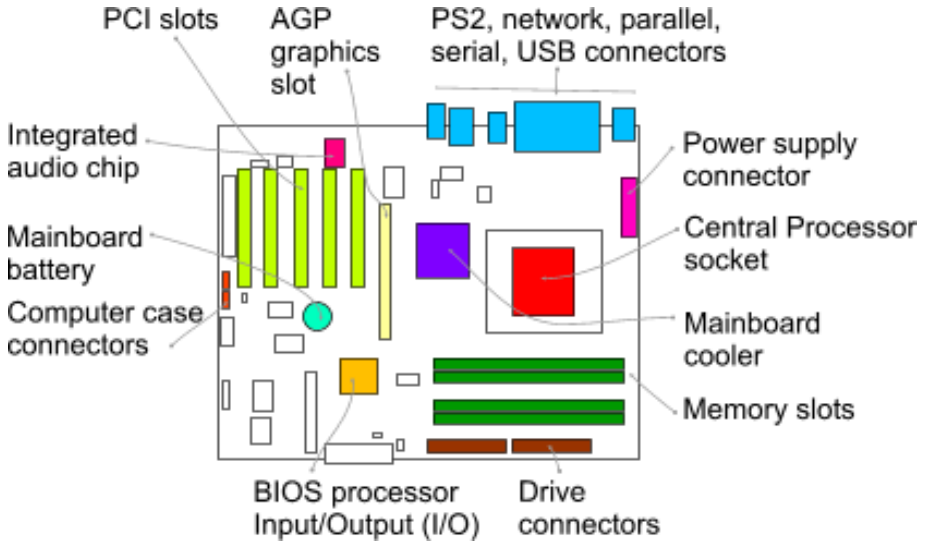
The slides include material from Tanenbaum & Bos, Modern Operating Systems, 4th ed., Pearson and Halıcı, U., EE442 Lecture Notes

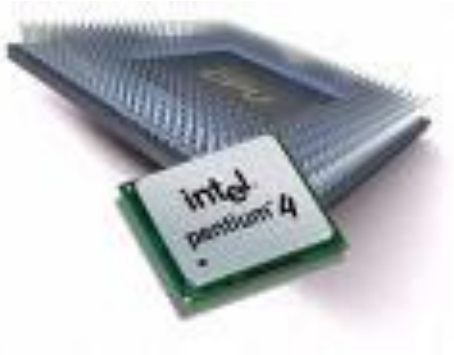
Components of a Modern Computer (1)

- One or more processors
- Main memory
- Disks
- Printers
- Keyboard
- Mouse
- Display
- Network interfaces
- I/O devices



Components of a Modern Computer (2)





Components of a Modern Computer (3)

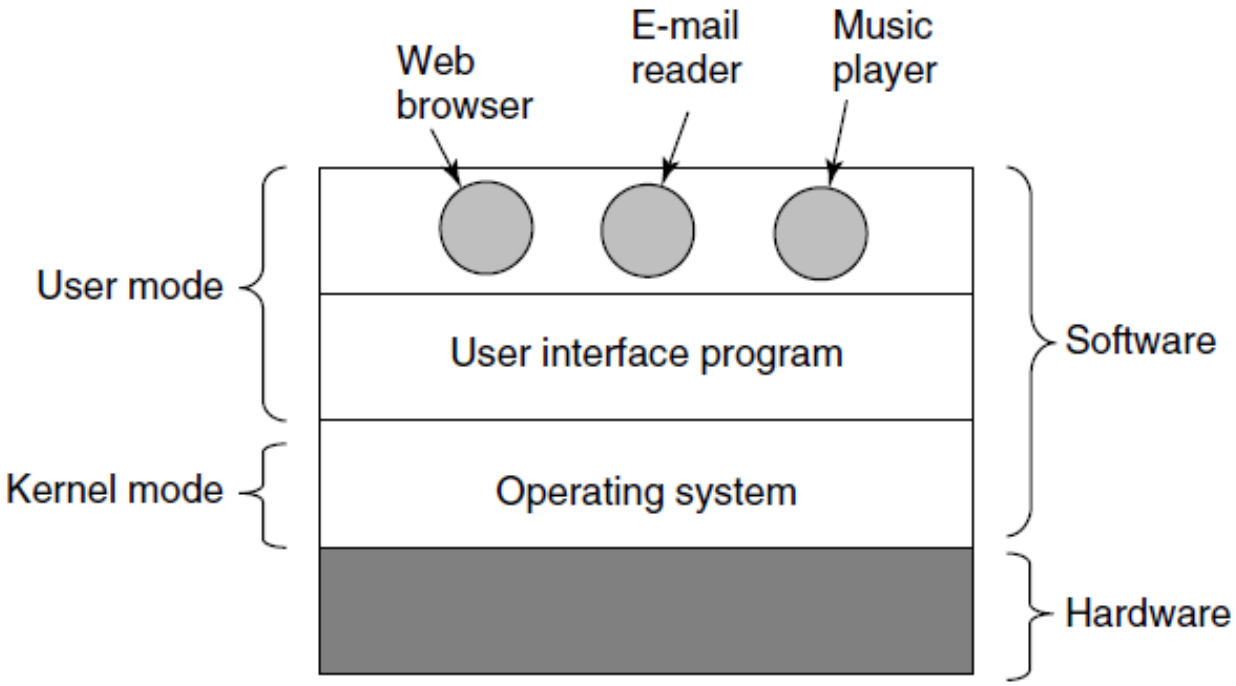


Figure 1-1. Where the operating system fits in.

Components of a Modern Computer (3)

- OS uses **kernel mode** of the microprocessor, whereas other programs use **user mode**.

- The difference is that; all hardware instructions are valid in **kernel mode**, where some of them cannot be used in the **user mode**.

Components of a Modern Computer (4)

- A user **can** change his program or write his own program piece if he is not satisfied.
- A user **cannot** change, for example, the clock interrupt handler, which is part of OS, and is protected by H/W against attempts by users to modify it.
- Note: embedded systems or interpreted systems might not have a kernel mode.

Operating System (OS) as an Extended Machine (1)

- To perform addition in a system where there is no OS installed, we may need to consider memory locations and hardware knowledge of the underlying system as follows:

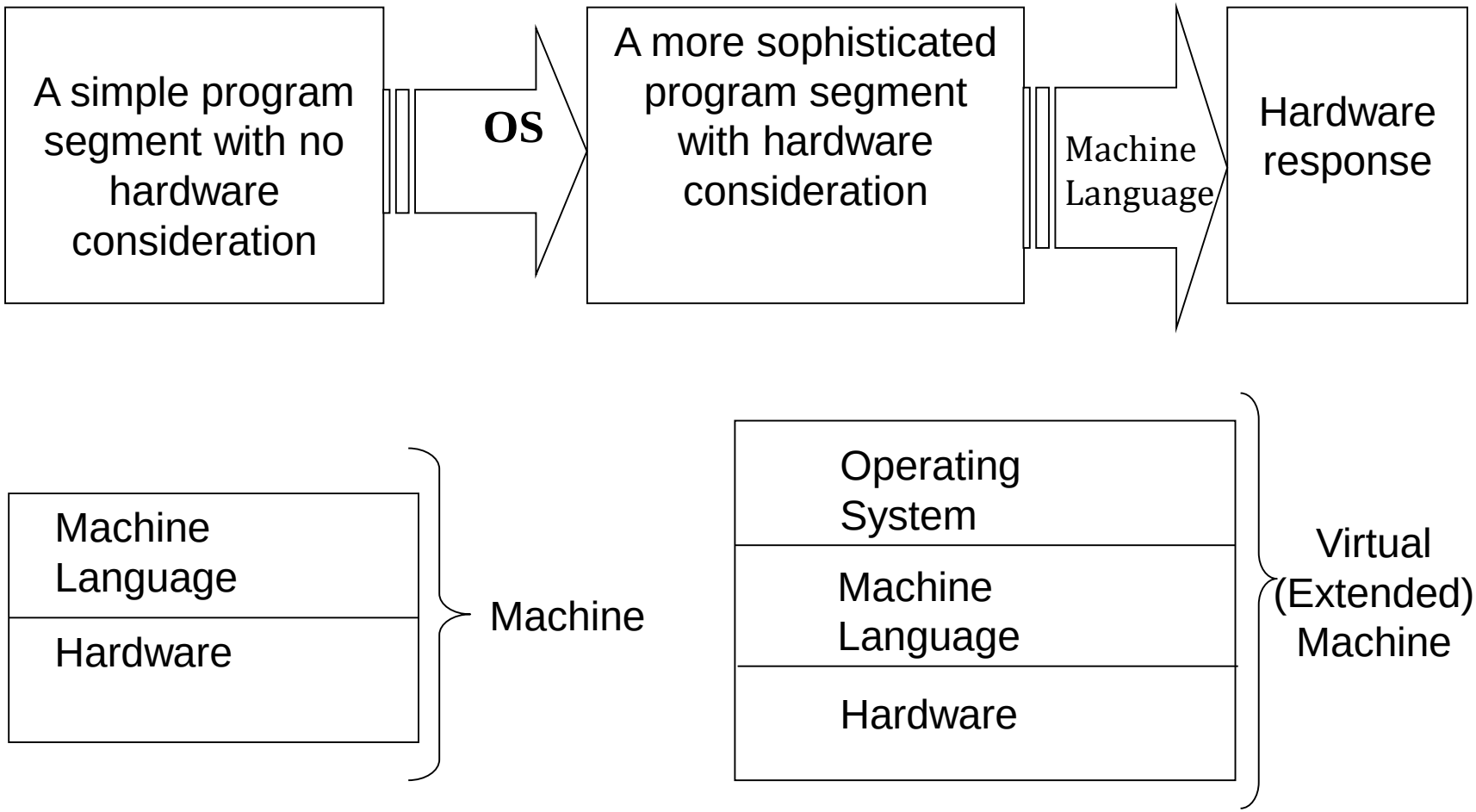
  (Assuming MC 6811 or any other similar computer hardware knowledge)

  LDAA $80 → Loading register A the number at memory location 80
  LDAB $81 → Loading register B the number at memory location 81
  ADDB → Adding these two numbers
  STAA $55 → Storing the sum in register A to memory location 55

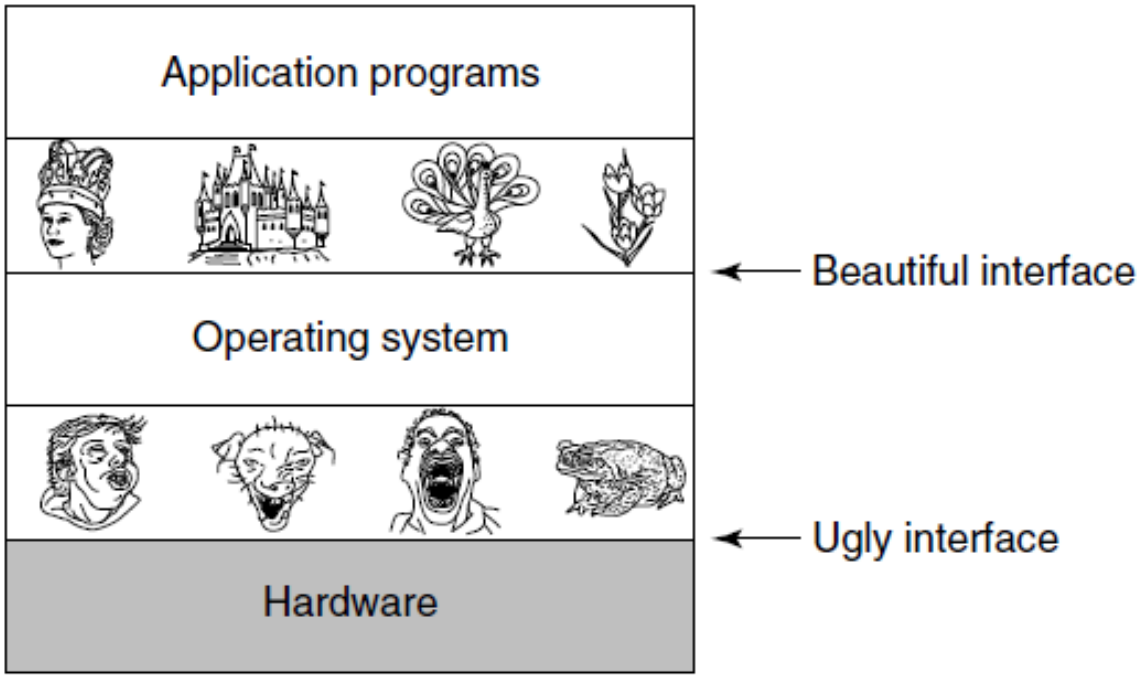
OS as an Extended Machine (2)

- The above program segment would not work for an 8086 machine, but
- If we have an intermediate layer, our programs may obtain *some advantage of mobility* by not dealing with the hardware.
- and "c = a + b ;" syntax may be suitable for both processors.
- Another example:
  - disk drive hardware ← disk driver ← files

OS as an Extended Machine (3)

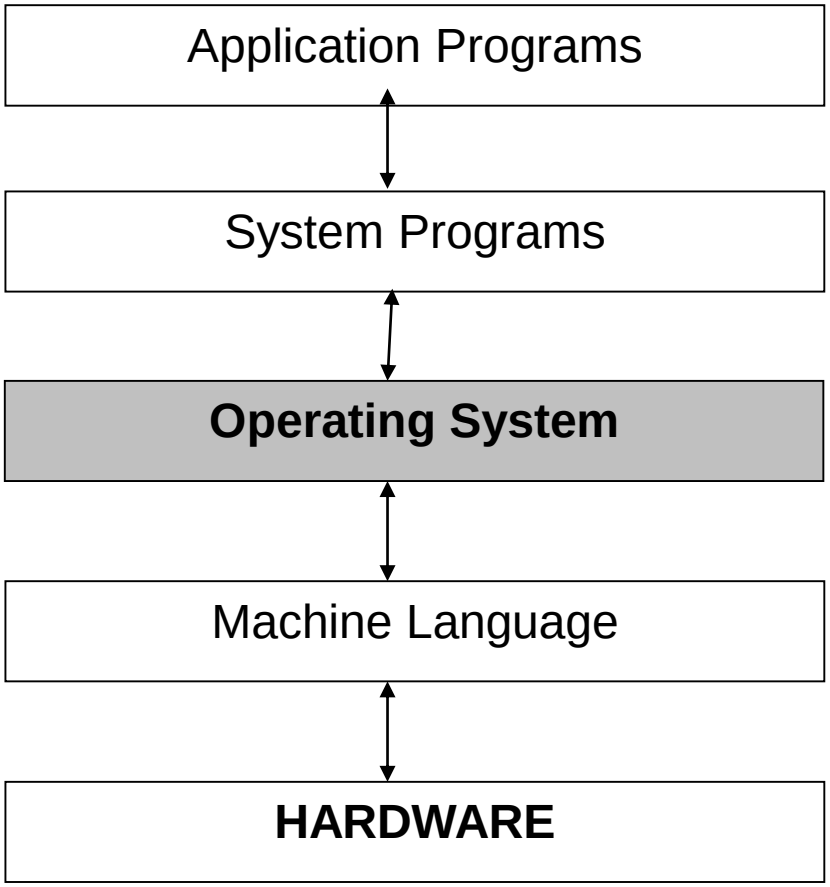


OS as an Extended Machine (4)



Operating systems turn ugly hardware into beautiful abstractions.

