1

CHAPTER 1. INTRODUCTION

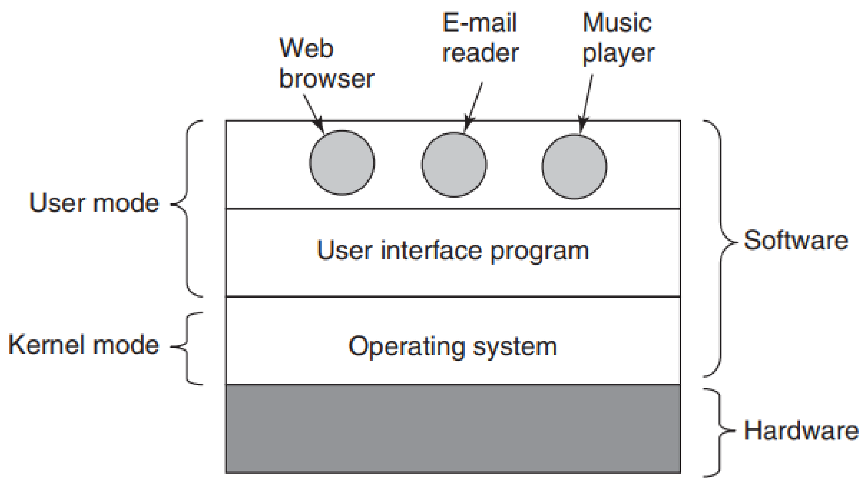


Figure 1-1. Where the operating system fits in.

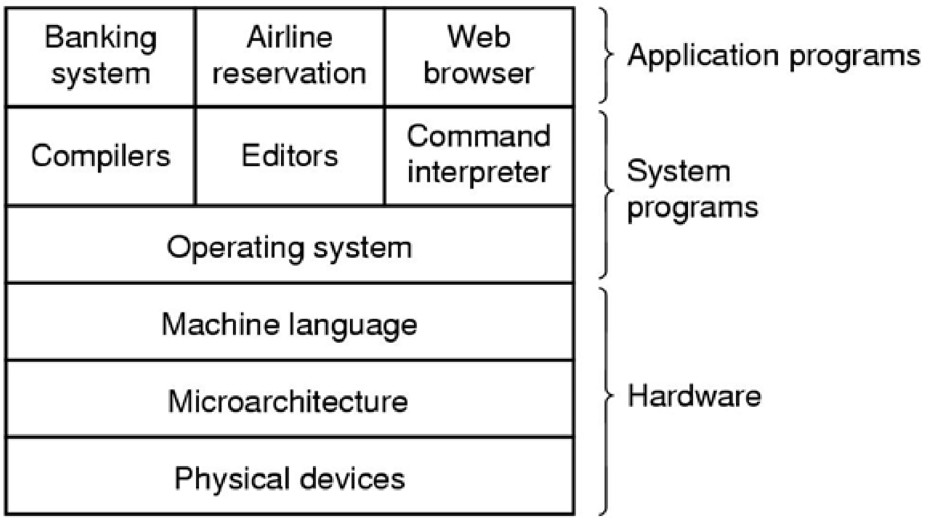
p.33 / Coursebook

1.1. WHAT IS AN OPERATING SYSTEM?

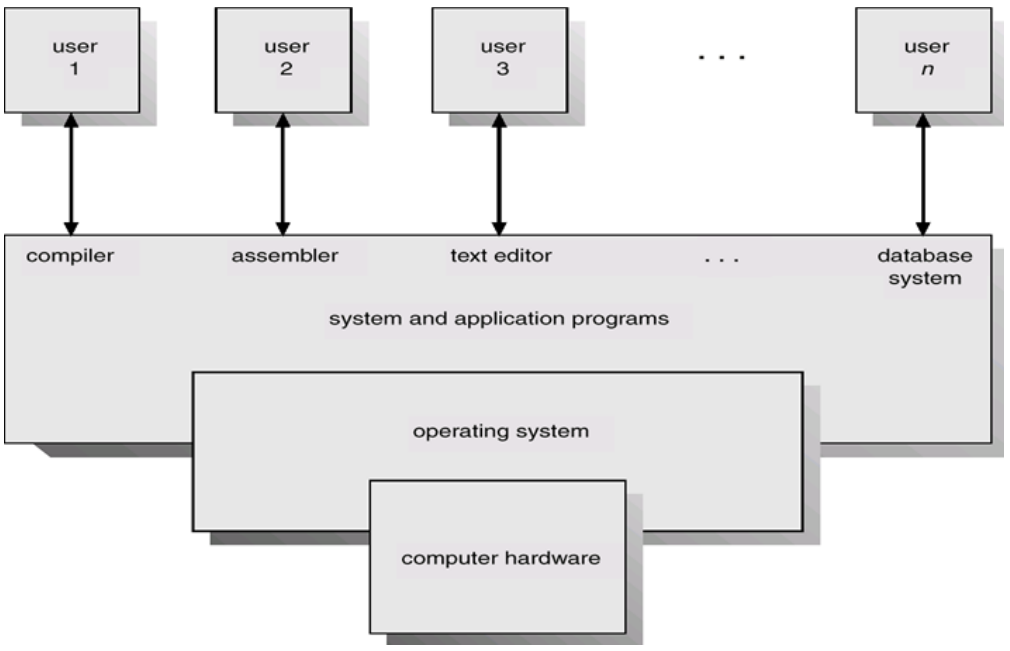
The software that runs in kernel mode - is not always true. OS perform 2 essentially unrelated functions:

- Providing application programmers (and application programs) a clean abstract set of resources instead of the messy hardware ons.
- Managing these hardware resources.

Computer system



What is an OS? → A program that acts as an intermediary between a user of a computer and the computer hardware.



1.1.1. The OS as an Extended Machine

- Hides the messy details which must be performed
- Presents the user with a virtual machine, easier to use

Keywords: `Abstraction`

The **architecture** (ins set, mem organization, I/O, bus structure) of most computers at the machine-language level is primitive and awkward to program.

No programmer would want to deal with this disk at the hardware level. Instead, a piece of software, called a **disk driver**, deals with the hardware and provides an interface to read and write disk blocks, without getting into the details.

This level is much too low for most applications → all OS provide yet another layer of **abstraction** for using disks: `FILES`. Using this abstraction, programs can create, write, read files, without having to deal with the messy details of how the hardware actually works.

Good abstractions turn a nearly impossible task into two manageable ones:

- Defining and implementing the abstractions.
- Using these abstractions to solve the problem at hand.

OS turns ugly hardware into beautiful abstractions.

OS's real customers are the application programs (via application programmers), they deal directly with the OS and its abstractions.

End users deal with the abstractions provided by the user interface, either a command-line shell or graphical interface.

1.1.2. The OS as a Resource Manager

- Each program gets time with the resource
- Each program gets space on the resource

Keywords: `Bottom-up view`, `Top-down view`,

Top-down view

An OS as primarily providing abstractions to application programs.

Bottom-up view

OS manages all the pieces of a complex system.

The job of the OS is to provide for an orderly and controlled allocation of the processors, memories, and I/O devices among various programs wanting them.

When a computer has more than one user, the need for managing and protecting resources since the user might interfere with one another. Users also need to share information. OS's primary tasks:

- keep track of which programs are using which resource
- grant resource requests
- account for usage
- mediate conflicting requests from different programs and users.

Resource management includes **multiplexing (sharing) resources** in two different ways: in time and in space.

▼ In Time

Def: When a resource is time-multiplexed, different programs or users take turns using it. First one of them gets to use the resource, then another, and so on

E.g: only one CPU and multiple programs that want to run on it; sharing the printer

Note: time-multiplexed - who goes next and for how long

▼ In Space

Def: Each one gets part of the resource

E.g: Main memory is normally divided up among several running programs, each one can be resident at the same time; Disk - a single disk can hold files from many users at the same time.

