1. Because the teacher is fucking awful
so decided to learn from NESO Academy

2. Total 28 lectures about $15 \times 28 = 420$ mins $\approx$ 7hrs

3. Have 18 weeks with 54 hrs

→ Aims : 6 weeks finish about 4 lesson in one week

4. Syllables : Introduction ×1

Basics OS ×3

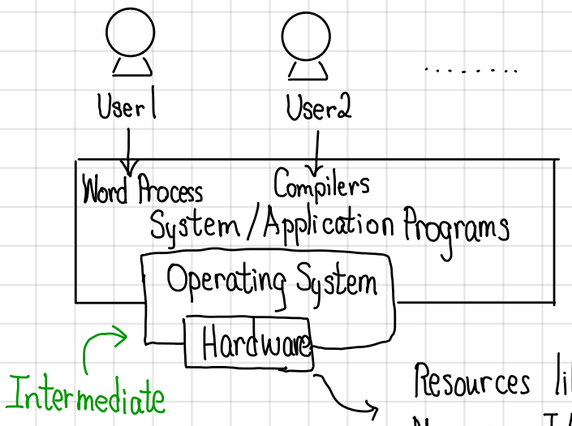
OS structure ×2

Process ×4.

Introduction to Operating System

→ An Operating System (OS) is a program that manages the computer hardware.

→ provides a basis for Application Programs and acts as an intermediary between User and Hardware



→ Windows, Linux, Ubuntu, Mac OSX, iOS, Android

Types of OS:
→ Batch OS
→ Time Sharing OS
→ Distributed OS
→ Network OS
→ Real Time OS
→ Multi Programming / Processing / Tasking OS

- Goals
  i) Convenience
  ii) Efficiency
  iii) Both

functions of OS
→ interface < User / Hardware
→ Allocation of Resources
→ Management of { Memory / Security

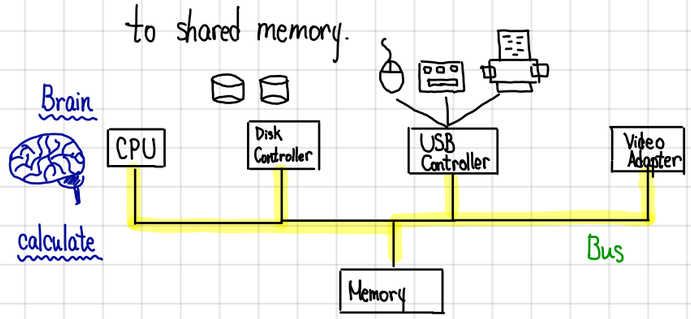
☆ Schedule is important

Basic of Operating System

↳ basic knowledge of structure of Computer System to understand OS.

Definition

A modern general-purpose computer system consists one or more CPUs and a number of device controllers connected through a common bus that provides access to shared memory.



→ Each device controller is in charge of a specific type of device
→ The CPU and the device controllers and can execute concurrently (Same-time), competing for memory cycles.
→ To ensure orderly access to the shared memory, a memory controller whose function is to synchronize access to the memory.

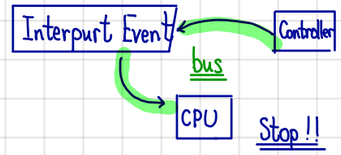
- When the CPU is interrupted, it stops what it is doing and immediately transfers execution to a fixed location.
  ↓
  The fixed location usually contains the starting address where the Service Routine of the interrupt is located.
  Something where, what the interrupts actually wants to do.
  ↝ written in Service Routine
  (ISR)
  → The Interrupt Service Routine executes

On completion, the CPU resumes the interrupted computation.