OS as an Extended Machine (5)



OS as a Resource Manager (1)

- Top down view
  - Provide abstractions to application programs
- Bottom up view
  - Manage pieces of complex system
- Alternative view
  - Provide orderly, controlled allocation of resources

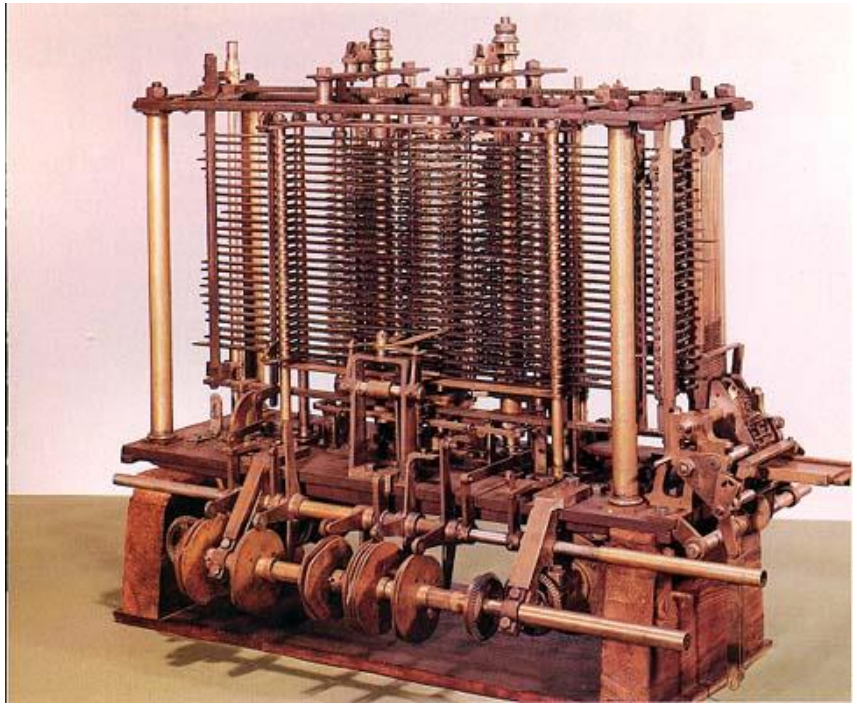
OS as a Resource Manager (2)

- An operating system
  - keeps track of which programs are using which resource,
  - grant resource requests,
  - account for usage, and
  - mediate conflicting requests from different programs and users.
- Resource management includes multiplexing (sharing) resources in time and in space.

History of Operating Systems (1)

- The first generation (1945–55) vacuum tubes
- The second generation (1955–65) transistors and batch systems
- The third generation (1965–1980) ICs and multiprogramming
- The fourth generation (1980–present) personal computers
- The fifth generation (1990–present) mobile computers

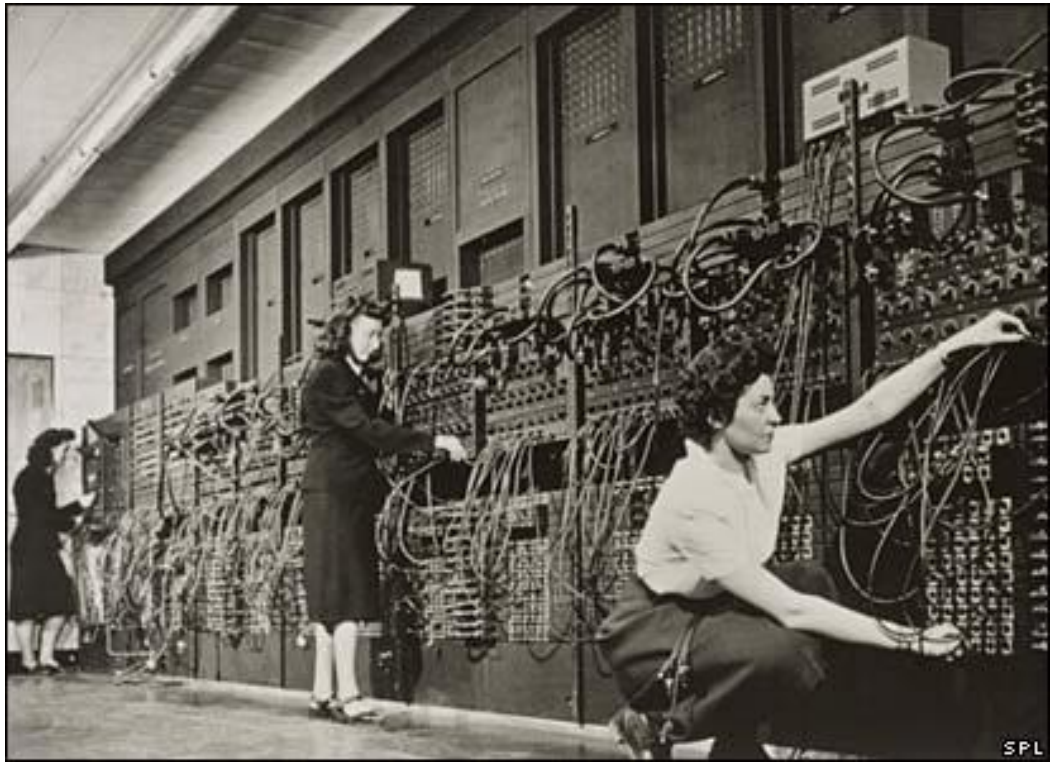
1st Generation Computing



Babbage's analytical engine
(designed in 1840's by Charles Babbage, but cold not be constructed by him.
An earlier and simpler version is constructed in 2002, in London)

http://www.computerhistory.org/babbage/

1st Generation Computing (2)

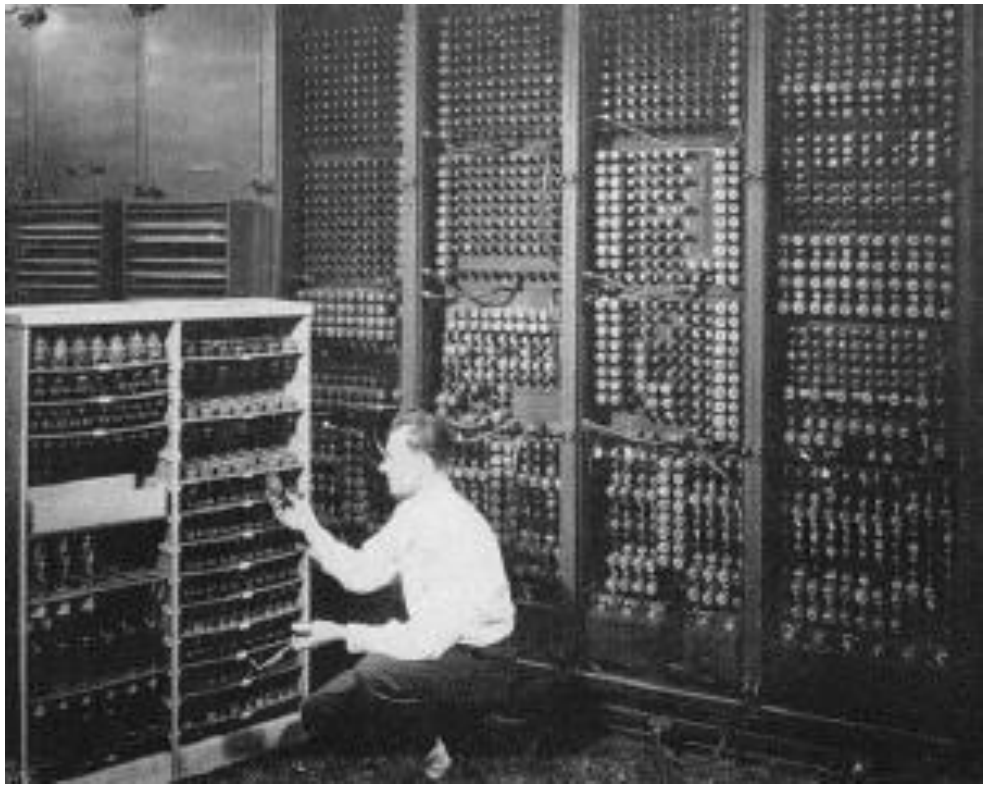


ENIAC 1943

ENIAC (Electronic Numerical Integrator and Computer), at the U.S. Army's Aberdeen Proving Ground in Maryland.

- built in the 1940s,
- weighed 30 tons,
- 2.5m high, 1m deep, and 30m long
- contained over 18,000 vacuum tubes that were cooled by 80 air blowers.

1st Generation Computing (3)

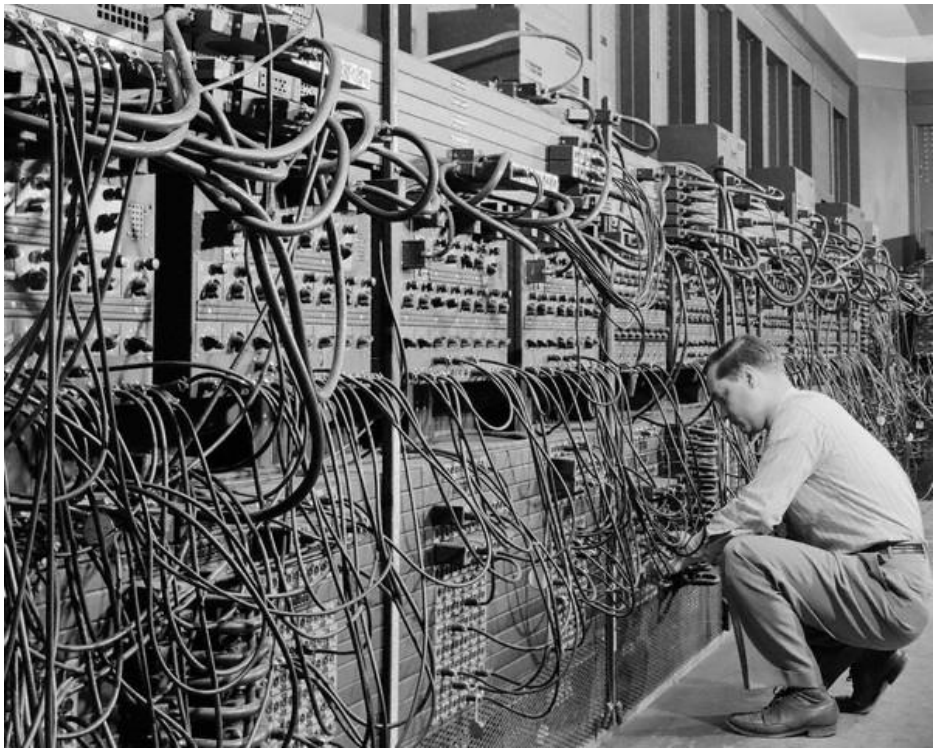


ENIAC's backside

ENIAC's vacuum tubes

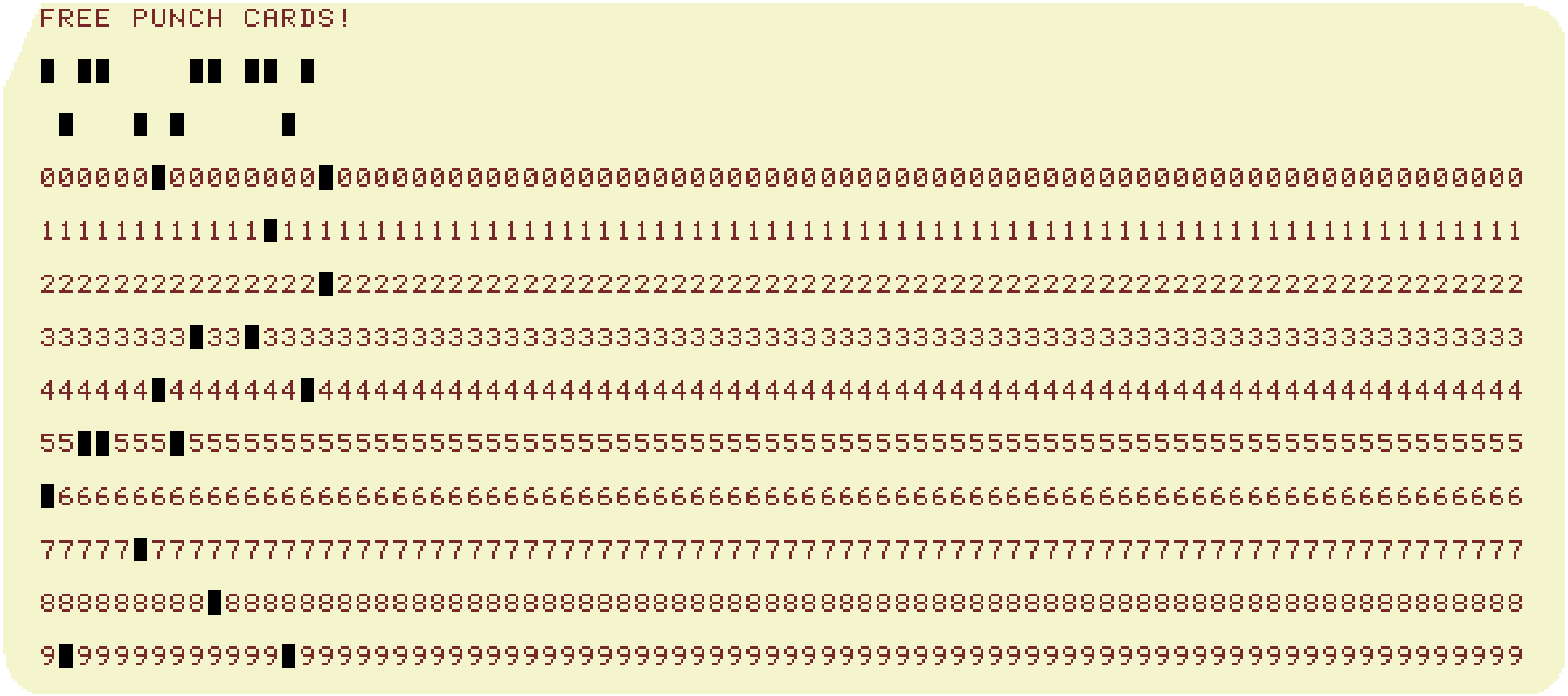
1st Generation Computing (4)

Programs were loaded into memory manually using switches, punched cards, or paper tapes.