Note: It's more efficient to hold several programs in mem at once rather than give one of them all of it.

1.2. HISTORY OF OS

💡 The first true digital computer was designed by the English mathematician Charles Babbage (1792–1871). Although Babbage spent most of his life and fortune trying to build his "analytical engine," he never got it working properly because it was purely mechanical, and the technology of his day could not produce the required wheels, gears, and cogs to the high precision that he needed. Needless to say, the analytical engine did not have an operating system.
As an interesting historical aside, Babbage realized that he would need software for his analytical engine, so he hired a young woman named Ada Lovelace, who was the daughter of the famed British poet Lord Byron, as the world's first programmer. **The programming language Ada is named after her**

1.2.1. The First Generation (1945-55)

`Vacuum Tubes`

World War II period

Keywords: vacuum tubes, plug boards → operator

Computers: ENIAC, UNIVAC…

Operating System: No OS

Machine Language, plugboards

A single group of engineers designed, built, programmed, operated, and maintained each machine.

- Programming was done in absolute `machine language`: wiring up an electrical circuit by connecting thousand of cables to the `plugboards` to control the machine's basic functions.
- No programming language, No OS

The problems were simple: straightforward mathematical, numerical calculations (grinding out tables of sines, cosines, logarithms; computing artillery trajectories)

By the early 1950s, the introduction of punched cards (thẻ đục lỗ) → possible to write programs on cards and read them in instead of using plugboards.

1.2.2. The Second Generation (1955-65)

`Transistor`, `Batch Systems`

Keywords: mainframes, job, batch system

Computers: IBM 1401, IBM 7094…
Operating System: FMS (Fortran Monitor System), IBSYS for Computer 7094
Batch Systems
+ Function of Early Batch System
+ Structure of a typical FMS job
Separation between designers, builders, programmers, operators and maintenance personnel

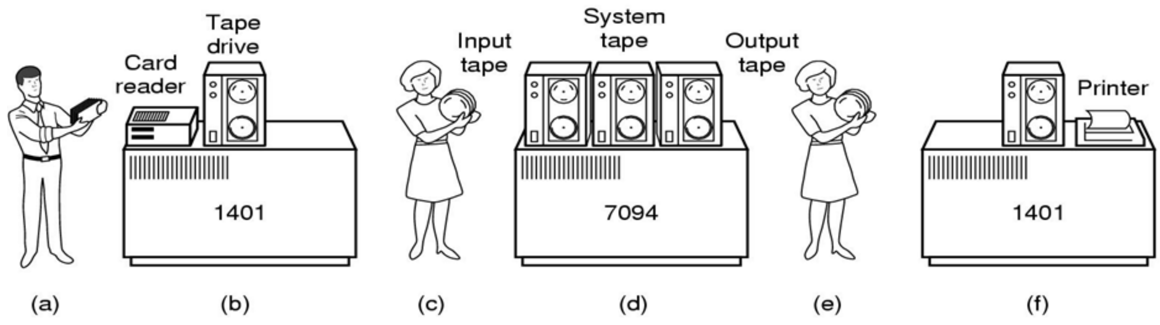
---

Use mostly for scientific and engineering calculations, such as solving the partial differential equations that often occur in physics and engineering.

Programming language: FORTRAN, assembly language.

▼ Early BATCH SYSTEM

bring cards to 1401
read cards to tape
put tape on 7094 which does computing
put tape on 1401 which prints output



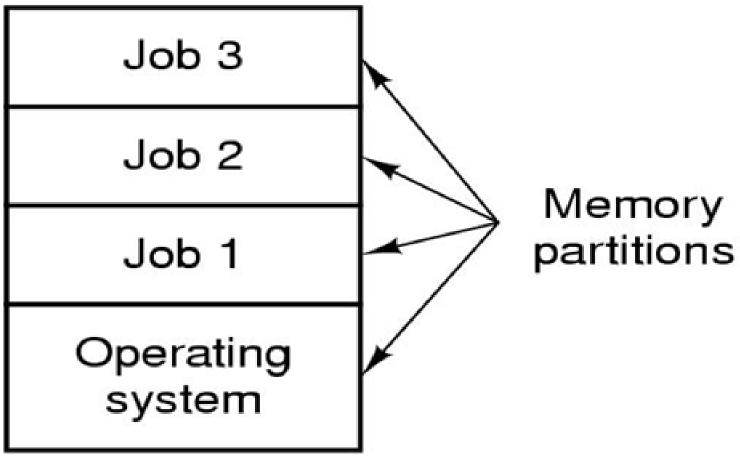
1.2.3. The Third Generation (1965 - 1980)

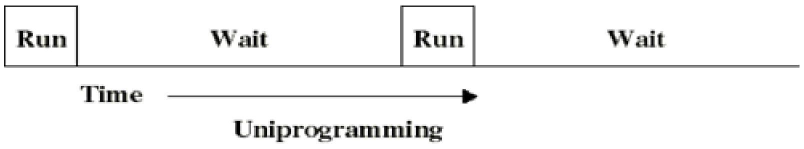
`ICs`, `Multiprogramming`

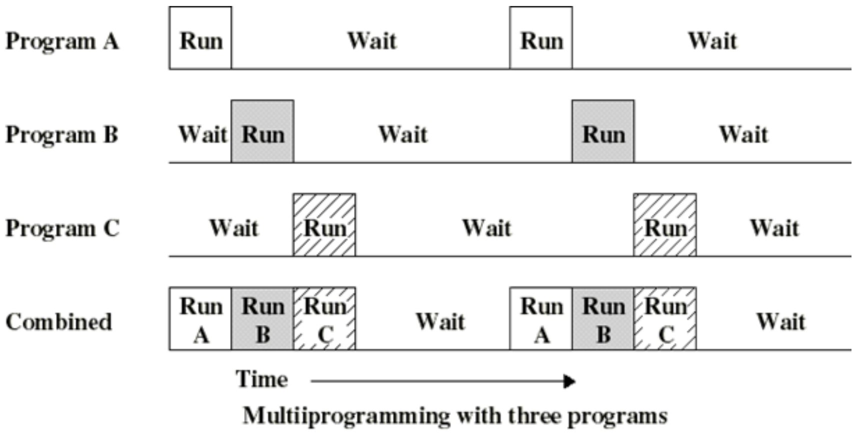
Keywords:
+ OS/360
+ ICs - Integrated Circuits
+ Multiprogramming
+ Spooling - Simultaneous Peripheral Operation On-Line)
+ Time-sharing
+ **CTSS**
+ **MULTICS**
+ Cloud computing
+ Unix
+ BSD - Berkeley Software Distribution
+ System V
+ MINIX, Linux

Computers: System/360, IBM370, IBM4300...
Operating System:

- OS/360: to work on all models
- Multiprogramming
- Time Sharing:
  + CTSS (Compatible Time Sharing System),
  + MULTICS (MULTiplexed Information and Computing Service),
  + Unix







1.2.4. The Fourth Generation 1980 - present

`Personal Computers`

Keywords:
+ LSI - Large Scale Integration
+ Microcomputers
+ CP/M - Control Program for Microcimputers
+ DOS - Disk Operating System
+ MS-DOS - Microsoft Disk Operating System
+ User friendly
+ Mac OS X
+ Windows NT (NT - new technology)
+ Windows Me (Millennium Edition)
+ Service pack
+ Digital Rights Management
+ x86-32, x86-64
+ FreeBSD, Gnome, KDE

+ Network operating systems
+ Distributed operating systems

Computers: IBM PC 80×86, Macintosh...
Operating System:
1977: CP/M (Control Program for Microcomputer)
1980: DOS (Disk Operating System)
GUI with Macintosh
1985-1995: Window 3.x
1995: Window 95
1996: Window NT 4.0
1999: Window 2000
Window 2003
Unix