Important terms

1) Bootstrap Program → The initial program that runs when a computer is powered up or rebooted.
   → It is stored in the ROM. (Read only Mem)
   → It must know how to load the OS and start executing that system.
   → It must locate and load into memory the OS Kernel.
   Heart

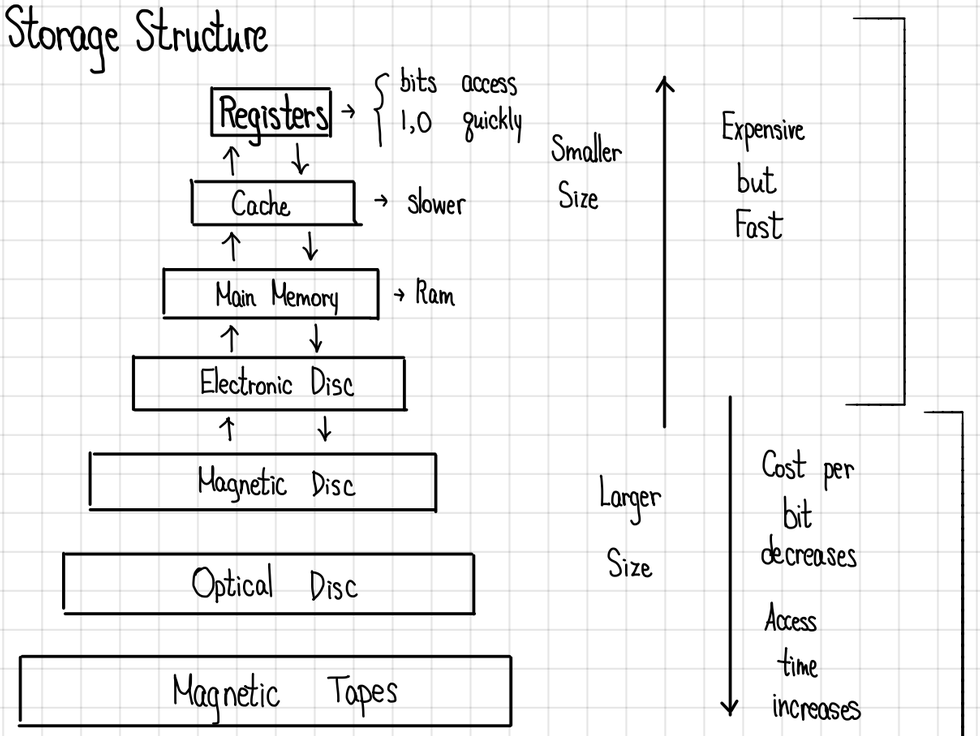
2) Interrupt → The occurence of an event is usually signalled by an Interrupt from Hardware or Software.



→ Hardware may trigger an interrupt at any time by sending a signal to the CPU, usually by the way of the way of the System bus.

3) System Call (Monitor call) + Software may trigger an interrupt by executing a special operation called System call

- Storage Structure



volatile
→ loses its contents when power is removed

non-volatile
→ Retains its contents even when power is removed

- Main Memory
→ RAM
→ All Executions loaded here before executed.
→ volatile
→ Size limited
→ faster when we have more RAM

- Secondary Memory
→ store the data
→ loaded into Main Memory when needed
→ bigger, larger

I/O Structure

→ storage is only one of the many type of I/O devices within a computer
→ A large portion of operating System code is dedicated to managing I/O, both because of its importance to the reliability and performance of a system and because of the varing nature of the device
→ A general-purpose computer system consists of CPUs and multiple device controllers that are connected through a common bus.

→ Remember the basic concept of the computer

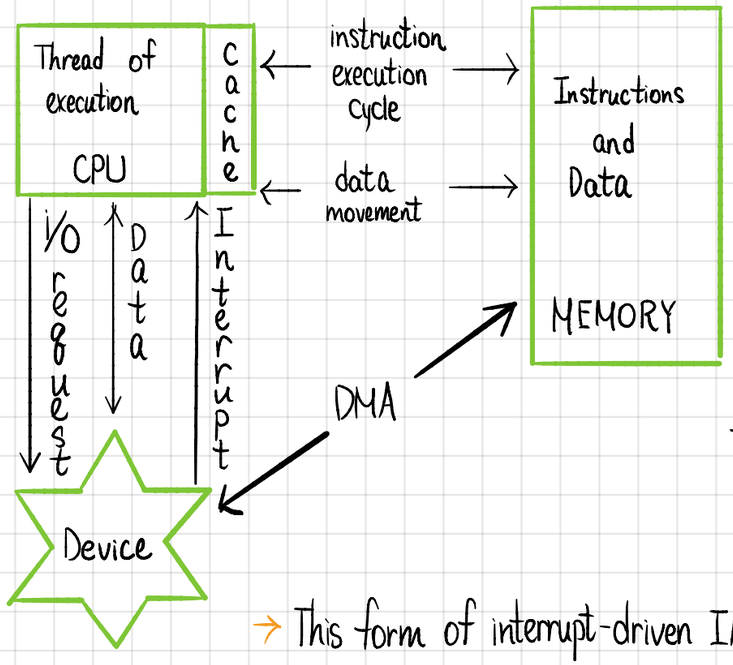
→ Each device controller is in charge of a specific type of device