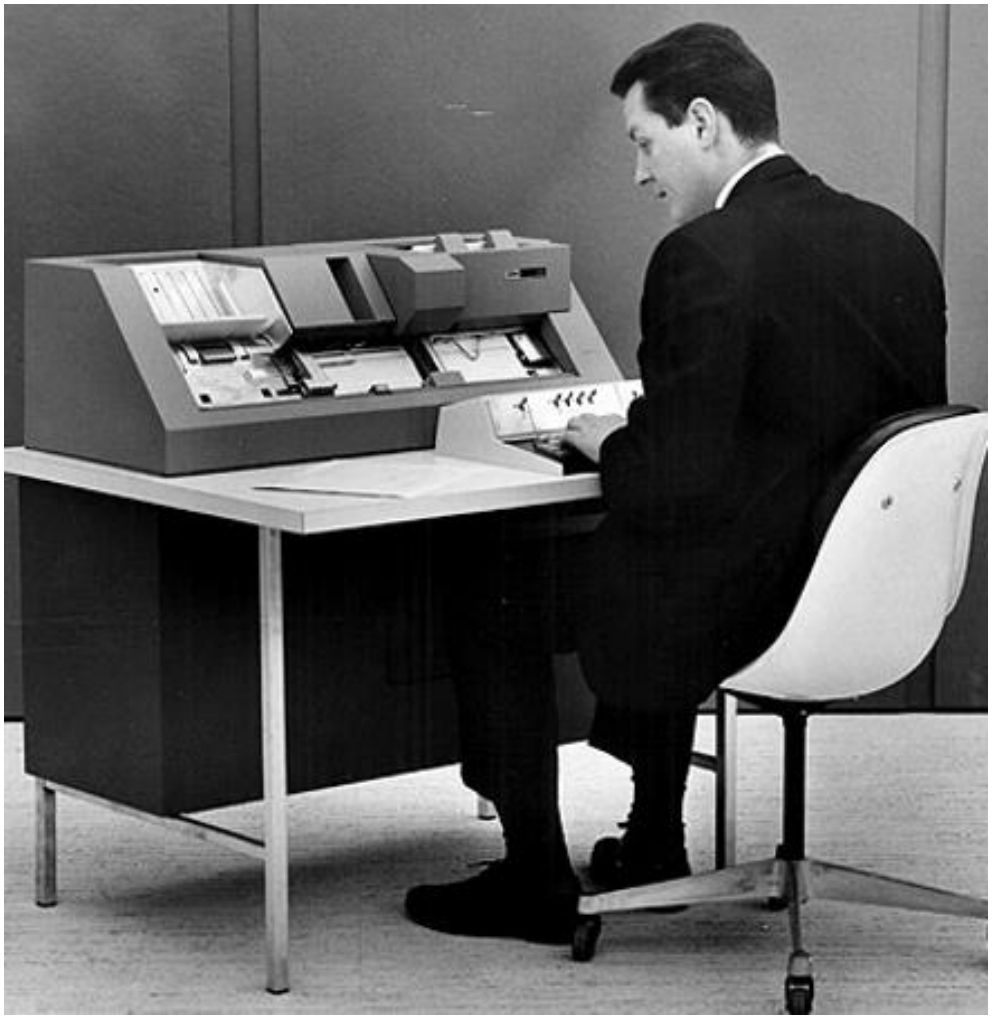


ENIAC : coding by cable connections

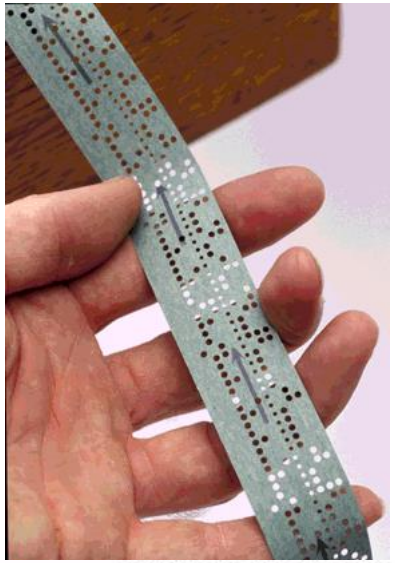
2nd Generation Computing (1)

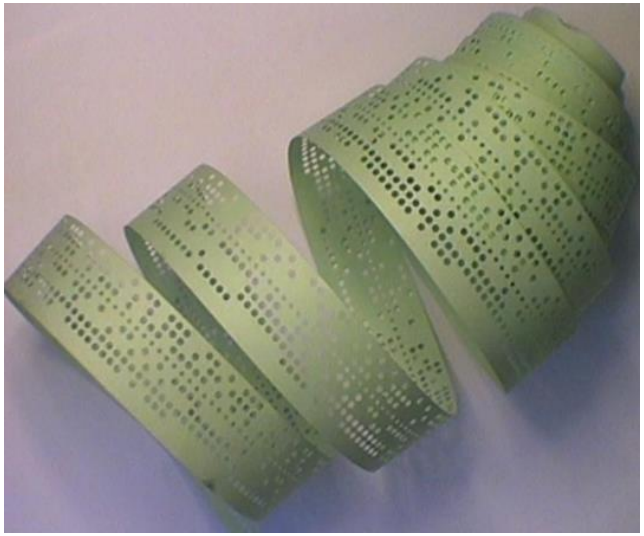


2nd Generation Computing (2)

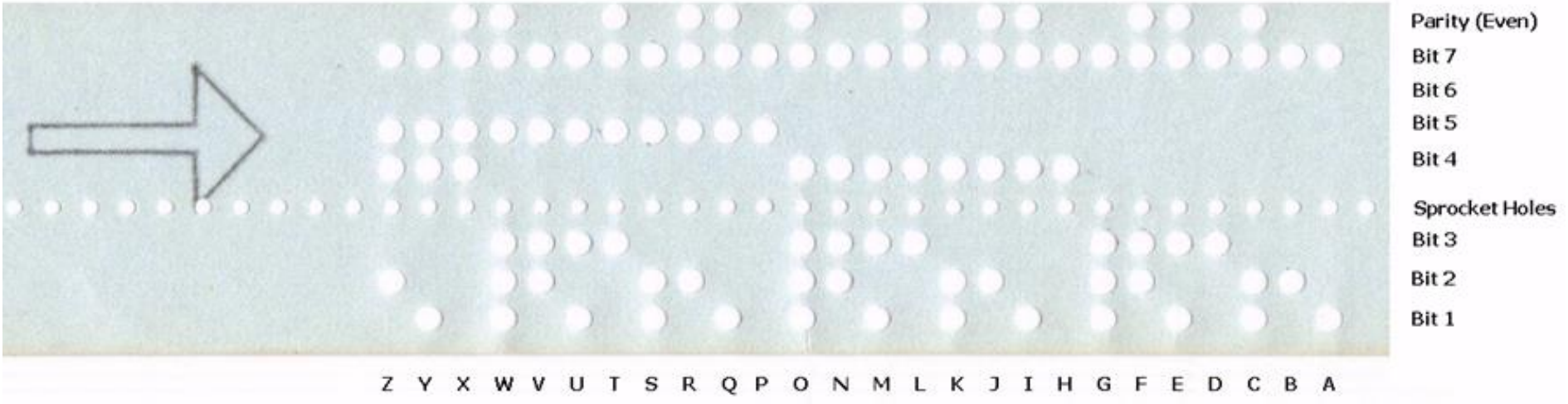


2nd Generation Computing (3)

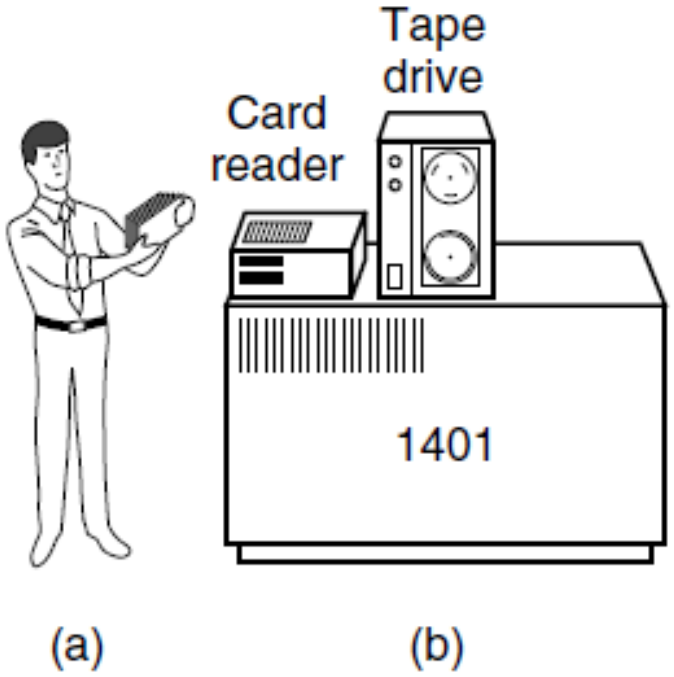




Punched Paper Tape 25.4 mm wide. Ascii 7-bit character code. Even Parity.

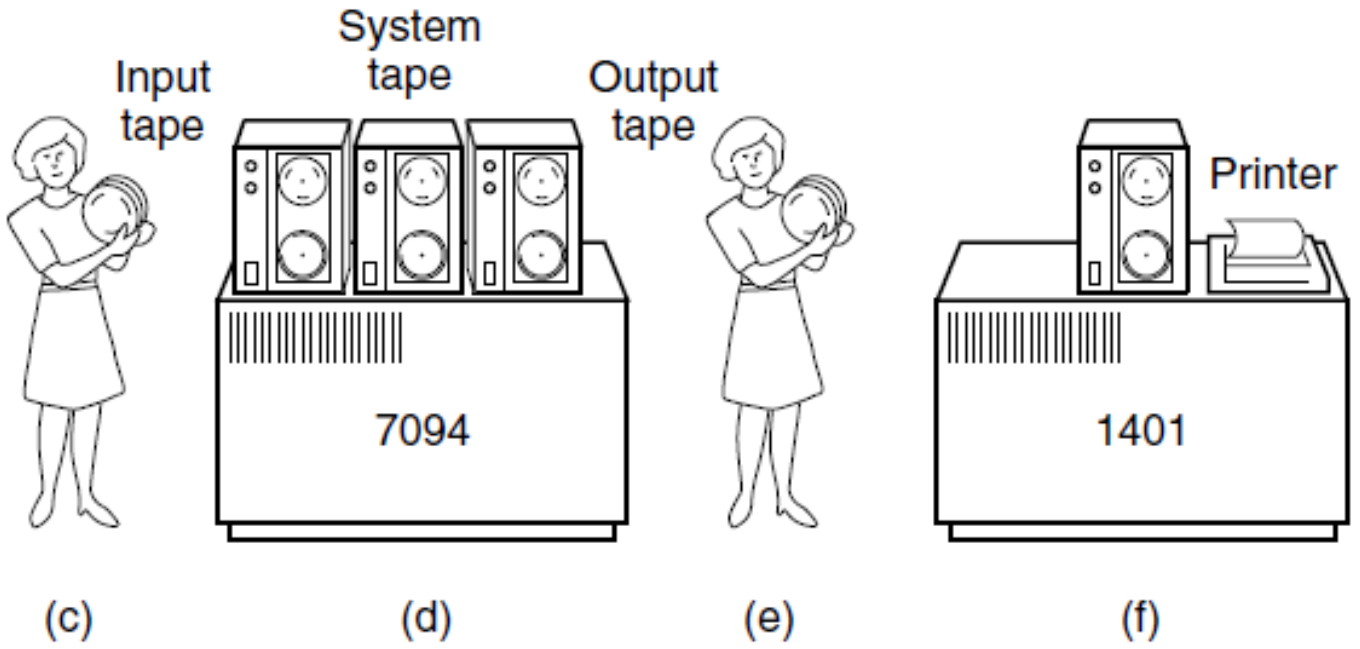


Transistors and Batch Systems (1)