1.2.5. The Fifth Generation (1990 - Present)

`Mobile Computer`

Keywords:
+ PDA - Personal Digital Assistant
+ Symbian OS
+ iPhone

1.3. The OS Zoo

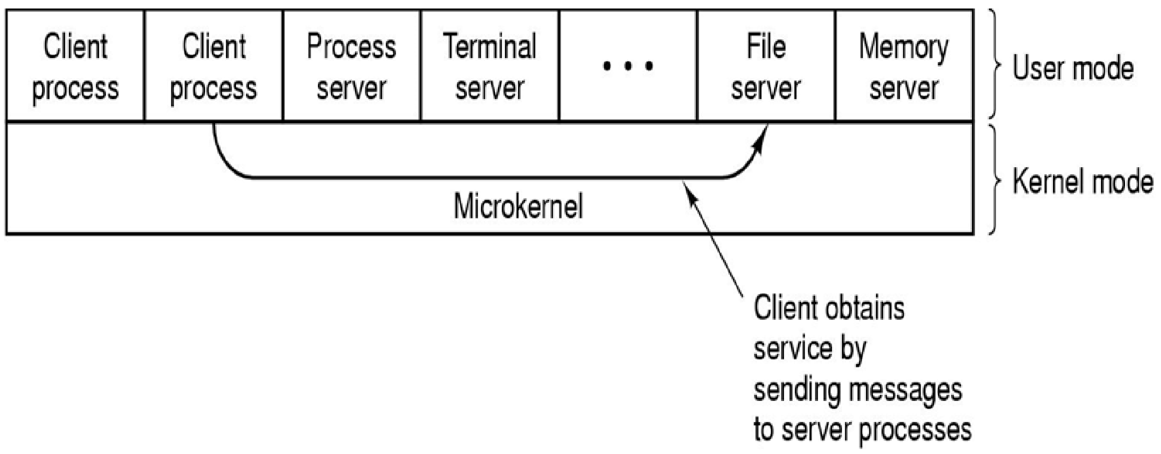
▼ Mainframe operating systems

batch
multiprogrammed
time-sharing, multitasking
Application: High-End Web Server, Servers for Business-To Business transactions
Example: OS/390

▼ Server operating systems

Windows Server NT4, 2000, 2003, 2008, 2012, 2019
Linux: CentOS, RedHat, Ubuntu Server
Solaris
Unix

System core is time-consuming



▼ Multiprocessor operating systems

Multiple CPU
Share computer bus, clock
Advantage:
+ High system throughput
+ High availability
+ Multiprocessor system and Multicomputer system

▼ Personal computer operating systems

Many I/O devices
Interface to single user
Many OS (MS Windows, Mac OS, Solaris, Linux,...).

▼ Real-time operating systems - ROS

Time is a key parameter
Two types of real-time system
+ Hard real-time system for industrial process control system...
+ Soft real-time system for multimedia system

▼ Embedded operating systems

Personal digital assistant (PDA): Palm, Pocket-PC, Cellular phones), control devices
Restriction of memory size, speed of CPU, screen size, powers
Operating System: PalmOS, Windows CE (Consumer Electronic)

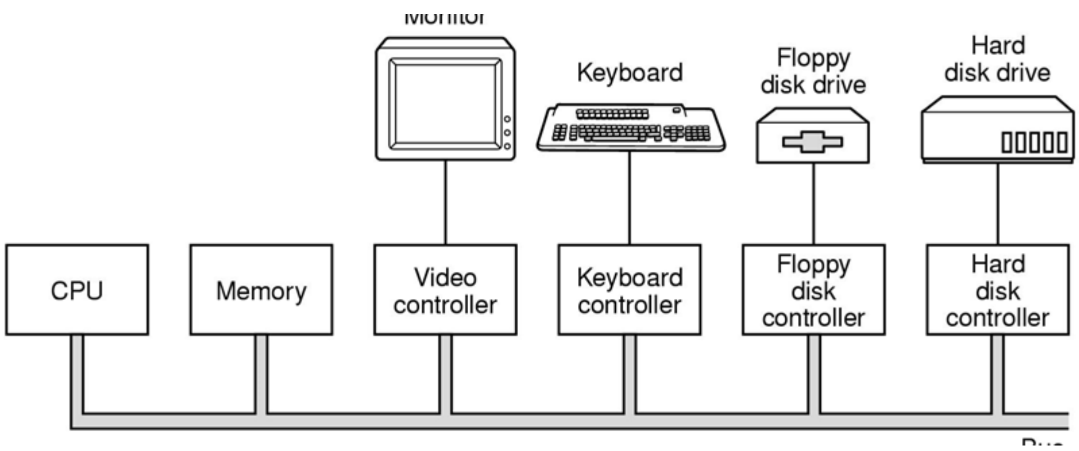
Windows 10 IoT Core,
Raspberry Pi Desktop
https://www.raspberrypi.org/downloads/

palmtop

▼ Smart card operating systems

CPU chips on Card
Server processing power and memory constraint
Specific Application:
+ Single function: electronic payments
+ Multiple function: proprietary systems
+ Java oriented: holds interpreter JVM

1.4. Computer Hardware Review



Components of a simple personal computer

1.4.1. Processors

💡 Fetches instructions from memory and executes them.

▼ The basic cycle of every CPU

Fetch the first instruction from memory, decode it to determine its type and operands, execute it, and then fetch, decode, and execute subsequent instructions. The cycle is repeated until the program finishes.

All CPUs contain some registers inside to hold key variables and temporary results.

Registers: tiny table-storage, intentionally kept small; less than 1KB of data.

Computers have several special registers that are visible to the programmer:

▼ Program counter (PC)

Contain the memory address of the next instructions to be fetched.

After that instruction has been fetched, the program counter is updated to point to its successor.

▼ Stack pointer (SP)

Points to the top of the current stack in memory.

The stack contains one frame for each procedure that has been entered but not yet exited

▼ Program status word (PSW)

Contains the condition code bits, which are set by comparison instructions, the CPU priority, the mode (user or kernel), and various other control bits.

Plays an important role in system calls and I/O.

To improve performance, CPU designers have long abandoned the simple model of fetching, decoding, and executing one instruction at a time.

Organization

Pipiline

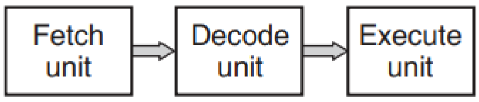
A CPU might have separate fetch, decode, and execute units

▼ **More**

while it is executing instruction n, it could also be decoding instruction n + 1 and fetching instruction n + 2

▼ **Cons**

Pipelines cause compiler writers and operating system writers great headaches because they expose the complexities of the underlying machine to them and they have to deal with them



Superscalar

Multiple execution units are present

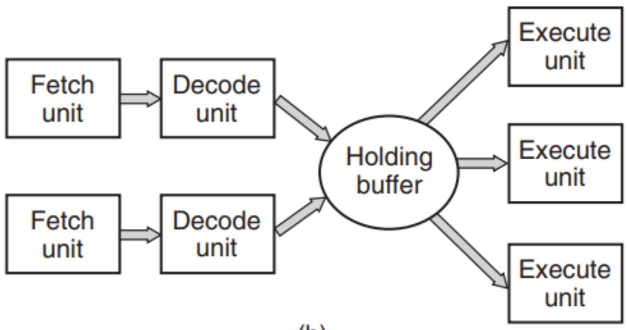
▼ **More**

Two or more instructions are fetched at once, decoded, and dumped into a holding buffer until they can be executed.

As soon as an execution unit becomes available, it looks in the holding buffer to see if there is an instruction it can handle, and if so, it removes the instruction from the buffer and executes it.