maintains
Local Buffer Storage
Set of Special Purpose Registers

→ Typically, operating systems have device driver for each device controller

→ This device driver understands the device controller and presents a uniform interface to the device to the rest of the operating system.

Working of an I/O Operating :



→ To start an I/O operation, the device driver loads the appropriate registers within the device controllers

→ These device controller, in turn, examines the contents of these registers to determine what action to take

→ The controller starts the transfer of data from the device to its local buffer in the device

→ Once the transfer of data is complete, the device controller informs the device driver via an interrupt that it has finished its operation.

→ The device driver then returns control to the operating system.

→ This form of interrupt-driven I/O is fine for moving small amounts of data but can produce high overhead when used for bulk data movement

→ To solve this problem, Direct Memory Access is used (DMA)

After setting up buffers, pointers, and counters for the I/O device, the device controller transfers an entire block of data directly to or from its own buffer storage to memory, with no intervention by the CPU.

still needs CPU

but transfer without CPU intermediate.

→ Only one interrupt is generate per block, to tell the device that the operation has completed.

→ While the device controller is performing these operations, the CPU is available to accomplish other works.

Computer System Architecture

Main purpose → Types of Computer Systems based on number of General Purpose Processors:
→ we categorize

1. Single Processor Systems
2. Multiprocessor Systems
3. Clustered Systems

(1.) Single Processor Systems:

- One main CPU capable of executing a general purpose instruction set including instructions from user process.
- Other special purpose processors are also present which perform device specific tasks. etc. key board, mouse → contain micro-processor
  - ↳ Although the system may contain multiple processors ; however, it only has one general purpose processors.

(2) Multiprocessor Systems:

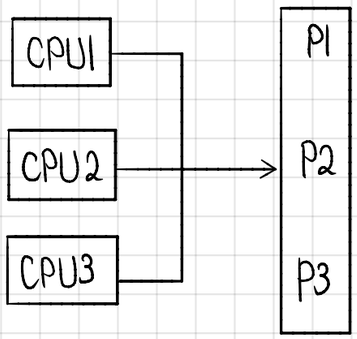
- Also known as parallel systems or tightly coupled systems.
- Has two or more processors in close communication, sharing the computer bus and somethings the clock, memory and peripheral devices.

Advantage:

1. Icreased throughput (amount of data that can be transferred from one location to another ~ measurement of the performance)
2. Economy of Scale { multiple processors are more economic / compare to single processors
   etc. 2× multiple processors (performance) ⇒ 2 × single processors (need more individual resources)
3. Increase reliability ~ same reason as up
   etc. when one processor have failed multiple processors are able to keep perform tasks

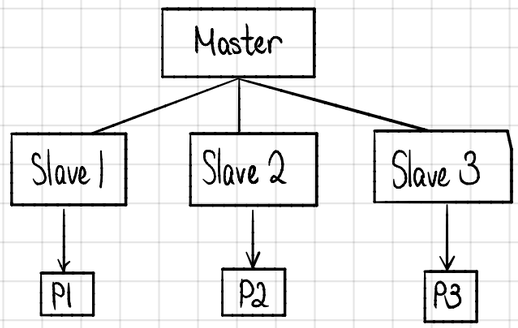
Types of multiprocessor Systems:

Symmetric Multiprocessing



→ Every CPU are same as each other
→ Like Peers.