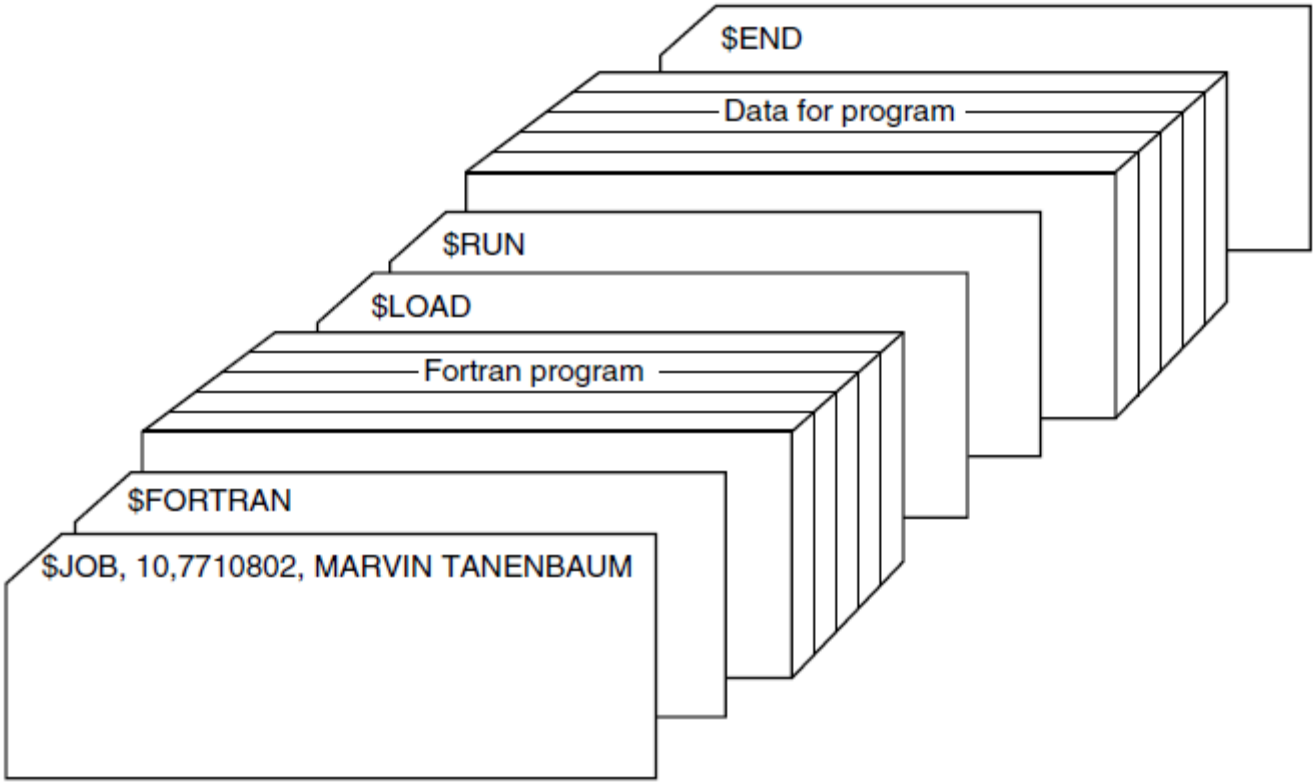
An early batch system. (a) Programmers bring cards to 1401. (b) 1401 reads batch of jobs onto tape.

Transistors and Batch Systems (2)



An early batch system. (c) Operator carries input tape to 7094. (d)7094 does computing. (e) Operator carries output tape to 1401. (f) 1401 prints output.

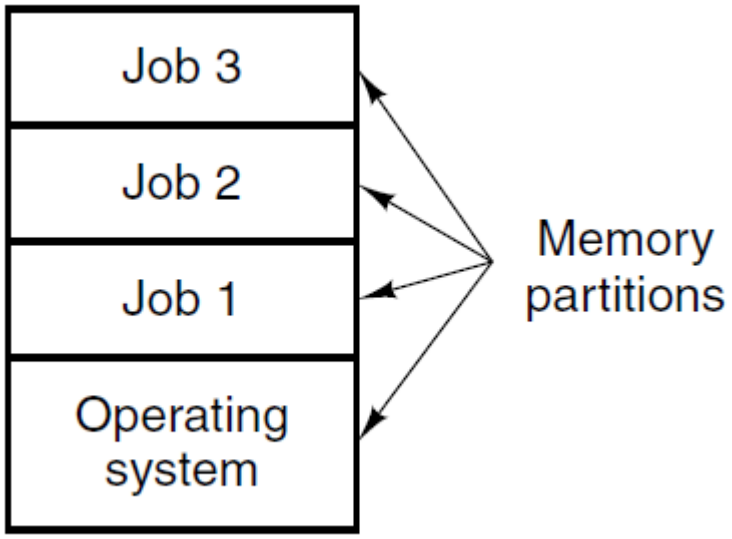
Transistors and Batch Systems (3)



Structure of a typical FMS job.

3rd Generation: ICs and Multiprogramming

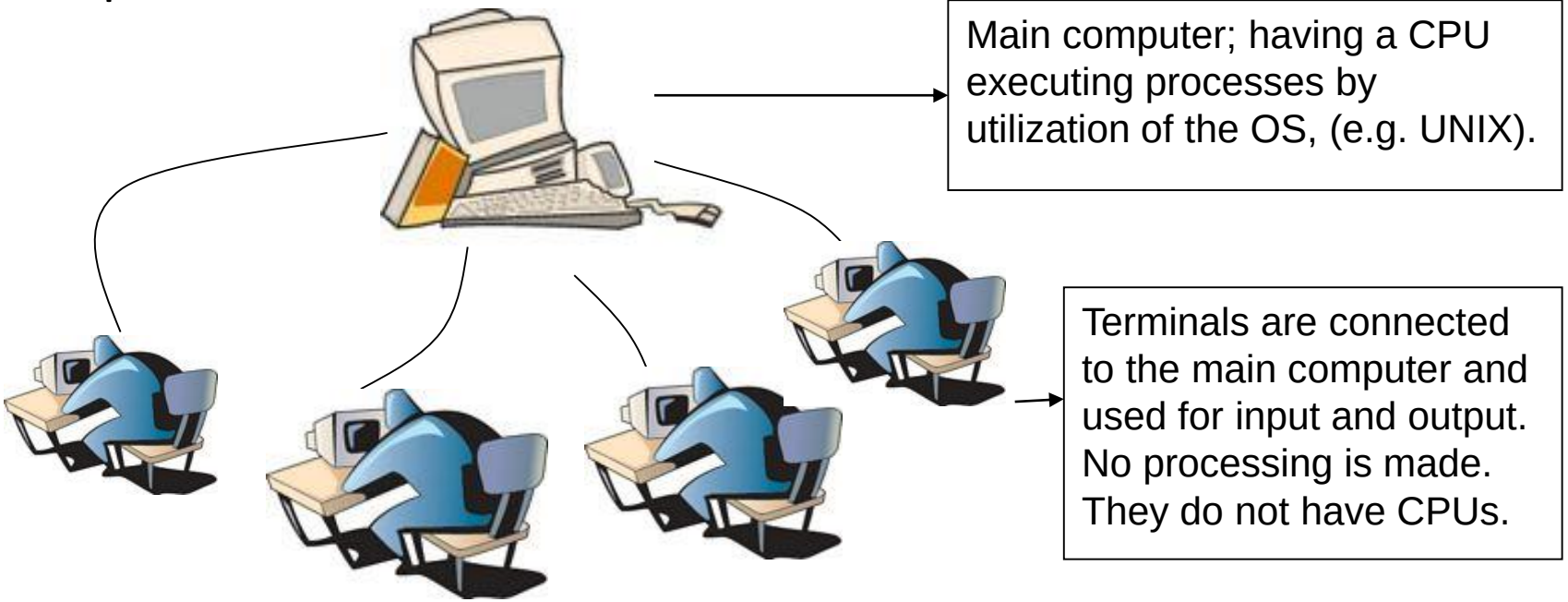
- Finally, the idea of **multiprogramming** came.
- Multiprogramming means sharing of resources between more than one processes.
- By multiprogramming the CPU time is not wasted, because, while one process moves on some I/O work, the OS picks another process to execute till the current one passes to I/O operation.



A multiprogramming system with three jobs in memory.

ICs and Multiprogramming

- With the development of interactive computation in 1970s, **time-sharing systems** emerged.
- In these systems, multiple users have *terminals* (not computers) connected to a *main computer* and execute their tasks in the main computer.



3rd Generation OSs

- Computing as a utility (like electricity) (idea by GE)
  - GE-645 mainframe
  - **MULTICS** (MULTiplexed Information and Computing Service)
- MULTICS had a huge influence on subsequent operating systems (especially UNIX and its derivatives, FreeBSD, Linux, iOS, and Android).
- A stripped-down, one-user version of MULTICS led to **UNIX**
- Two major UNIX versions followed: System V, from AT&T, and BSD from the University of California at Berkeley.
- To make it possible to write programs that could run on any UNIX system, IEEE developed a standard for UNIX, called **POSIX**, that defines a minimal system-call interface

4th Generation OSs

- Minicomputers -> microcomputers (personal computers)
  - Intel 8080, the first general-purpose 8-bit CPU
  - First a disk-based operating system named CP/M (Control Program for Microcomputers)
- IBM PC market
  - DOS (Disk Operating System) (Bill Gates)
  - MS-DOS (MicroSoft Disk Operating System)
- Apple PC market
  - MAC OS X (Unix and GUI based)
- Microsoft
  - Windows, Windows 95, Windows 98, Windows NT, Windows 2000, Windows XP, Windows 2007, ... (GUI based)

4th Generation OSs

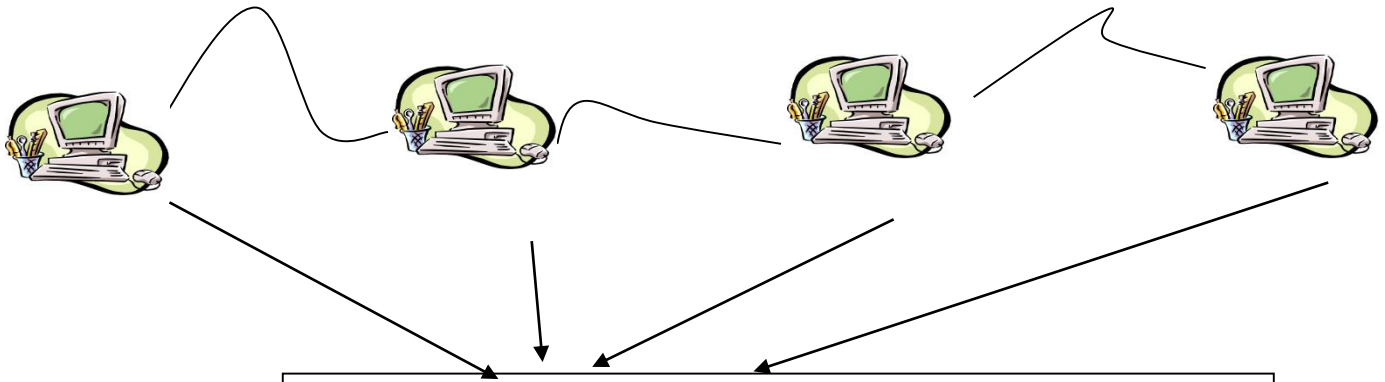
- The other major contender in the PC world is UNIX (and its derivatives) such as FreeBSD

- X Window (X11) (basic window functions)
- Gnome or KDE (complete GUI running on top of X11)

- MINIX (a small unix clone for education purposes)
- Linux (free production version of MINIX)

4th Generation OSs

- Use of the networks required network OSs.
- In network systems, each process runs in its own machine but the OS have access to other machines.
- By this way, file sharing, messaging, etc. became possible.
- In networks, users are not aware of the fact that they are working in a network and when information is exchanged. The user explicitly handles the transfer of information.



4th Generation OSs

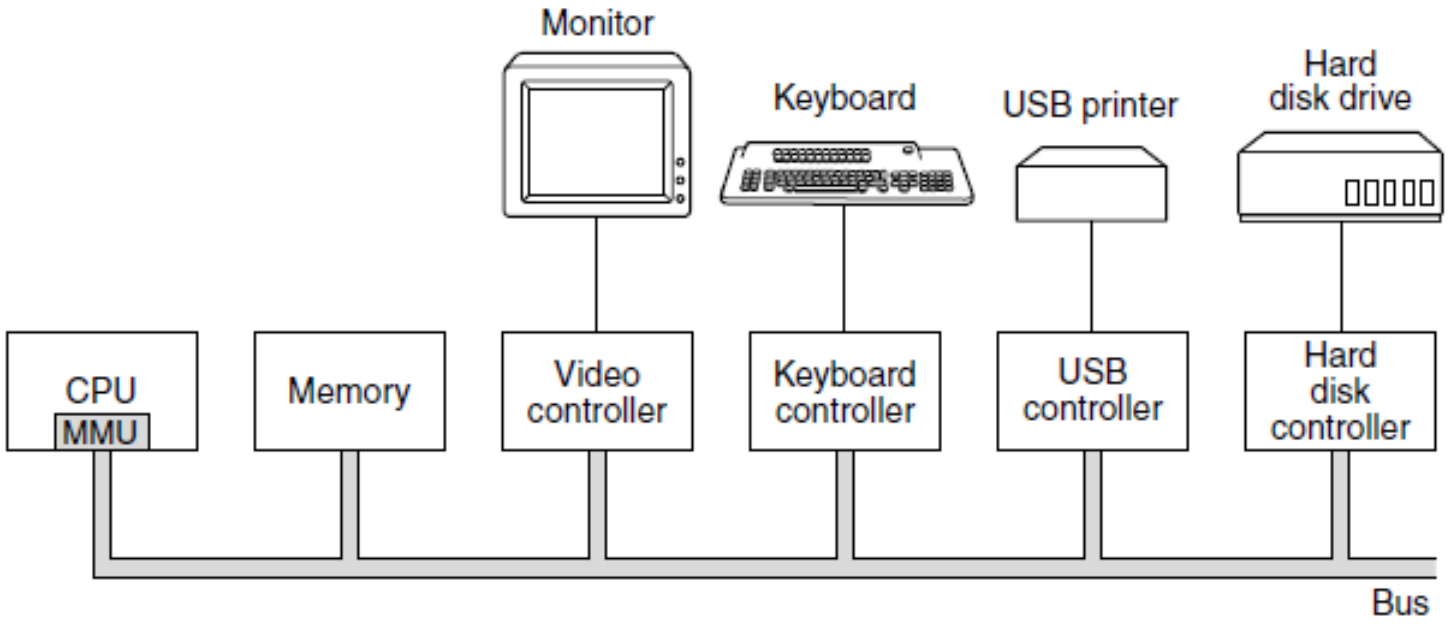
- Distributed systems are similar to networks.
- However in such systems, there is no need to exchange information explicitly, it is handled by the OS itself whenever necessary.
- A distributed OS is one that appears to its users as a traditional uniprocessor system, even though it is actually composed of multiple processors.
- The users are not aware where their programs are being run or where their files are located; that should all be handled automatically and efficiently by the operating system.