▼ **Cons**

program instructions are often executed out of order. For the most part, it is up to the hardware to make sure the result produced is the same one a sequential implementation would have produced, but an annoying amount of the complexity is foisted onto the operating system, as we shall see.

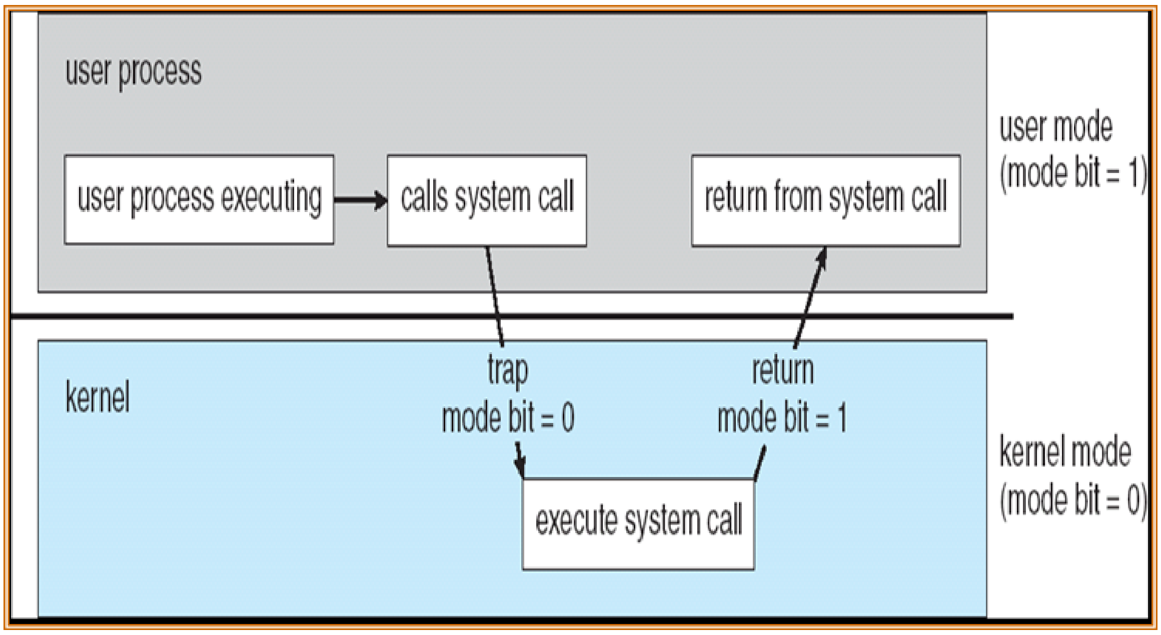


(b)

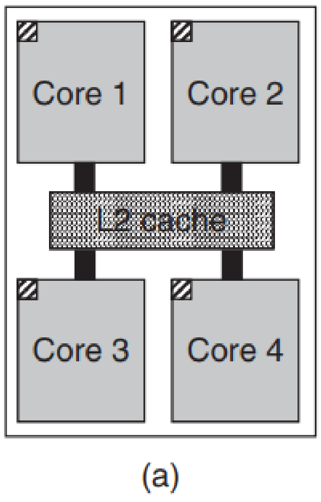
To obtain services from the operating system, a user program must make a `system call`, which traps into the kernel and invokes the operating system.
System call - A special kind of procedure call that has the additional property of switching from user mode to kernel mode.

Dual-mode operation allows OS to protect itself and other system components

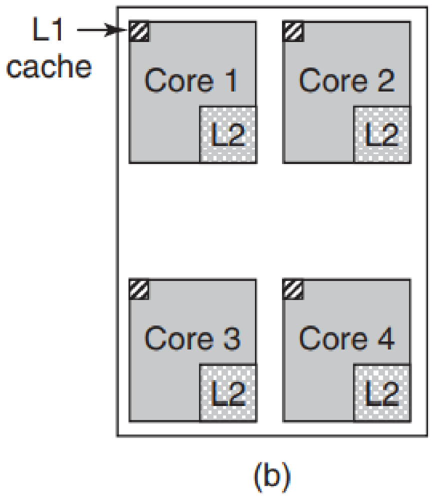
- User mode and kernel mode
- Mode bit provided by hardware
  + Provides ability to distinguish when system is running user code or kernel code
  + Some instructions designated as **privileged**, only executable in kernel mode
  + **System call** changes mode to kernel, return from call resets it to user



Multithread & Multicore Chips



A quad-core chip with a shared L2 cache



A quad-core chip with separate L2 caches

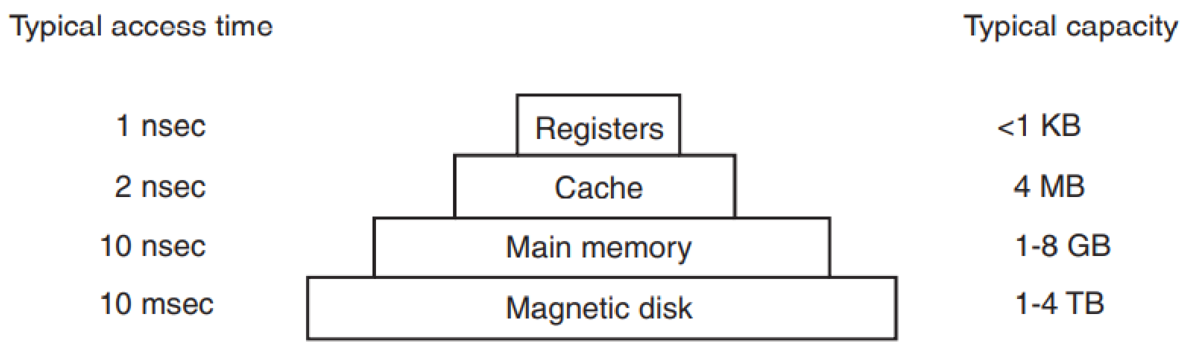
Abundance of transistors → superscalar architectures, with multiple function units.
As the number of transistors increases, even more is possible. → Put bigger caches on the CPU chip.

The next step, replicate not only the functional units but also some of the control logic.
Intel Pentium 4 introduced this property → `multithreading / hyperthreading`

Allow the CPU to hold the state of two different threads and then switch back and forth on a `nanosecond time scale`. Multithreading does not offer true parallelism. Only one process at a time is running. Each thread appears to the operating system as a separate CPU.

Many CPU chips now have 4, 8, or more complete processors or cores. Making use of such a multicore chip will definitely require a `multiprocessor operating system.`

1.4.2. Memory



A typical memory hierarchy

Storage systems are organized in a hierarchy: **Speed; Cost; Size; Volatility.**

Top layers - higher speed, smaller capacity, and greater cost per bit than the lower ones, often by factors of a billion or more.

▼ **Register**

Internal CPU, small number.

Storage available is 32×32 bits (32-bit CPU), 64×64 bits (64-bit CPU).

Programs must manage the registers.

▼ **Cache: c**opying information into the faster storage system

Controlled by the hardware. Main mem is divided up into `cache lines`, 64 bytes (address 0-63 in cache line 0, 64-127 in cache-line 1,...).

Important principle, performed at many levels in a computer (in hardware, operating system, software)

Information in use copied from slower to faster storage temporarily

Faster storage (cache) checked first to determine if information is there
+ If it is, information used directly from the cache (fast)
+ If not, data copied to cache and used there

Cache smaller than storage being cached
+ Cache management important design problem
+ Cache size and replacement policy

When the program needs to read a memory word, the cache hardware checks to see if the line needed is in the cache:
+ Yes: `Cache hit` - no memory request is sent over the bus to the main memory. (about 2 clock cycles)
+ No: `Cache misses` - go to memory, with a substantial time penalty.

Cache memory is limited in size due to its high cost. Some machines have 2 / 3 levels of cache, each one slower and bigger than the one before it.