Asymmetric Multiprocessing



→ Master assign task to each Slave
→ Master is a monitor

3. Clustered Systems

- Like multiprocessor systems, clusted systems gather together multiple CPUs to accomplish computational work.
- They are composed of two or more individual systems coupled together.
- Provides high availability
  - ↳ when systems fail it still is able to operate
  - ↳ unlikely to complete failure
- Can be structed asymmetrically or symmetrically (better)
  - asymmetrically:
    - Master: · One machine in Hot-Standby mode
    - Slave: · Others run applications; · Monitors each other
  - symmetrically:
    - · Two or more hosts run applications

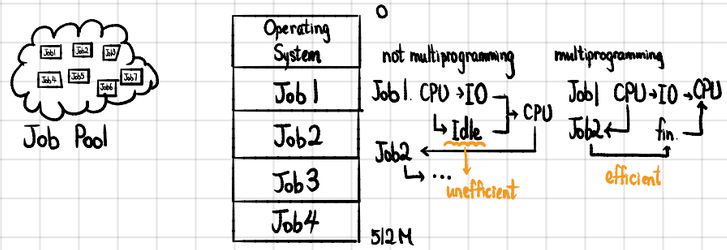
Operating System Structure

(Multiprogramming & Multitasking)

- Operating Systems vary greatly in their makeup internally
- Commonalities
  - → Multiprogramming
  - → Time sharing (Multitasking)

Multiprogramming

- A single user cannot, ingeneral, keep either the CPU or the I/O devices busy at all times
- Multiprogramming increases utilization by organizing jobs (code and data) so that the CPU always has one to execute.



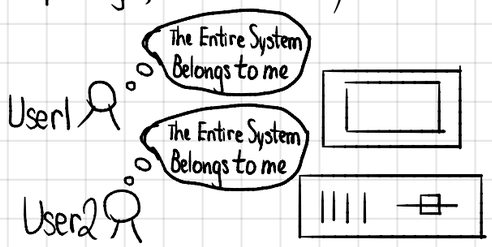
Memory layout for a multiprogramming system

Conclusion:

Multiprogrammed systems provide an evironments in which the various system resources (for example, CPU, memory, and periphical devices) are utilized effectively, but they do not provide for user interation with the Computer system.

Time sharing (Multitasking)

- CPU executes multiple jobs by switching among them
- Switches occur so frequently that the users can interact with each program while it is running.
- Time sharing requires an interactive (or hands-on) computer system, which provides direct communication between the user and the system.
- A time-shared operating system allows many users to share the computer simultaneously



The time gap between User's action > Computer job switching

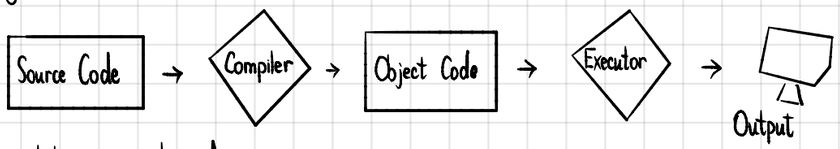
- Uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer.
- Each user at least one seperate program in memory
- A program loaded into and executing is called Process

Operating Systems Services

- An OS provides an environments for the execution of programs.
- It provides certain services to programs and to users of those programs.

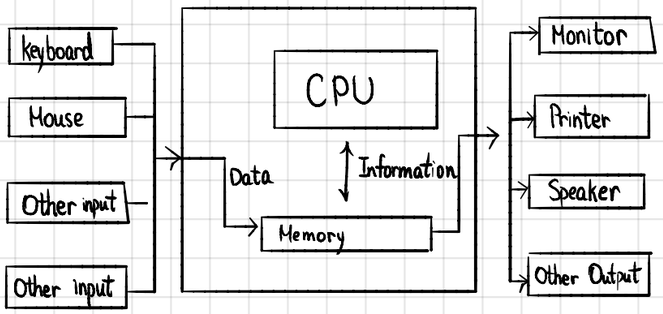
1) User Interface
   → Allow User to interact with the operating system or itself
   → ex: Command Line Interface (CLI)
   Graphical User Interface (GUI)

2) Program Execution



⇒ able to load program to the memory and run the program.