5th Generation OSs

- A mobile phone + PDA (Personal Digital Assistant) = smartphone

- Symbian OS (earlier models of Samsung, Sony Ericsson, Motorola, and Nokia)
- RIM's Blackberry OS (introduced in 2002)
- Apple's iOS (released in 2007)
- Nokia Windows Phone
- Google's Android (a Linux-based OS)
    - open source and available under a permissive license.
    - huge community of developers

Processors (1)



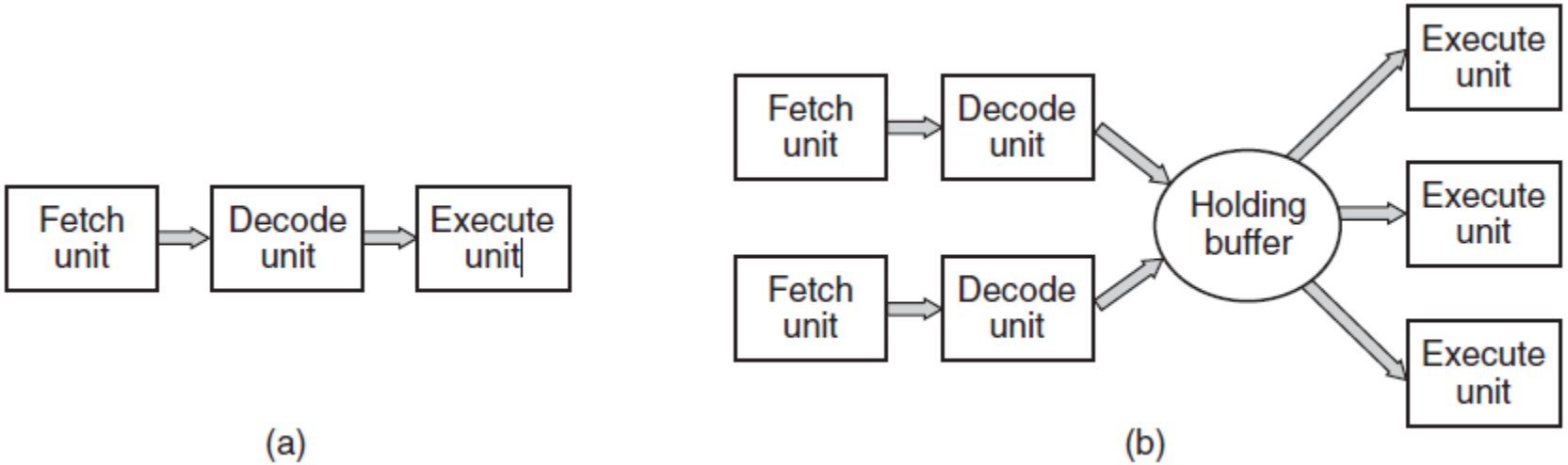
Some of the components of a simple personal computer.

Processors (2)

- Pipelined architecture
- Superscalar architecture
- Multithreading (hyperthreading) architecture
- Multicore architecture

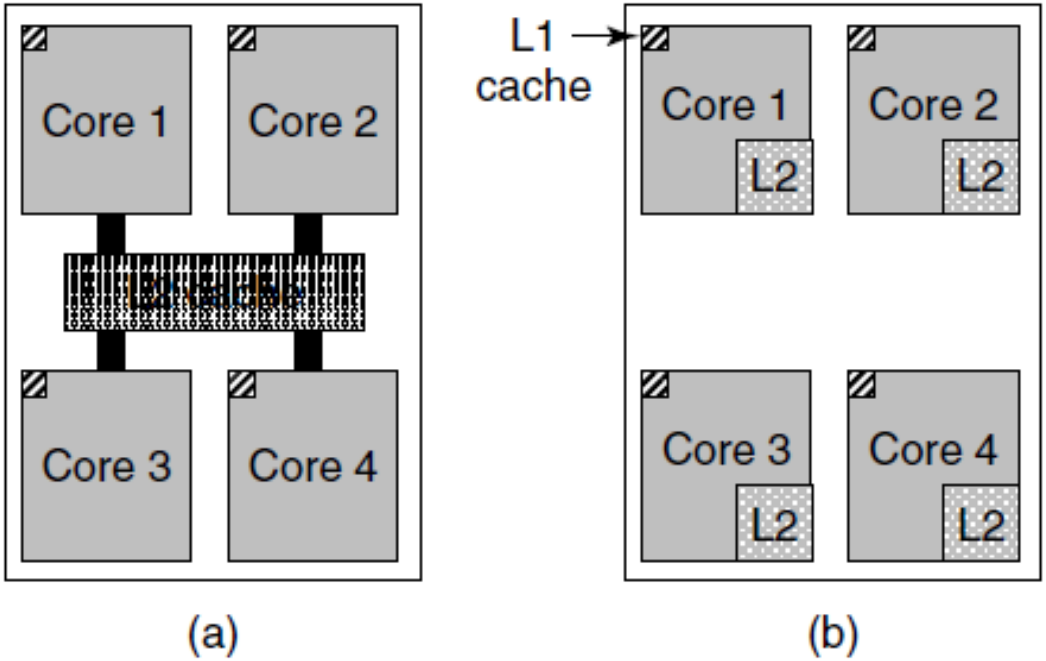
All have implications and complications on OS.

Processors (3)



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

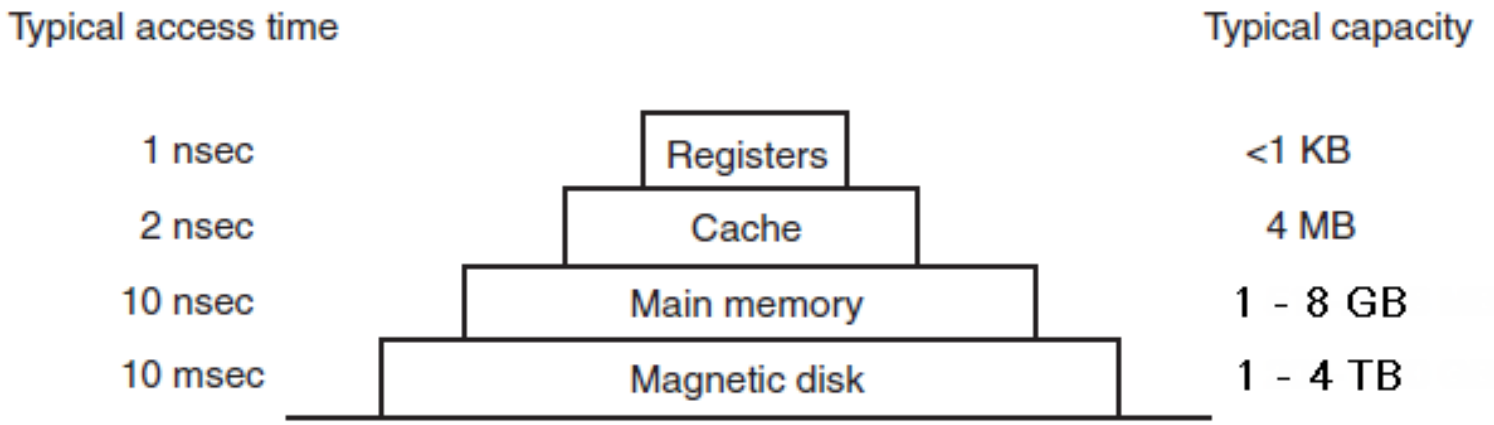
Processors (4)



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

Memory (1)

Ideally, memory should be extremely fast, abundantly large, and very cheap.



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

Memory (2)

Whenever a resource can be divided into pieces, some of which are used much more heavily than others, caching is often used to improve performance.

Caching system issues:

1. When to put a new item into the cache.
2. Which cache line to put the new item in.
3. Which item to remove from the cache when a slot is needed.
4. Where to put a newly evicted item in the larger memory.

Memory (3)

L1 cache is always inside the CPU and usually feeds decoded instructions into the CPU's execution engine (typically 16 KB).

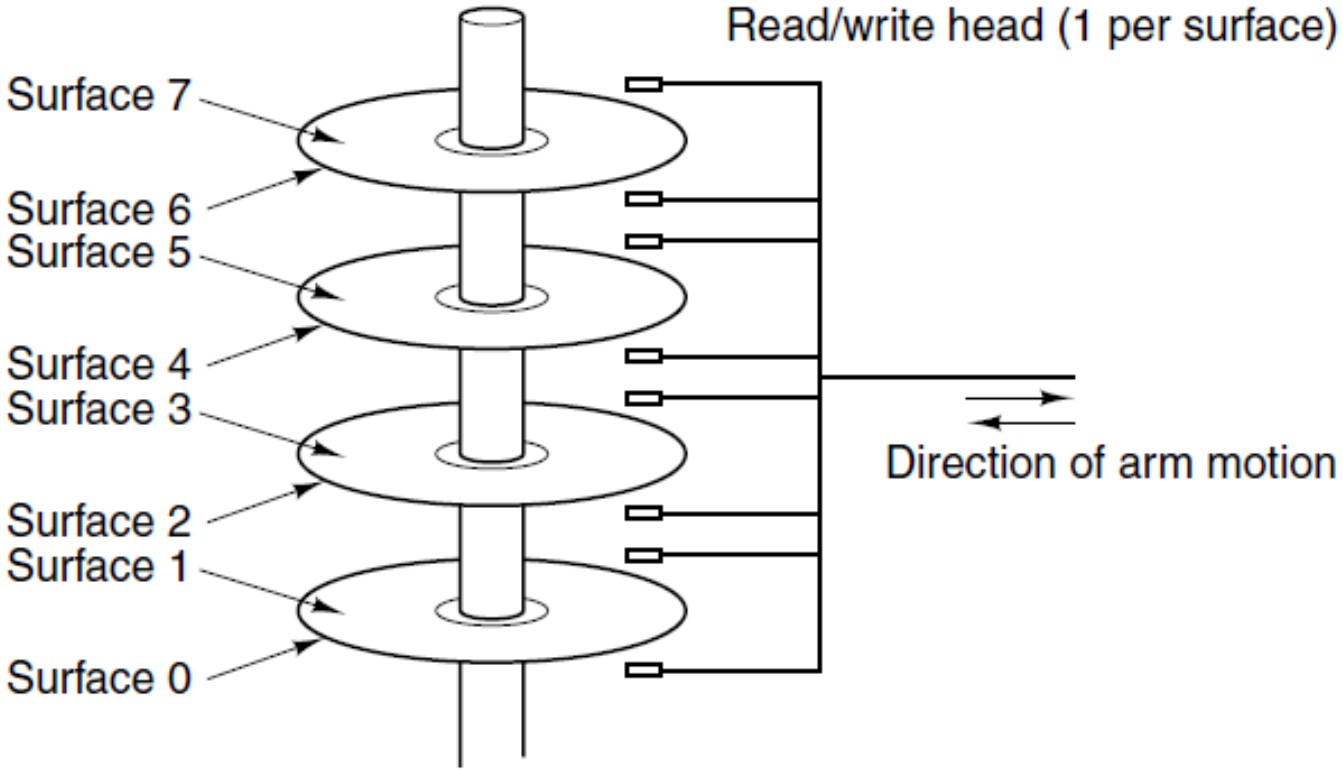
L2 cache holds several megabytes of recently used memory words.

L1 cache no delay, L2 cache delay of one or two clock cycles.

Storage types:

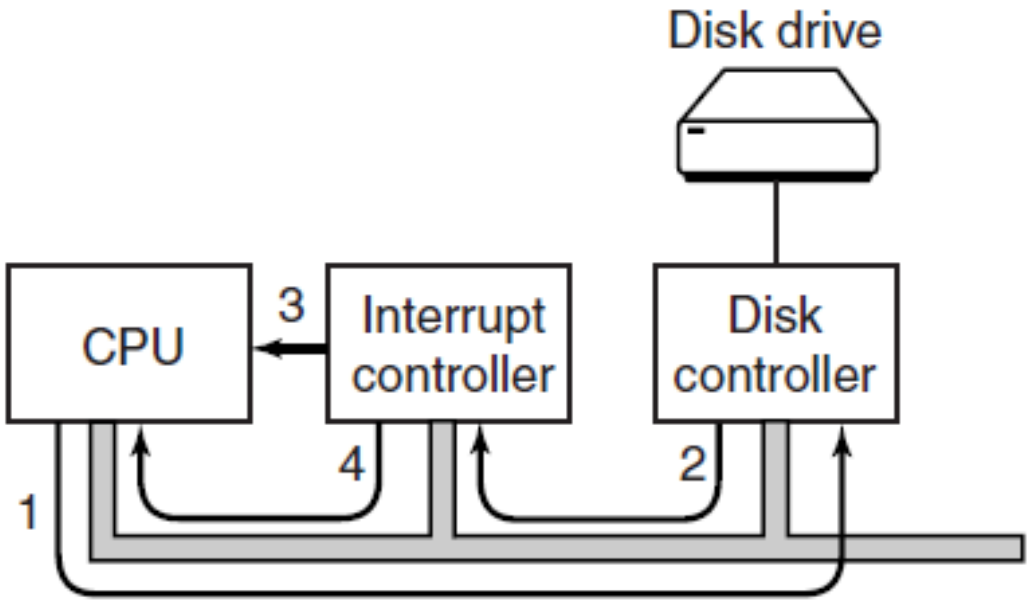
- RAM
- ROM
- EEPROM
- Flash
- Magnetic (Hard) Disk

Disks



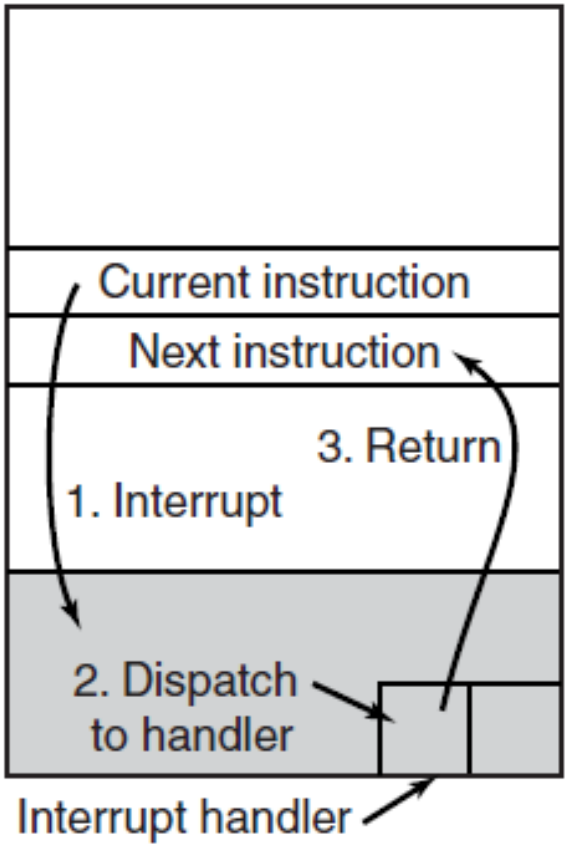
Structure of a disk drive.