▼ **Main memory:** all CPU request that cannot be satisfied out of the cache go to main mem

RAM (Random Access Memory) - core memory

Problem:
+ How to protect the program from one another and the kernel from them all
+ How to handle the relocation

Solution: CPU equipped with two special registers

Base Register and Limit Register

MMU (Memory Management Unit) convert Virtual Address to Physical Address

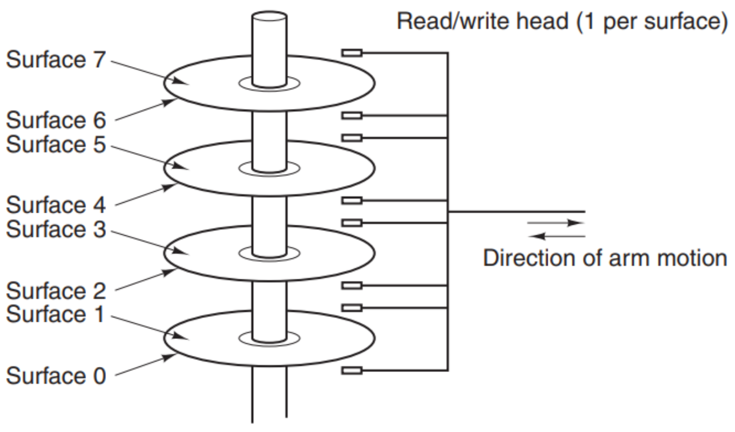
Context Switch; switching from one program to another (may be necessary to flush all modified blocks from the cache and change the mapping registers in the MMU)

Non-volatile random-access memory: does not lose contents when switching off.
ROM: fast, inexpensive; some bootstrap loader used to start the computer is contained in ROM.

EEPROM: BIOS Basic Input Output System

Flash memory:

1.4.3. Disks



Structure of a disk drive

Disk storage: cheaper, larger than RAM; the time to randomly access data is slower. (Because disk is a mechanical device)

A disk consists of one or more metal platters that rotate at 5400, 7200, 10800 rpm

Information is written onto the disk in a series of concentric circles.

At any given arm position, each of the heads can read an annular region called a `track`.

Together, all the tracks for a given arm position form a `cylinder`.

Track divided into sectors, typically 512 bytes/sector. Outer cylinders contain more sectors than inner.

Access time:

- Moving arm: to next cylinder 1 msec, random 5-10 msec
- Drive must wait for the needed sector to rotate under the head → delay of 5-10 msec
- Read/write: at a rate of 50MB-160MB/sec

1.4.4. I/O devices

Computer-System Operation
+ I/O devices and the CPU can execute concurrently.
+ Each device controller is in charge of a particular device type.
+ Each device controller has a **local buffer.**
+ CPU moves data from/to main memory to/from local buffers
+ I/O is from the device to local buffer of controller.
+ Device controller informs CPU that it has finished its operation by causing an **interrupt**.

Term:

▼ The controller

A chip or a set of chips that physically controls the device.

It accepts commands from the operating system.

▼ Devices

Devices have fairly simple interfaces, both because they cannot do much and to make them standard. The latter is needed so that any SAT A disk controller can handle any SAT A disk.

Device driver: The software that talks to a controller, giving it commands and accepting responses.

To be used, the driver has to be put into the operating system so it can run in kernel mode. Drivers can actually run outside the kernel, and operating systems like Linux and Windows nowadays do offer some support for doing so. The vast majority of the drivers still run below the kernel boundary.

▼ SATA

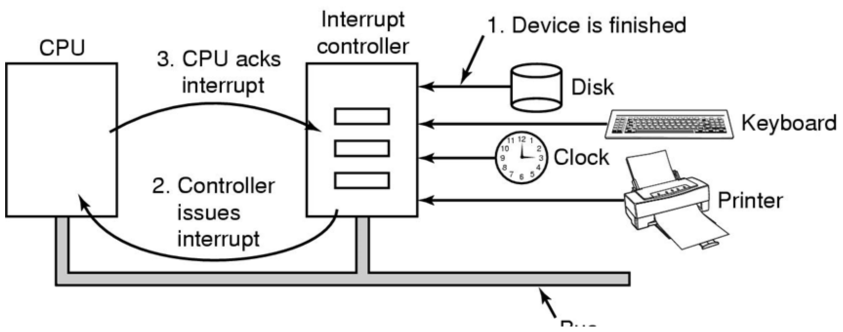
Serial Advanced Technology Attachment

The standard type of disk on many computers

▼ 3 ways the driver can be put into the kernel

1. Relink the kernel with the new driver and then reboot the system.
2. Make an entry in an operating system file telling it that it needs the driver and then reboot the system. At boot time, the operating system goes and finds the drivers it needs and loads them. (Window works this way)
3. For the operating system to be able to accept new drivers while running and install them on the fly without the need to reboot (dynamically loaded drivers).

Every controller has a small number of registers that are used to communicate with it. The collection of all the device registers forms the **I/O port space.**



How interrupts happens. Connections between devices and interrupt controllers actually use interrupt lines on the bus rather than dedicated wires.

▼ Common Functions of **Interrupts:**

- Interrupt transfers control to the interrupt service routine generally, through the interrupt vector table, which contains the addresses of all the service routines.
- Interrupt architecture must save the address of the interrupted instruction.
- Incoming interrupts are disabled while another interrupt is being processed to prevent a lost interrupt.
- A trap is a software-generated interrupt caused either by an error or a user request.
- An operating system is interrupt-driven.

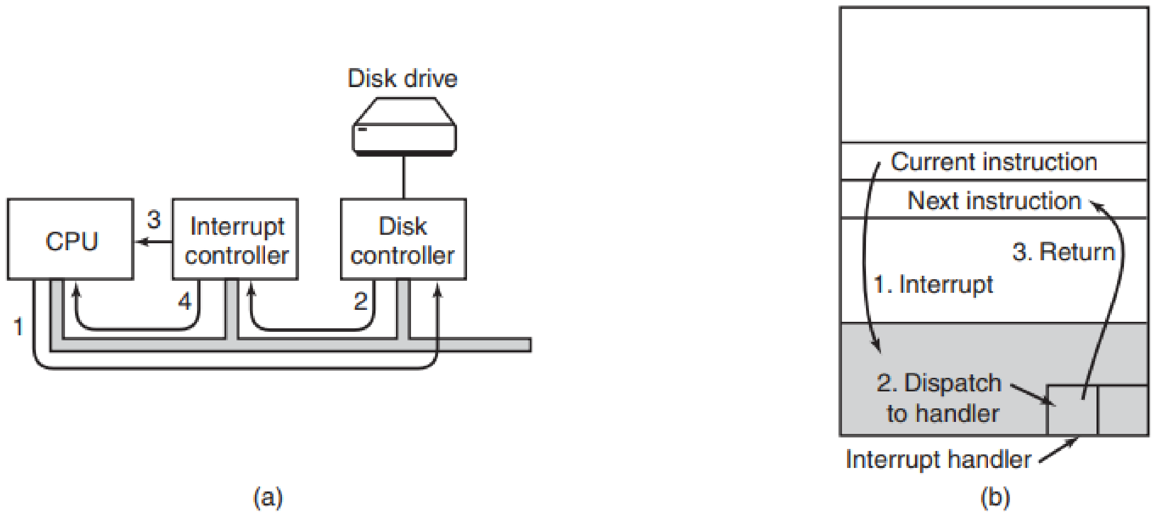
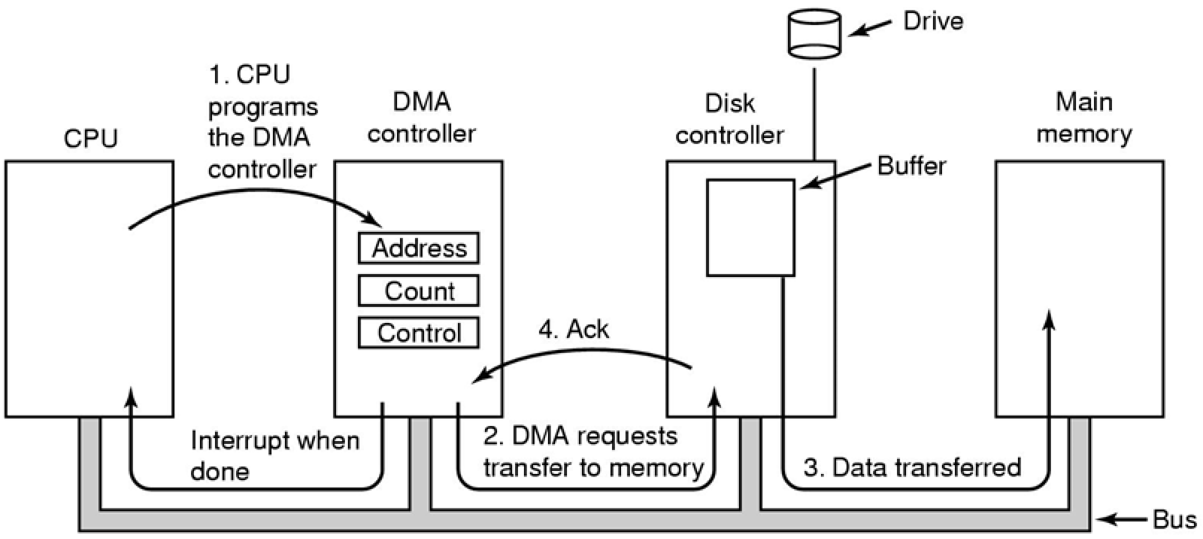