3) I/O Operations



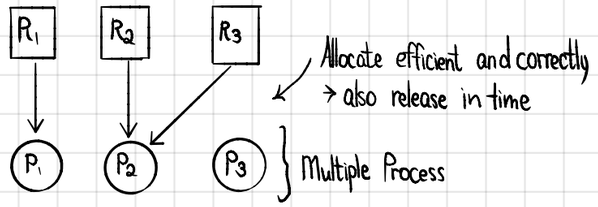
- Users are unable to directly control hardware but through I/O systems

4) Files System Manipulation
   → Search, add, delete files
   → Restrictions with accessing files

5) Communications
   → between Process (a program that is in execution)
   → may between { same computer / different computers but tied with networks

6) Error detection
   → Error may occur in everyway
   → but we have to handle and not to break down completely
   → debugging methods.

7) Resource Allocation (etc. memory, I/O, CPU ....)



8) Accounting
   → keep track of Users Useage and thoroughput of Resource
   → how the resources is used, actually if it is working

9) Security and Protection
   → Data, process should be secure
   → process should not interfer each other
   → access to resource should be controlled
   → outsiders shouldn't be access
   → record invalid access
   → should be inplement thoroughly

CLI example:
- Linuxs System

james@james-computer
↑ User Name ↑ Computer name

User Operating Interface

There are two fundamental approaches for users to interface with operating system:

1) Provide a Command-Line Interface (CLI) or Command Interpreter that allows users to directly enter commands that are to be performed by the operating system.

2) Allow the user to interface with the operating system via a Graphical User Interface or GUI

Command Interpreter

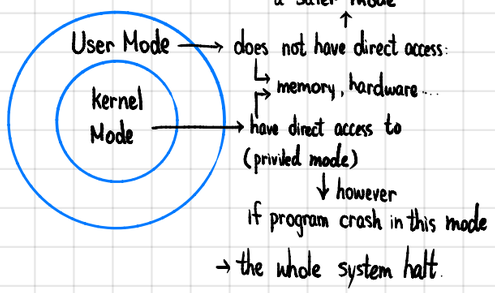
- Some operating systems include the command interpreter in the kernel.
- Others, such as Windows XP and UNIX, treat the command interpreter as a special program. (Not Linux, often used by Enterprise.)
- On systems with multiple command interpreters to choose from, the interpreters are known as shells.
  E.g.
  - Bourne shell
  - C shell
  - Bourne-Again shell (BASH)
  - Korn shell, etc.

How command Interpreter perform the tasks that we enter the shell?
→ 1) code for certain task is included in CI itself
ex: task (create a file)
↳ enter the code → CI perform it
→ 2) code are written in certain programs
ex: task
↳ enter the code → CI call the program
↓
certain program execute it

System Calls

- System calls provide an interface to the sevices made available by an Operating System.

Different Mode Need to understand :



∴ Most Program use User Mode

↳ But still need access to hardware resourse

↳ it switches to kernel mode → access to resource

(context switching)

↓

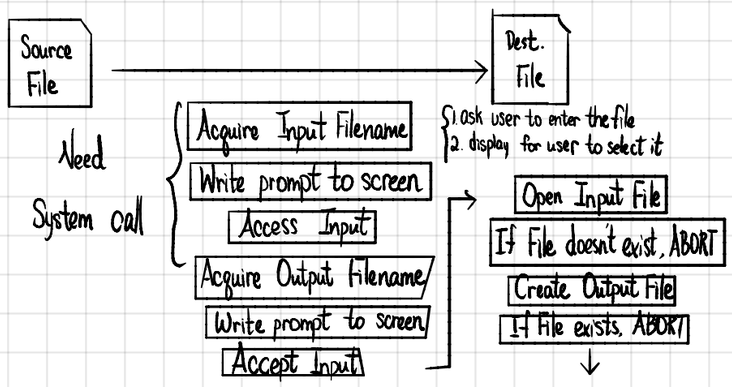
fin.