I/O Devices



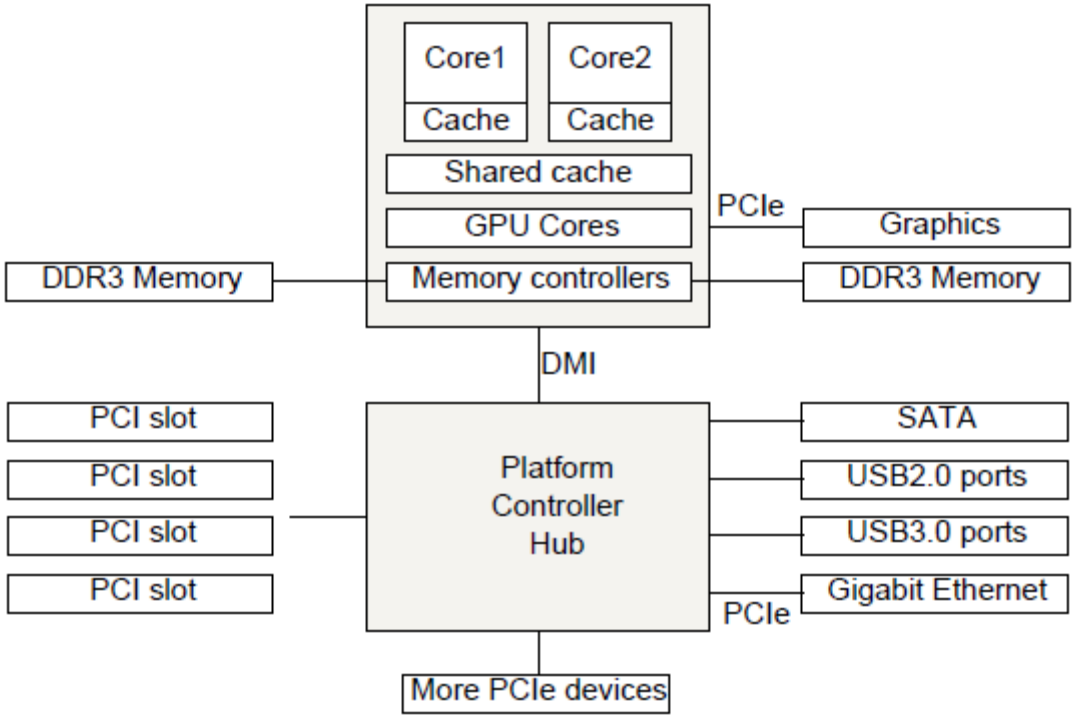
(a) The steps in starting an I/O device and getting an interrupt.

I/O Devices



(b) Interrupt processing involves taking the interrupt, running the interrupt handler, and returning to the user program.

Buses



The structure of a large x86 system

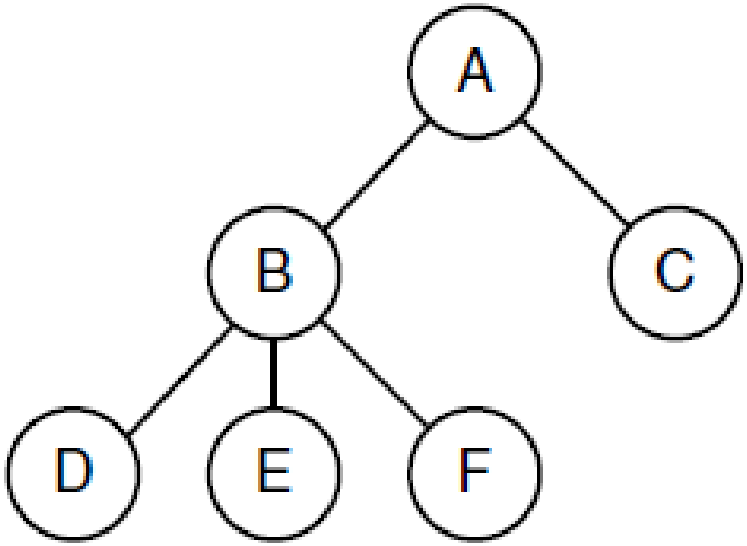
The Operating System Zoo

- Mainframe Operating Systems
- Server Operating Systems
- Multiprocessor Operating Systems
- Personal Computer Operating Systems
- Handheld Computer Operating Systems
- Embedded Operating Systems
- Sensor Node Operating Systems
- Real-Time Operating Systems
- Smart Card Operating Systems

Processes (1)

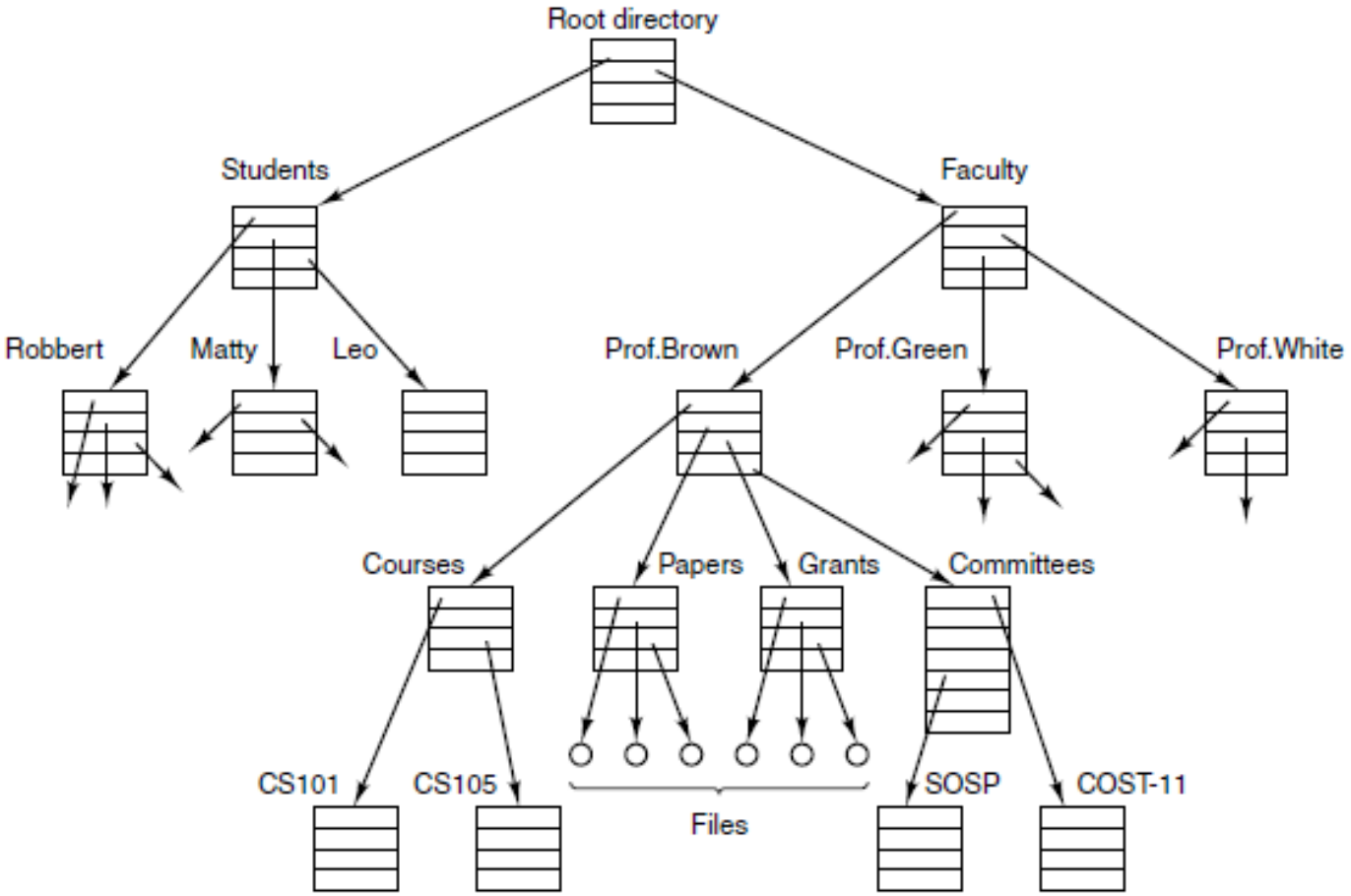
- Key concept in all operating systems
- Definition: a program in execution
- Process is associated with an address space
- Also associated with set of resources
- Process can be thought of as a container
  - Holds all information needed to run program

Processes (2)



A process tree. Process A created two child processes, B and C. Process B created three child processes, D, E, and F.

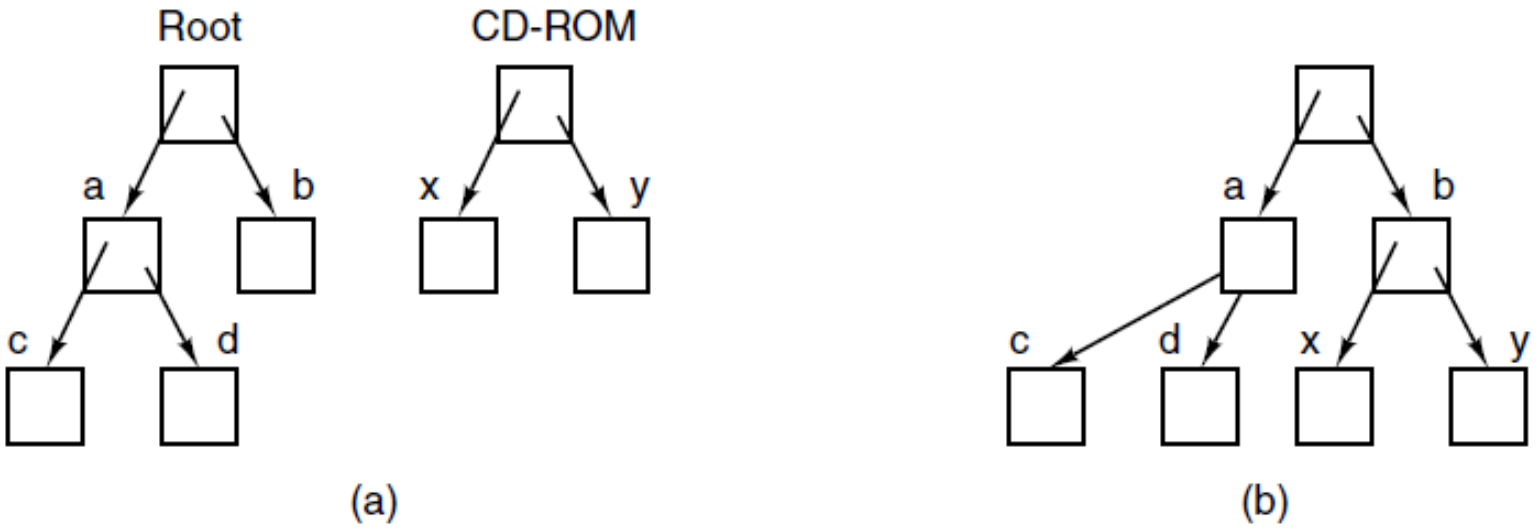
Files (1)



A file system for a university department.

Files (2)

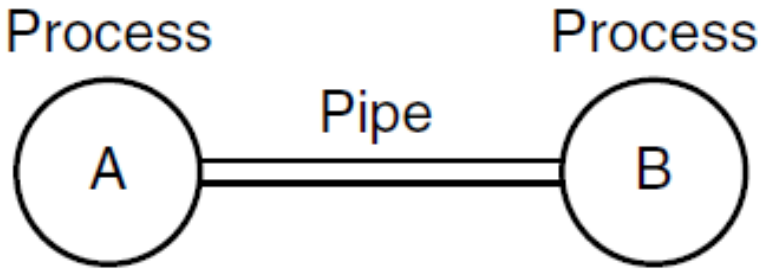
UNIX mount operation



(a) Before mounting, the files on the CD-ROM are not accessible.
(b) After mounting, they are part of the file hierarchy.

Files (3)

- A pipe is a sort of pseudo file that can be used to connect two processes.
- Communication between processes in UNIX looks very much like ordinary file reads and writes.



Two processes connected by a pipe.