Figure 1-11. (a) The steps in starting an I/O device and getting an interrupt. (b) Interrupt processing involves taking the interrupt, running the interrupt handler, and returning to the user program.

DMA (Direct Memory Access)



Used for high-speed I/O devices able to transmit information at **close to memory speeds.**

Device controller transfers **blocks of data** from buffer storage directly to main memory without CPU intervention.

Only one interrupt is generated per block, rather than the one interrupt per byte.

1.4.5. Buses

Cache BUS

Local BUS

System BUS

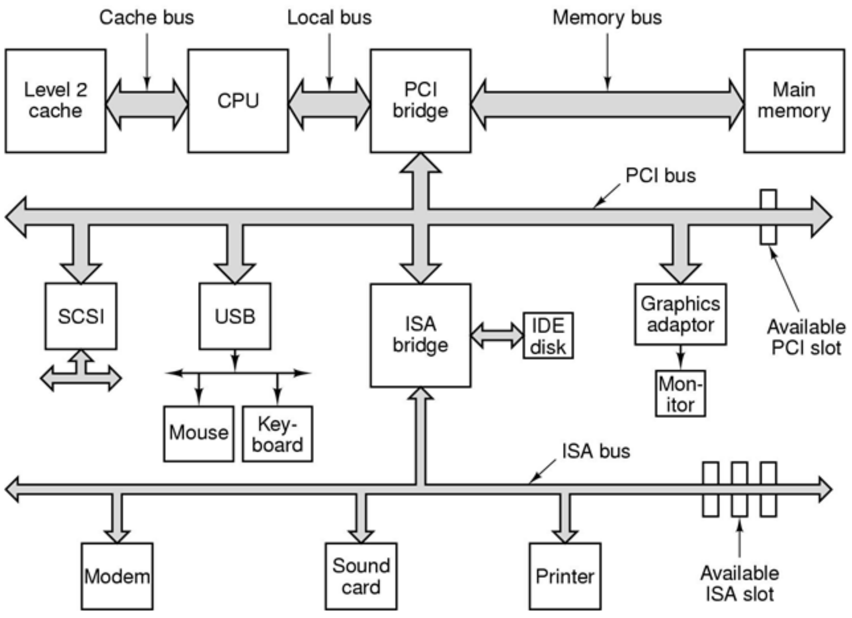
ISA - Industry Standard Architecture

PCI - Peripheral Component Interconnect / PCIe - Peripheral Component Interconnect Express)

USB - Universal Serial Bus

SCSI - Small Computer System Interface

IDE - Integrated Drive Electronic



- Architecture

  Shared bus architecture

  Parallel bus architecture

  Serial bus architecture

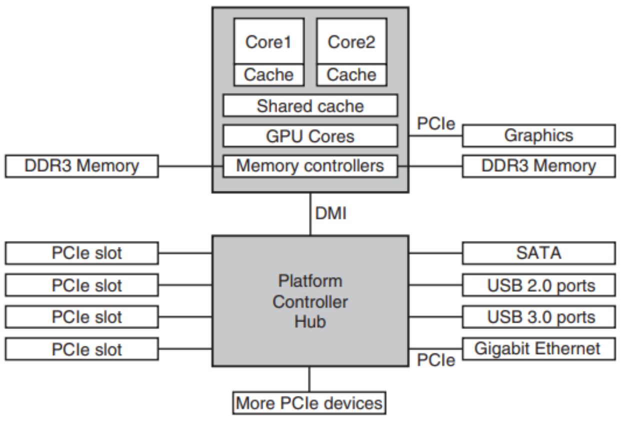


Figure 1-12. The structure of a large x86 system.

1.4.6. Booting the computer

- Boot process

  Every PC contains a parentboard, there is a program - system BIOS (Basic input-output system).

  BIOS first checks how much RAM is installed and whether the keyboard and other basic devices are installed and responding correctly. (scanning the PCIe and PCI buses to detect all the devices attached to them.)

The BIOS then determines the boot device by trying a list of devices stored in the CMOS memory.

The OS queries the BIOS to get the configuration information. For each device, it checks to see if it has the device driver. Once it has all the device drivers, the operating system loads them into the kernel. Then it initializes its tables, creates whatever background processes are needed, and starts up a login program or GUI.

▼ BIOS

BIOS contains low-level I/O software, including procedures to read the keyboard, write to the screen, and do disk I/O, among other things.

Nowadays, it is held in a flash RAM, which is nonvolatile but can be updated by the operating system when bugs are found in the BIOS.

1.5. Operating system concepts

OS components

System calls

OS structure

Process: is basically a program in execution.

Address space: a list of memory locations from 0 to some maximum, which the process can read and write. Address space contains the executable program, program's data, and its stack. **(p.72)**

Set of resources: commonly including registers (program counter and stack pointer), a list of open files, outstanding alarms, lists of related processes, all other info needed to run a program.

⇒ A process is fundamentally a container that holds all the information needed to run a program.

Each person authorized to use a system is assigned a **UID (User IDentification)** by the system administrator. Every process started has the UID of the person who started it.

A `child process` has the same UID as its parent.

Users can be members of groups, each of which has a **GID (Group IDentification)**

One UID, called the `superuser` (in UNIX), or `Administrator` (in Windows), has a special power and may override many of the protection rules.

▼ **Files**

A major function of the operating system is to hide the peculiarities of the disks and other I/O devices and present the programmer with a nice, clean abstract model of device-independent **files.**

Directory: group files together. Given rise to a hierarchy - the file system.

Every file within the directory hierarchy can be specified by giving its `path name` from the top of the directory hierarchy, the `root directory`.

The process and file hierarchies both are organized as trees.

Process hierarchy **File hierarchy**

Not very deep (more than three levels is unusual)	commonly 4, 5, or even more levels deep
Short-lived, generally minutes at most	Exist for years
Only a parent process may control or even access a child process.	Files and directories to be read by a wider group than just the owner.

▼ **Pipe**

Is a sort of pseudo-file that can be used to connect two processes.

If processes A and B wish to talk using a pipe, they must set it up in advance.