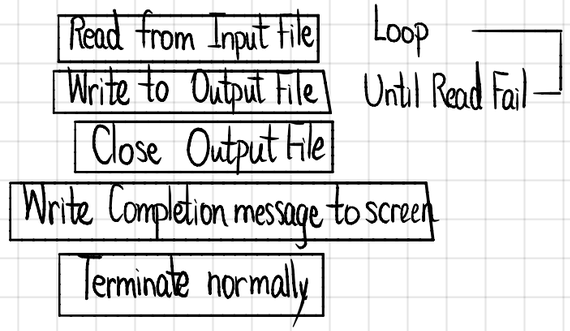
switch back ←

- System call is the programmatic way in which a computer program requests a service from the kernel of the operating system.
- These calls are generally available as routines written in C and C++

Example :

System call sequence for writing a simple program to read data from one file and copy them to another file





Types of System Calls

1. Process Control
2. File Manipulation
3. Device Management
4. Information Maintenance
5. Communications

1) Process Control

- end, abort { abort because of an error / halt normally
- load, execute
- create process, terminate process (process controll calls)
- get process attribute, set process attribute
- wait for time
- wait event, signal event
- allocate & free memory

2. File Manipulation

- create file, delete file
- open, close
- read, write, reposition
- get file attributes, set file attributes

3. Device Manipulation

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

not physically

4. Information Maintenance (Info. of sys.)

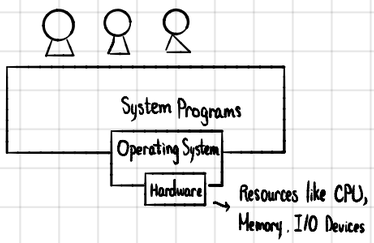
- get time or date, set time or date
- get system data, set system data
- get process, file, or device attributes
- set process, file, or device attributes

5. Communications

- Create, delete communication connection
- Send, receive messages
- transfer status information
- attach or detach remote devices

System calls

An important aspect of a modern system is the collection of system programs



- System programs provide a convenient environment for program development and execution.
- Some of them are simply user interfaces to system calls.
- Others are considerably more complex.

System Programs can be devided into the following categories:

File Management

- Create, Delete, Copy, Rename, Print, Dump List, and generally manipulate files and directories

Status Information

Ask the system for:

- Date, time, Amount of available memory or disk space, Number of users, Detailed performance, Logging, and debugging information etc.

File modification

- Several text editors may be available to create and modify the content of files stored on disk or other storage devices.
- There may also be special commands to search contents of files or perform transformations of the text.

Programming-language support

- Compilers, Assembles, Debuggers, Interpreters
- from common programming languages (Such as C, C++, Java, Visual Basic, and PERL) are often provided to the user with operating system.

Program loading and execution

- Once a program is assembled or compiled, it must be loaded into memory to be executed.
- The system may provide:
  - → Absolute loaders
  - → Linkage editors and
  - → Relocatable loaders
  - → Overlay loaders
- Debugging systems for either higher-level languages or machine language are needed as well.

Communications

These programs provide the mechanism for:

- Create Visual Connections among processes, users, and Computer systems.
- Allowing users to send messages to one another's screens
- To browse websites
- To send electronic-mail messages
- To log in remotely or to transfer files from one machine to another.

In addition to systems programs, most operating systems are supplied with programs that are useful in solving common problems or performing common operations.

ex: Web brower → Chrome
Word Processors → Word
} Application Programs

⋮

Operating System Design & Implementation

Design Goals:

1st Problem: Defining Goals and specification

↳ hard to have specify all the goals → diff. user : diff. specification

- Choice of Hardware
- Type of System

→ Beyond this highest design level, the requirements may be much harder to specify.

Requirements:

→ User Goals (User Requirements): { Convenient to use; Easy to learn & use; Reliable, safe & fast.

→ System Goals (Engineer, Designer Requirements): { Easy to design, Implement, maintain/operate; It should be flexible, reliable, error free & efficient.

- No particular formula

Mechanisms and Policies

{ Mechanisms determine how to do something
{ Policies determine what will be done

ex: driving a car

→ mechanisms: help the car move

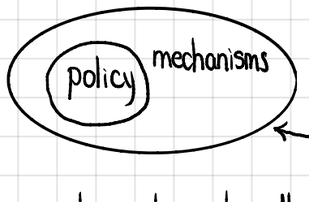
→ policies : should not exceed the speed

Principle 1 : Seperation of policies from mechanisms

→ mechanisms : car keep moving

| Independence

policy: slow down the car when it exceed the speed.

→ 

car moves at 50 km/hr

why not good: New policy → Need to change mechanisms

Implementation:

➢ Once an operating system is designed, it must be implemented.