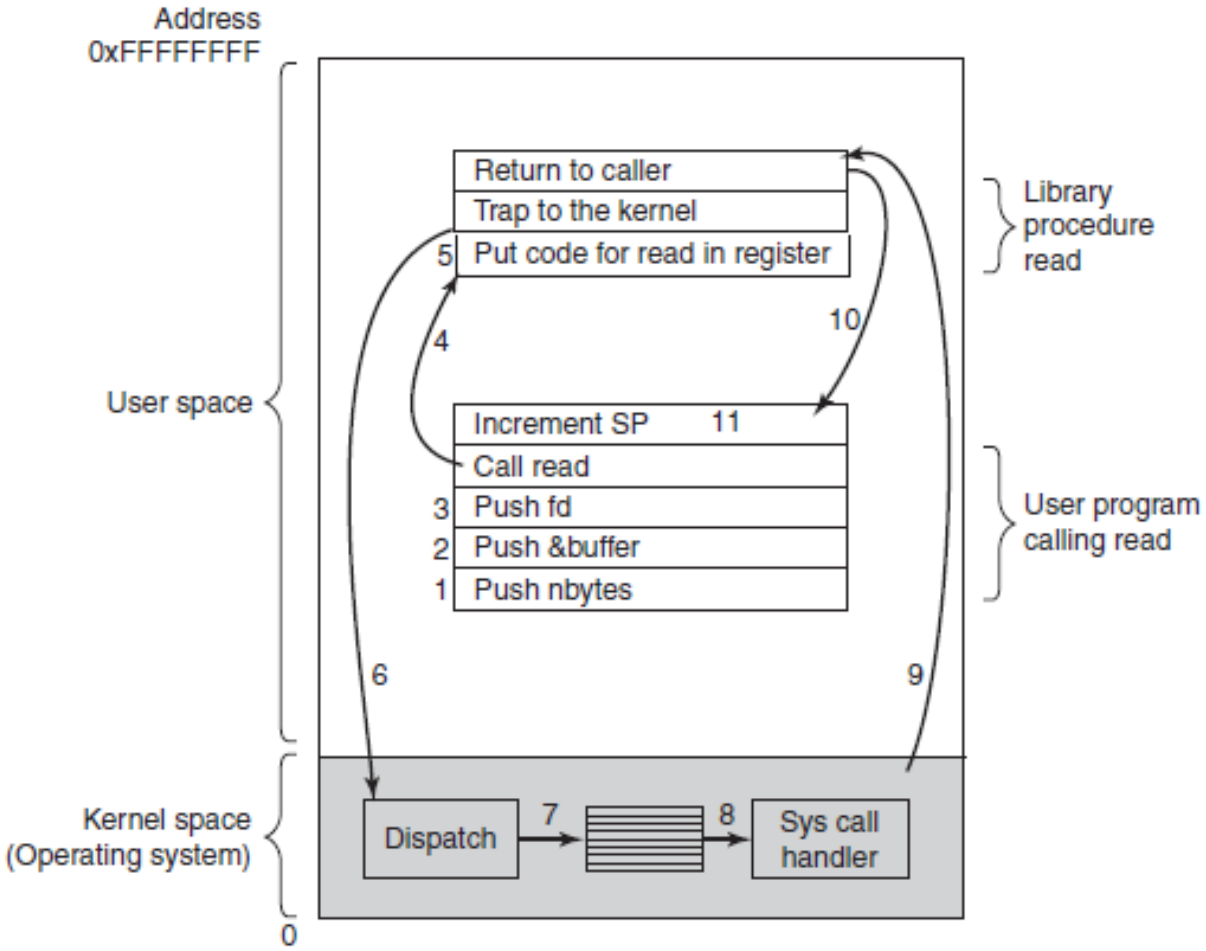
Ontogeny Recapitulates Phylogeny

- Each new “species” of computer
  - Goes through same development as “ancestors”
- Consequence of impermanence
  - Text often looks at “obsolete” concepts
  - Changes in technology may bring them back
- Happens with large memory, protection hardware, disks, virtual memory

System Calls (1)

An example call from a C program:

count = read(fd, buffer, nbytes);



11 steps in making the system call

System Calls (2)

- ~ 100 POSIX procedure calls

- The return code *s* is −1 if an error has occurred.
- The return codes are as follows:
  - *pid* is a process id, *fd* is a file descriptor, *n* is a byte count, *position* is an offset within the file, and *seconds* is the elapsed time.

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

System Calls for Process Management

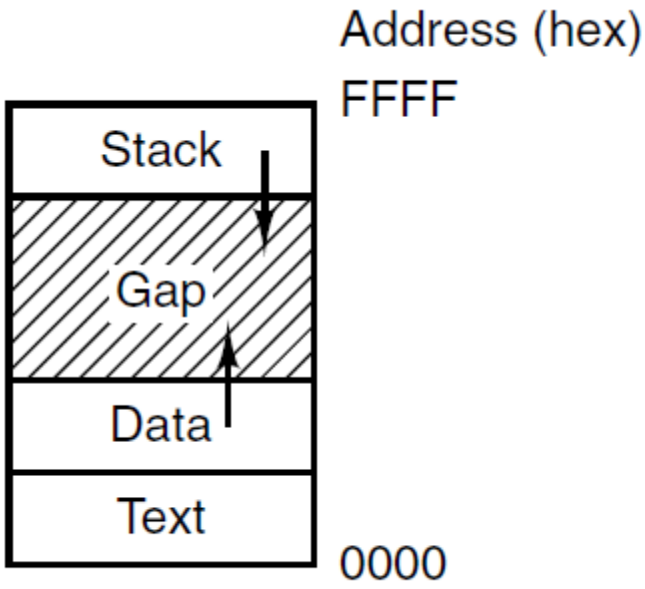
```
#define TRUE 1

while (TRUE) {                                  /* repeat forever */
        type_prompt( );                         /* display prompt on the screen */
        read_command(command, parameters);      /* read 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 */
        }
}
```

A stripped-down shell

System Calls for Process Management (2)



Processes have three segments:
text, data, and stack

System Calls (3)

File management

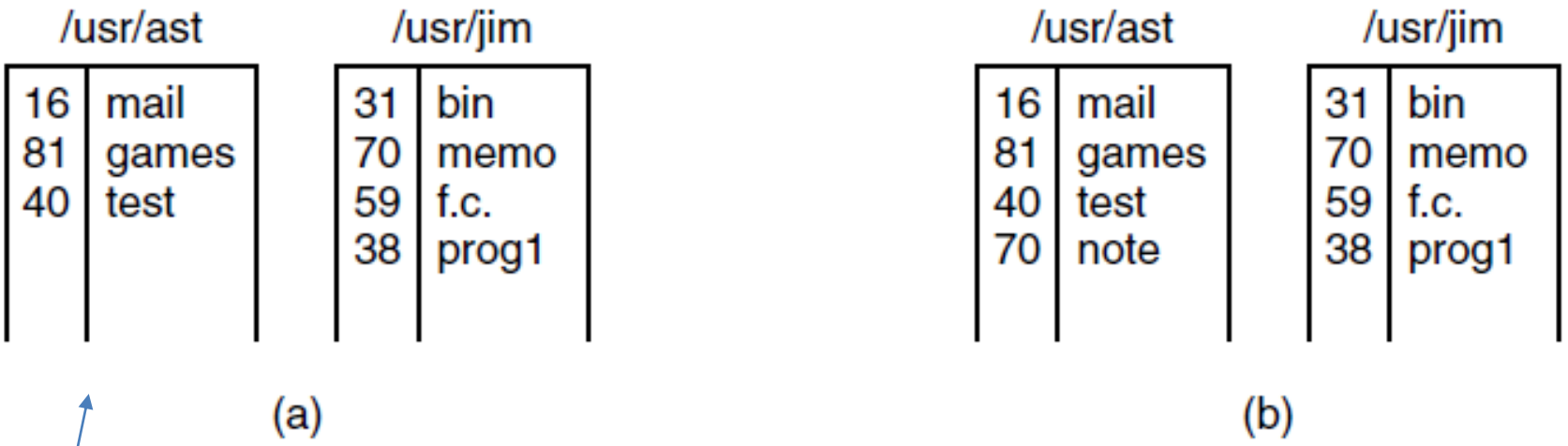
Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file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

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 for Directory Management (1)

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



(a) Two directories before linking *usr/jim/memo* to *ast*'s directory. (b) The same directories after linking.

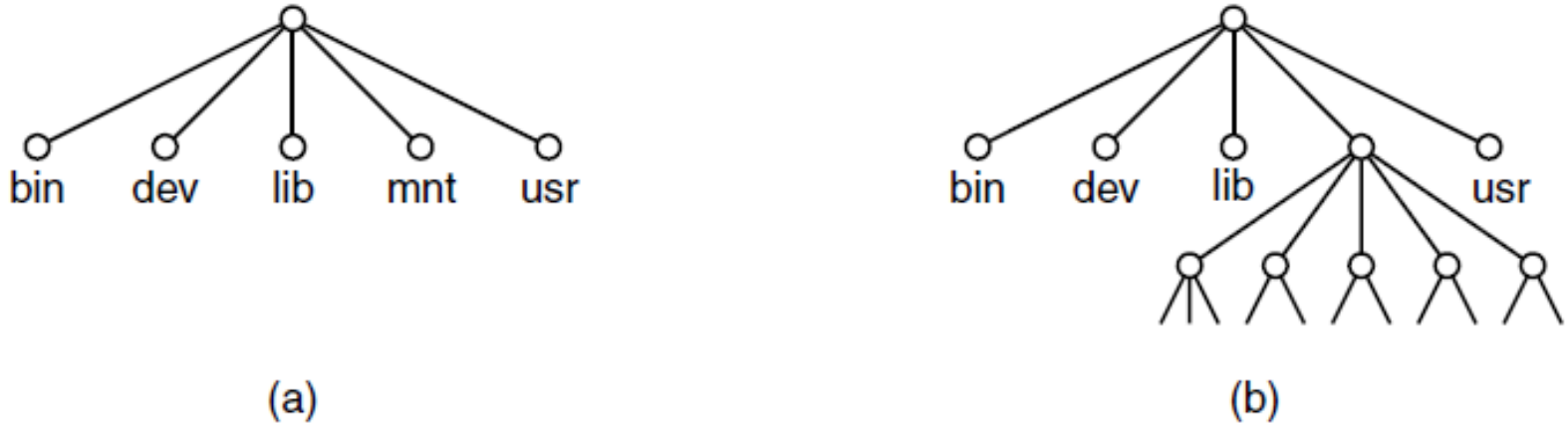
Directory is a set of (i_number, ascii name)

System Calls for Directory Management (2)

A typical statement in C to mount a USB disk to be read:

mount("/dev/sdb0", "/mnt", 0);

name of a block special file for USB drive 0



(a) File system before the mount. (b) File system after the mount.

System Calls (4)

Miscellaneous

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

The Windows Win32 API (1)

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

Win32 API calls that roughly correspond to some UNIX calls earlier

The number of Win32 API calls is extremely large, numbering in the thousands.

The Windows Win32 API (2)

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

Win32 API calls that roughly correspond to some UNIX calls earlier

OS Structure: Monolithic Systems (1)

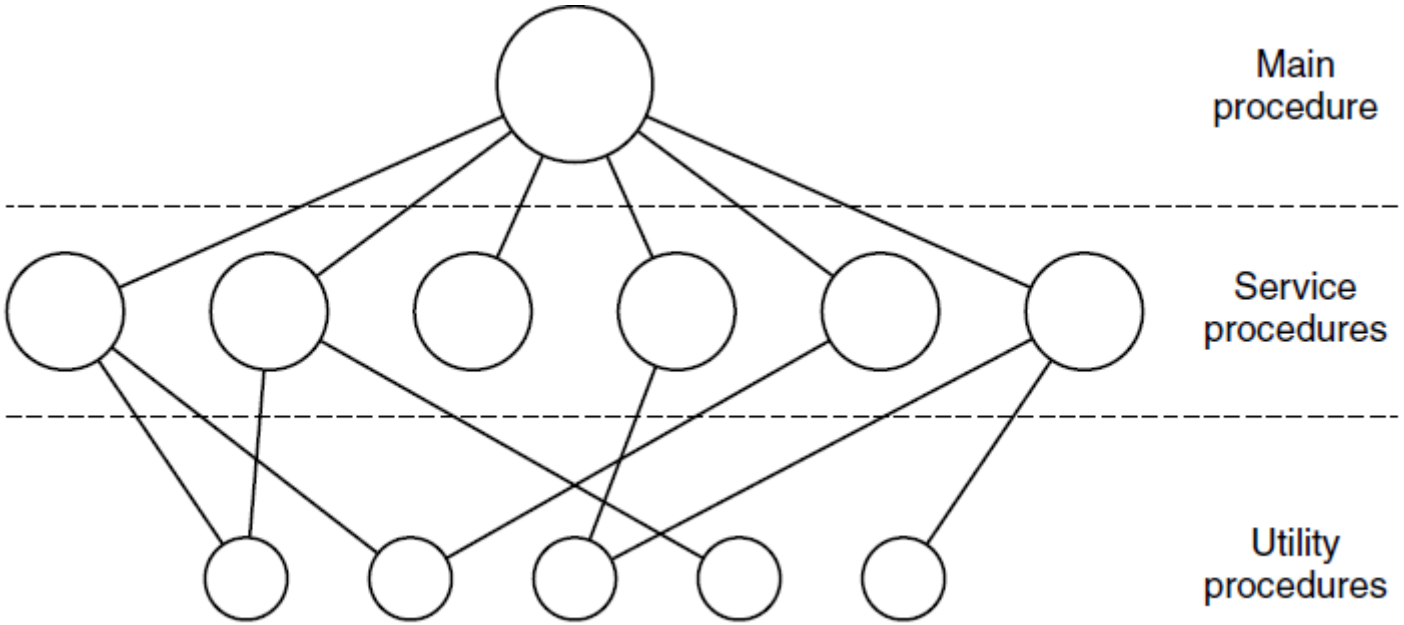
- In monolithic approach the entire OS runs as a single program in kernel mode.
- It is written as a collection of procedures, linked together into a single large executable binary program.
- Each procedure is free to call any other one – hence very efficient but difficult to understand and unreliable.

OS Structure: Monolithic Systems (2)

Basic structure of OS

1. A main program that invokes the requested service procedure.
2. A set of service procedures that carry out the system calls.
3. A set of utility procedures that help the service procedures.

OS Structure: Monolithic Systems (3)



A simple structuring model for a monolithic system.

- Loading on-demand extensions, such as I/O device drivers and file systems may also be supported such as in UNIX (shared libraries) and in Windows (DLLs).