The implementation of a pipe is very much like that of a file.

Communication between processes in UNIX looks very much like ordinary file reads and writes.

The only way a process can discover that the output file it is writing on is not really a file, but a **pipe** is by making a special system call. File systems are very important.

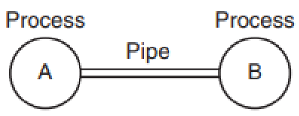


Figure 1-16. Two processes connected by a pipe.

1.5.1. OS components

a) Process management

A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.

▼ **Tasks of process management of OS:**

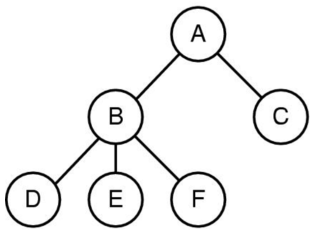
Process creation and deletion.

Process suspension and resume

Provision of mechanisms for:
- Process synchronization
- Interprocess communication
- Prevent or avoid deadlock

▼ **Process tree**

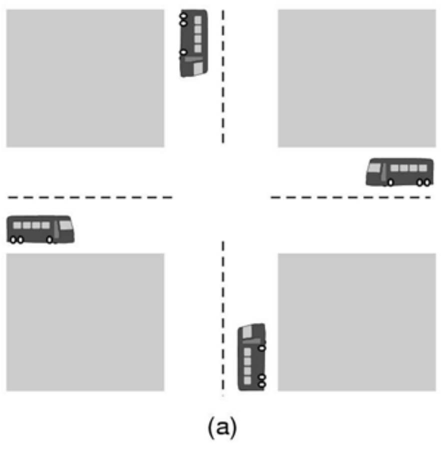
HĐH phản hồi vs ng dung, cho ng dung thấy 1 phần kết quả nhanh nhất
Chia nhỏ các chương trình lớn, thực thi từng cái và cho ra output



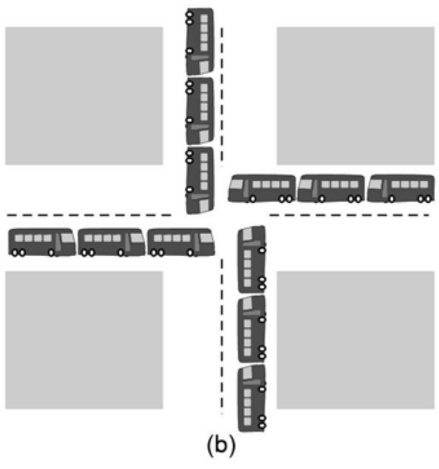
Two processes connected by a pipe

▼ **Deadlock problem**

A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.



(a) A potential deadlock.



(b) an actual deadlock.

b) Main Memory Management

Motivations:

- Increase system performance
- Maximize memory utilization

Task of main memory management:

- Keep track of which parts of memory are currently being used and by whom.
- Decide which processes to load when memory space becomes available.
- Allocate and deallocate memory space as needed

c) File management

File system:

- File
- Directory

Task of file management of OS:

- Create and delete File/Directory
- Manipulate: rename, copy, move, new,...
- Mapping files onto secondary storage.
- File backup on stable (nonvolatile) storage media

d) I/O Management

Hide the specialty of H/W devices

Task of I/O Management of OS:

- Manage main memory for the devices using `caching`, `buffering`, and `spooling`
- Maintain and provide a general device-driver interfaces
- Drivers for specific hardware devices.

e) Secondary Storage Management

Since main memory (primary storage) is volatile and too small to accommodate all data and programs permanently, the computer system must provide secondary storage to `back up` main memory.

Common secondary storage devices: `Magnetic disk` and `Optical disk`

Task of Secondary Storage Management of OS:

- Free space management
- Storage allocation
- Disk scheduling

f) Protection and Security

Protection – any mechanism for **controlling access** of processes or users to resources defined by the OS.

Security – defense of the system against **internal and external attacks**

Huge range, including denial-of-service, worms, viruses, identity theft, theft of service.

▼ Systems generally first distinguish among users, to determine who can do what

  User identities (user IDs, security IDs) include name and associated number, one per user.

  User ID then associated with all files, processes of that user to determine access control.

  Group identifier (group ID) allows set of users to be defined and controls managed, then also associated with each process, file.

  Privilege escalation allows user to change to effective ID with more rights.

1.5.2. OS services

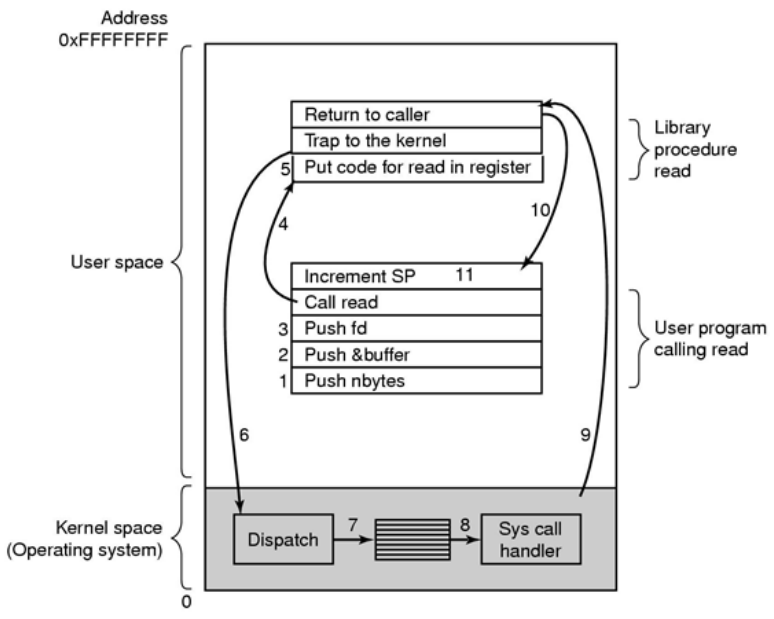
One set of operating-system services provides functions that are helpful to the user:

- User interface: Command-Line (CLI), Graphics User Interface (GUI)
- Program execution: The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
- Control access to I/O device.
- File-system manipulation
- Communications – Processes may exchange information, on the same computer or between computers over a network
- Error detection – OS needs to be constantly aware of possible errors

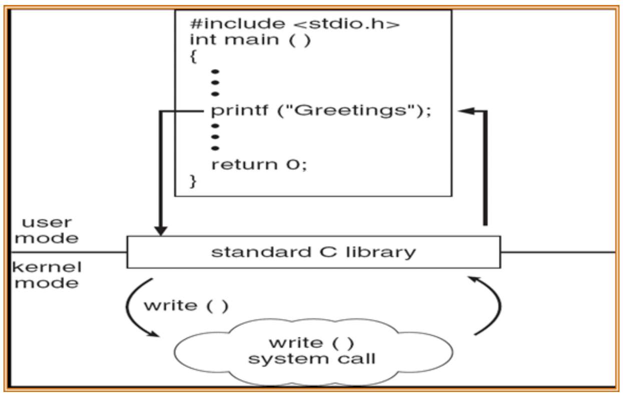
1.5.3. System calls

Making System Calls
Major POSIX System Calls
Examples of System Calls

a) Making System calls



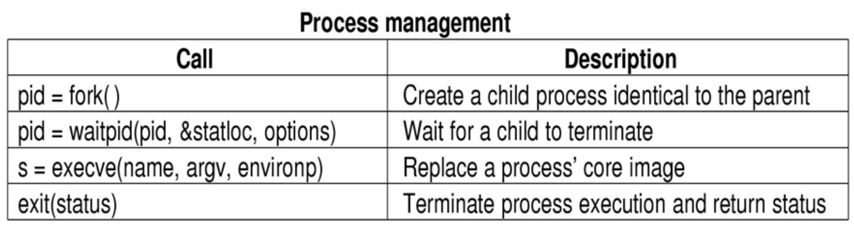
There are 11 steps in making the system call
`read (fd, buffer, nbytes)`



b) POSIX System Calls

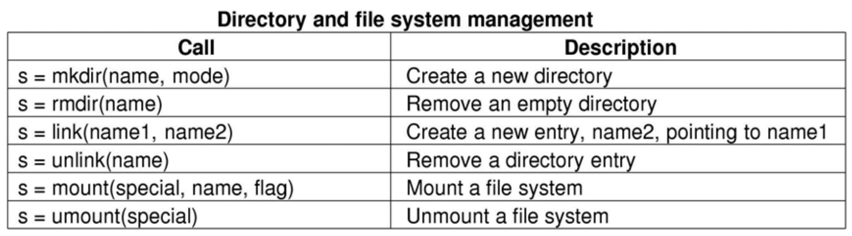
Some System Calls For Process Management