➢ Traditionally, operating systems have been written in assembly language

➢ Now, however, they are most commonly written in higher-level languages such as C or C++

Advantages of writing in higher level language:

➢ The code can be written faster

➢ It is more compact (緊湊)

➢ It is easier to understand and debug

➢ It is easier to port (hardware ✶ OS (Window) / OS (Unbuntu))

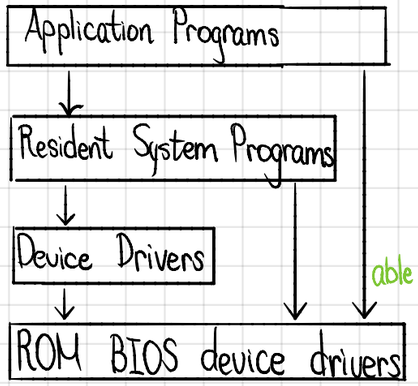
eg MS-DOS was written in Intel 8088 assembly language → only available on Intel family of CPUs

↕ contrast

The linux operating system, in contrast, is written mostly in C and is available on a number of different CPUs → Android family, Ubuntu.

Structure of operating System

Simple Structure



able to access hardware directly

← base hardware

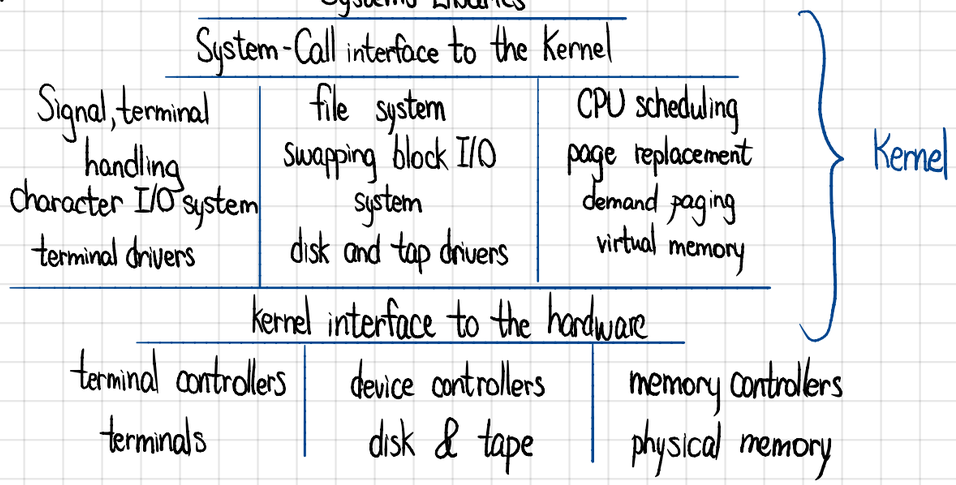
- not well defined
- MS-DOS
- not layered system
- vulnerable to malacious program

Monolithic Structure

→ Unix

- functions are packed into one layer
- too many functions into one layer

→ hard to maintain

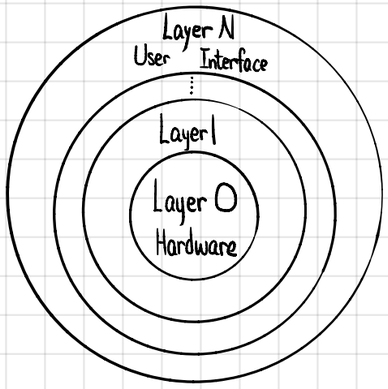


Ch2 Review Questions

1. What is “shell” in Unix? Describe its usage.
2. What is system call?
3. What is API?
4. What are differences between system call and API?
5. How does user program pass a block of data by a system call?
6. What is system program?
7. What is the structure of MS-DOS, UNIX, and Solaris?
8 How does Cocoa work? → IOS, 2-48 , Object C.
9 How does Android work? → 2-47
10. How does DTrace work? → 2.51
11. What is bootstrap program? How does it work?