OS Structure: Layered Systems

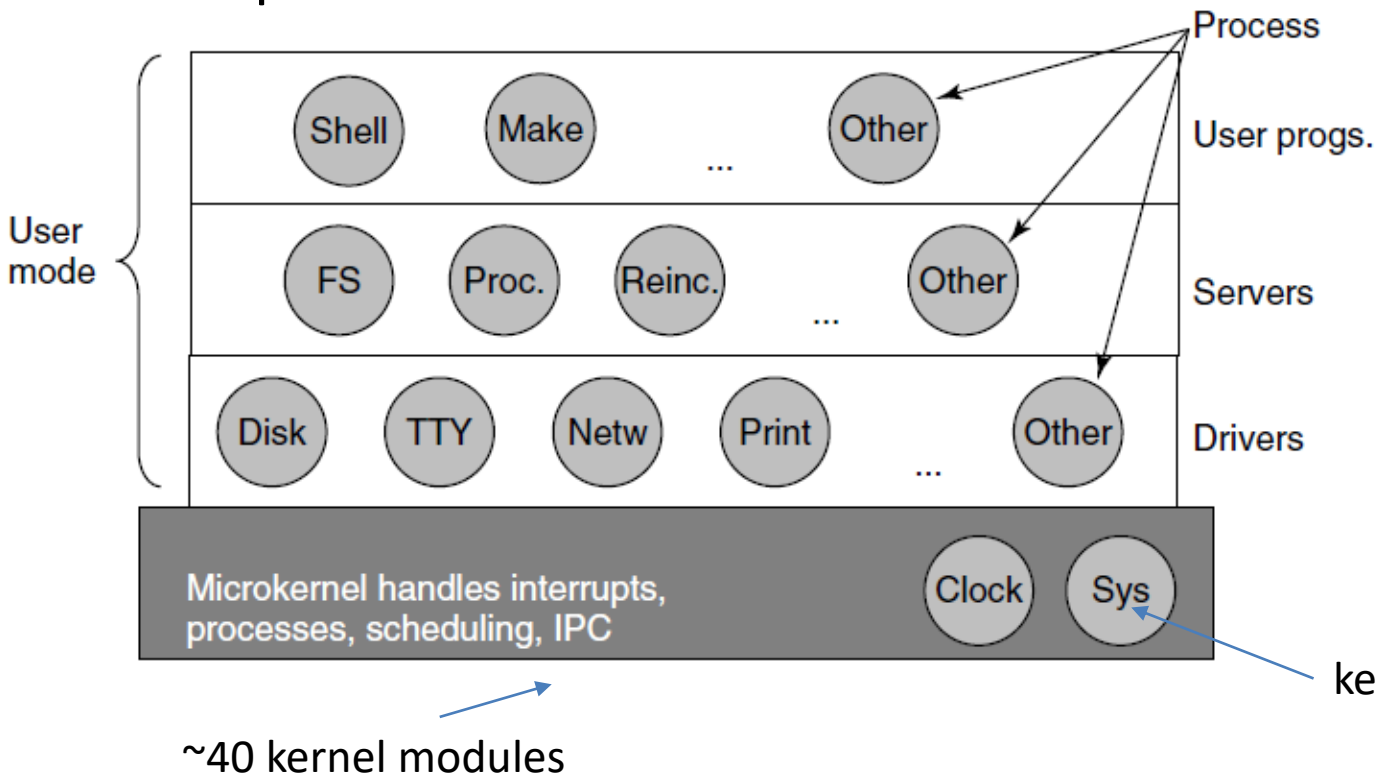
Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Structure of THE (Technische Hogeschool Eindhoven by E. W. Dijkstra (1968)) operating system.

OS Structure: Microkernels

Idea behind microkernel design is

- to achieve high reliability by splitting the OS into small, well-defined modules
- only one of which—the microkernel—runs in kernel mode and the rest run as relatively powerless ordinary user processes.



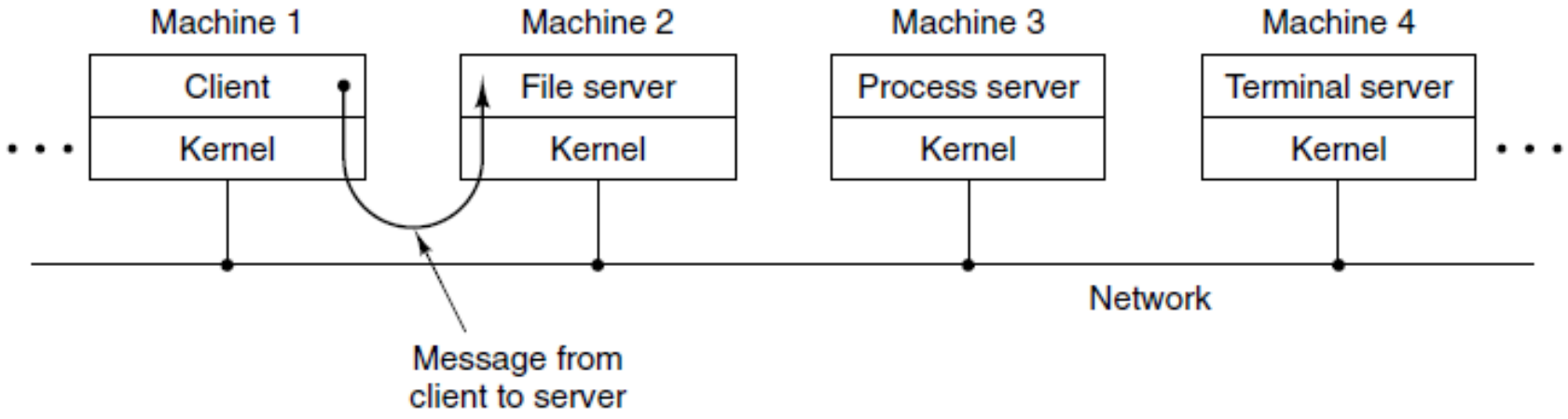
Simplified structure of the MINIX 3 system.

OS Structure: Client-Server Model

Idea is to distinguish two classes of processes,

- the servers, each of which provides some service, and
- the clients, which use these services.

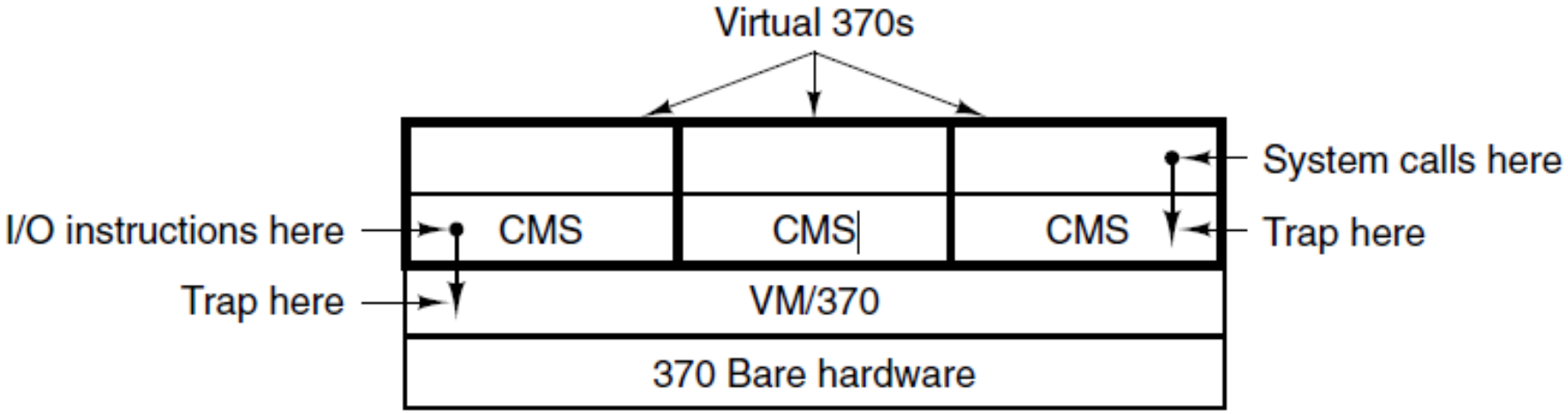
The model is an abstraction that can be used for a single machine or for a network of machines.



The client-server model over a network.

OS Structure: Virtual Machines

- **Virtual machine monitor**, runs on bare hardware and does the multiprogramming, providing not one, but several virtual machines to the next layer up.
- However, these virtual machines are not extended machines, with files and other nice features.
- Instead, they are exact copies of the bare hardware, including kernel/user mode, I/O, interrupts, and everything else the real machine has.



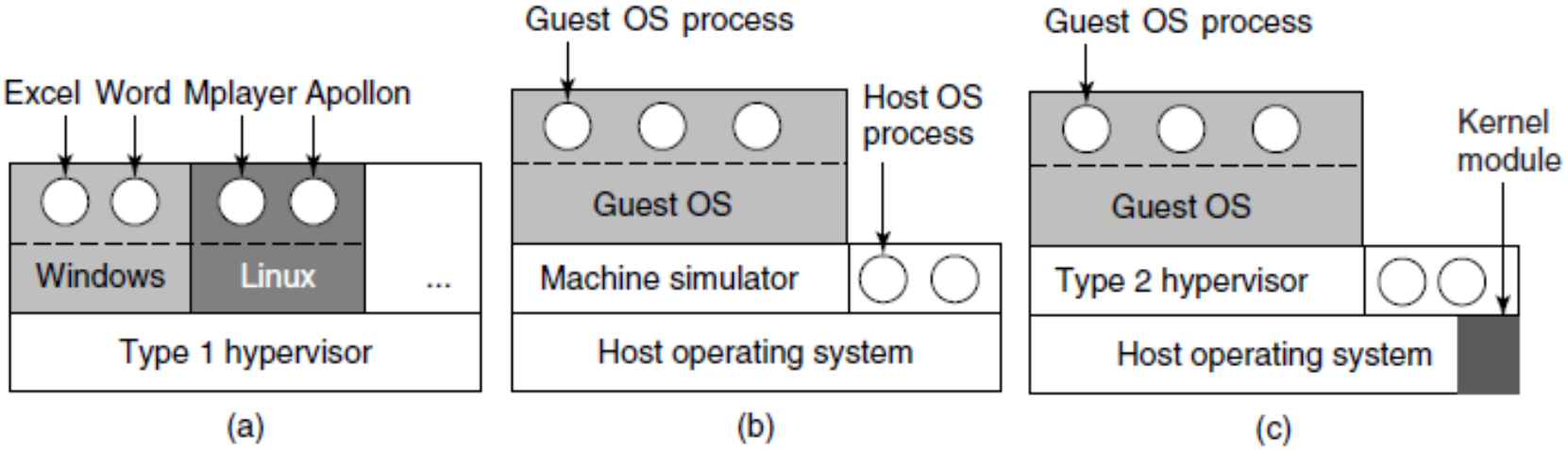
The structure of VM/370 with CMS.

OS Strct.: Virtual Machines Rediscovered

- New needs+new software+new technologies -> virtualization (hot topic)

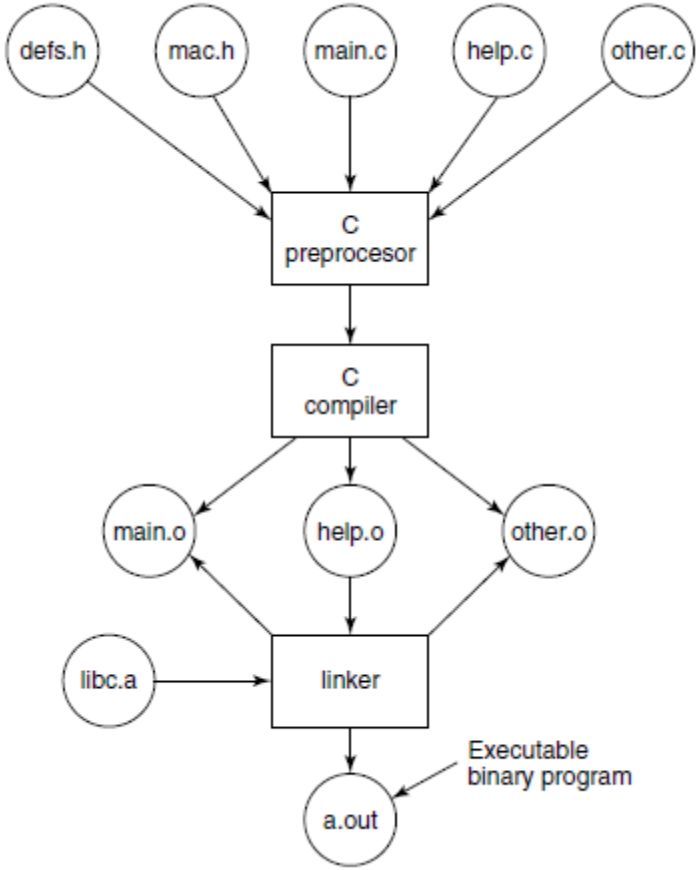
Example use:

- A company runs mail servers, Web servers, FTP servers, and other servers on separate computers, with different operating systems.
- However, virtualization, enables them to run them all on the same machine without having a crash of one server bringing down the rest.



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

Large Programming Projects



The process of compiling C and header files to make an executable.

Metric Units

Exp.	Explicit	Prefix	Exp.	Explicit	Prefix
10^{-3}	0.001	milli	10^{3}	1,000	Kilo
10^{-6}	0.000001	micro	10^{6}	1,000,000	Mega
10^{-9}	0.000000001	nano	10^{9}	1,000,000,000	Giga
10^{-12}	0.000000000001	pico	10^{12}	1,000,000,000,000	Tera
10^{-15}	0.000000000000001	femto	10^{15}	1,000,000,000,000,000	Peta
10^{-18}	0.000000000000000001	atto	10^{18}	1,000,000,000,000,000,000	Exa
10^{-21}	0.000000000000000000001	zepto	10^{21}	1,000,000,000,000,000,000,000	Zetta
10^{-24}	0.000000000000000000000001	yocto	10^{24}	1,000,000,000,000,000,000,000,000	Yotta

The principal metric prefixes.

Memory Size Units

Units for measuring memory sizes are slightly different :

- kilo means $2^{10} = 1024$ rather than $10^3 = 1000$

Thus

1-KB memory contains 1024 bytes, not 1000 bytes.
1-MB memory contains $2^{20} = 1{,}048{,}576$ bytes
1-GB memory contains $2^{30} = 1{,}073{,}741{,}824$ bytes.

However,

1-Kbps communication line transmits 1000 bits per second
10-Mbps LAN runs at 10,000,000 bits/sec

Summary

OSs can be viewed from two viewpoints:

- resource managers (manage different parts of the system efficiently)
- extended machines (provide users with abstractions that are more convenient to use than the actual machine)
  - processes,
  - address spaces
  - files

Summary

The basic OS concepts on which operating systems are built are

- processes
- memory management
- I/O management
- the file system
- security.

Summary

The heart of any operating system is the set of system calls that it can handle.

- process creation and termination
- reading and writing files
- directory management
- miscellaneous calls.

Operating systems can be structured as

- monolithic system
- a hierarchy of layers
- microkernel
- client-server,
- virtual machine
- exokernel.

End

Chapter 1