POSIX (Portable Operating System Interface)


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

Some System Calls For File, Directory Management


Directory and file system management

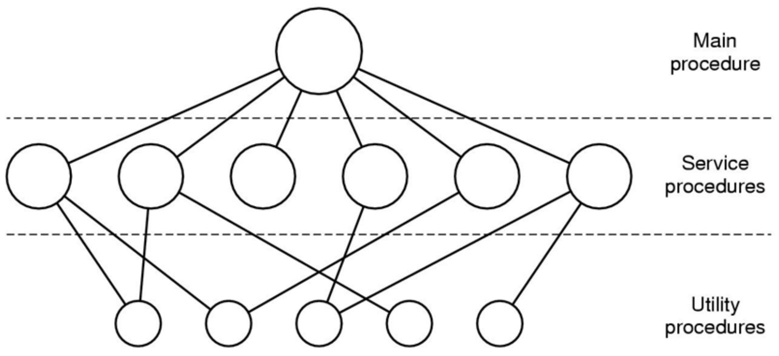
Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

1.5.4. OS structure

Monolithic system
Layered System
Virtual Machine
Client-server model
Microkernel

a) Monolithic system

Hệ thống nguyên khối



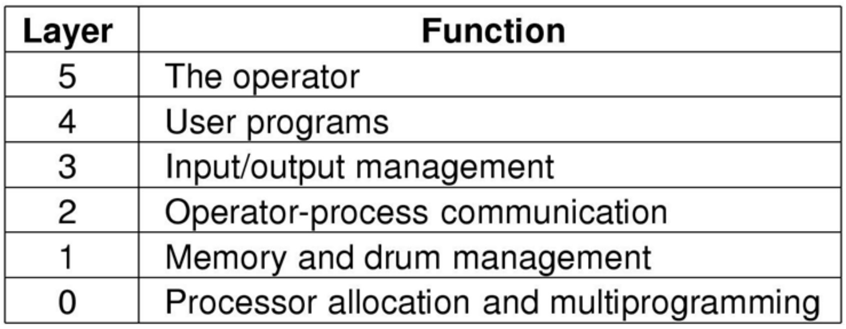
Simple structuring model for a monolithic system

Structure:

- A main program: invokes the requested service procedure.
- A set of service procedures: carry out the system calls.
- A set of utility procedures: help the service procedures.

E.g: MS-DOS

b) Layered 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

Structure of the THE operating system

Many Layers

Each layer has well defined functions

Upper layer can only calls functions of closely lower layer

▼ **Advantages:**

  Easier to extend
  Easier to debug from lower to upper layer

Use for modern model

c) Virtual Machine

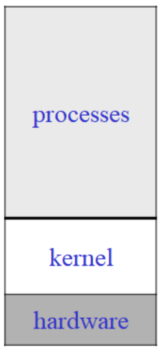
A **virtual machine** takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware.

Provide an interface *identical* to the underlying bare hardware.

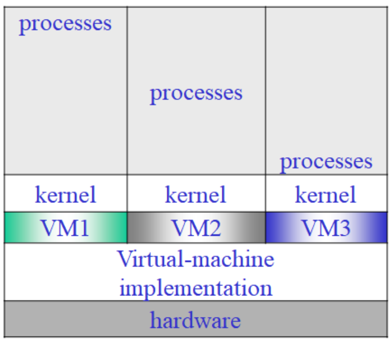
The OS creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory.

▼ The resources of the physical computer are shared to create VM

- CPU scheduling can create the appearance that users have their own processor
- `Spooling` and a file system can provide virtual card readers and virtual line printers
- A normal user time-sharing terminal serves as the virtual machine operator's console

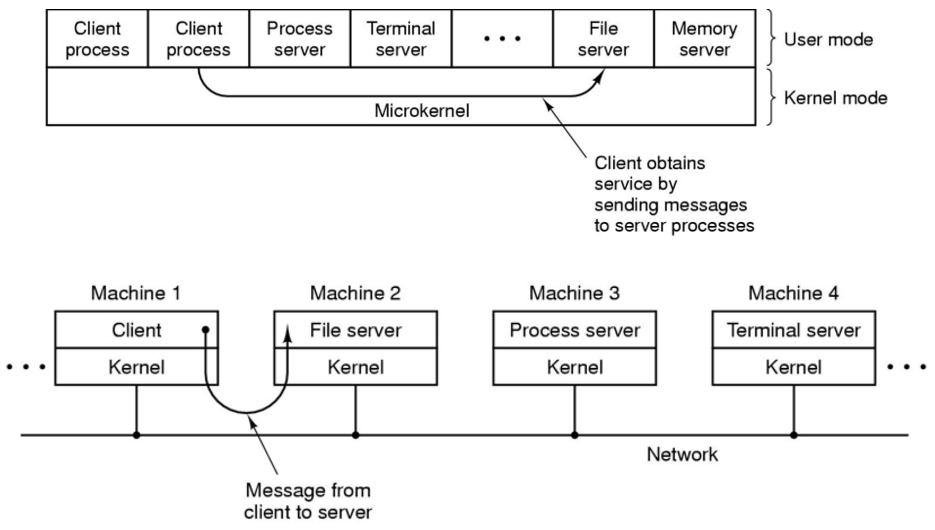


Non-VM system model



VM machine system model

d) Client-Server model

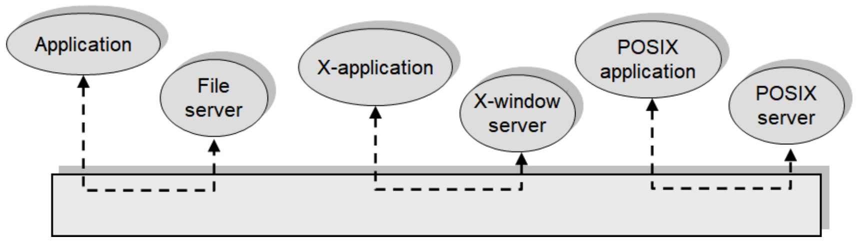


The client-server model in a distributed system

e) Microkernel

OS design: The operating system is divided into microkernel (CMU Mach OS - 1980)

- Moves as much from the kernel into "user" space
- kernel → microkernel
- Communication takes place between user modules using message passing.



Monolithic:

- Single
- (+) Fast | (-) Not flexible
- E.g: DOS

Layered:

- (+) Module hóa → dễ bảo trì
- (-) Phải xác định số layer
- E.g: Windows

Client-server:

- Kết hợp nhiều máy tính với nhau, 1 máy là server, còn lại là client
- Server sẽ cung cấp dịch vụ
- E.g: Mail, Web, App

Microkernel:

- Core lớn được chia thành các core nhỏ
- (+) Giảm cấu hình, độ phức tạp và số lỗi; Một số chức năng không có nhu cầu nên được cắt giảm

VM:

- E.g: VMware