開機程式,

Layered Structure



- Every layer have different functionality
- Debug seperate layer
- Only able access from outside to inside

Disadvanage:

- hard to design

ex: ↓ many requirement

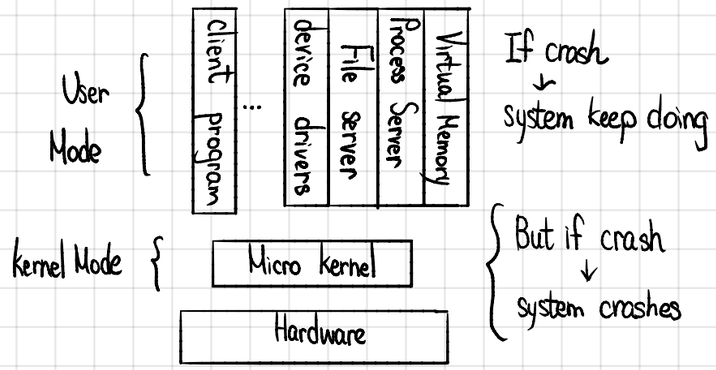
memorary management (outer layer)

↓

backing storage (inner layer)

- May not be efficient
  - ~ too many layer
  - ~ pass message one layer by one

Microkernels



Micro kernel → only provide necessary program

→ Able to access other application

→ provide communication between services

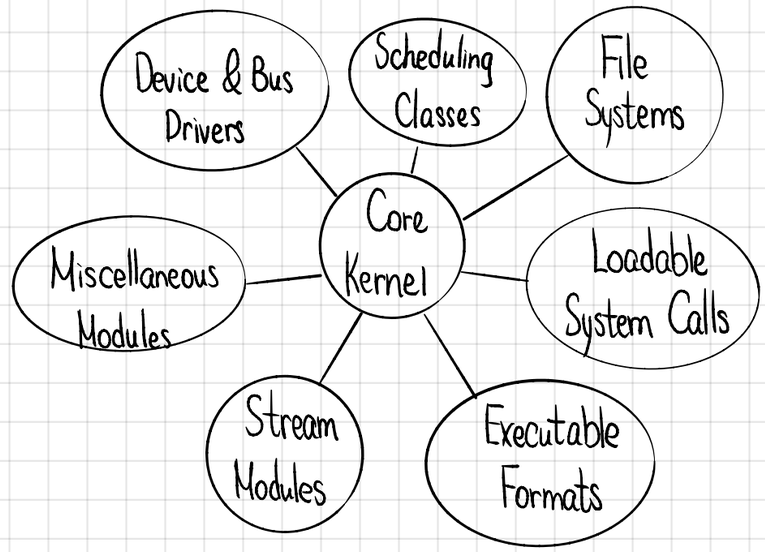
↳ message passing

Disadvantage:

→ System Overhead ⇒ decrease performance

any combination of excess or indirect computation time, memory, bandwidth, or other resources

Modules



↓

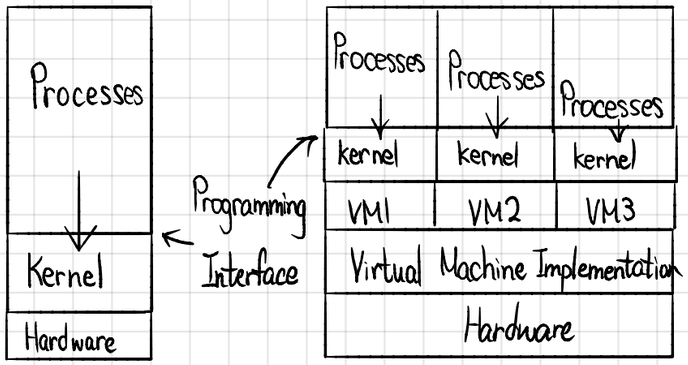
like microkernel

- each layers have protected layers
- flexible, more efficient, no need to go through all layers.
- used by most O.S

Virtual Machine

The fundamental idea behind a virtual machine is to abstract the hardware of a single computer (the CPU, memory, disk drivers, network interface cards, and so forth) into several different execution environments, thereby creating the illusion that each seperate execution environment is running its own private computer.

· feel like running on a private computer



(a) Nonvirtual Machine virtual Machine

Implementation

Real Physical Machine

Virtual Machine Software → Runs in Kernel mode

Virtual Machine itself → Runs in User mode

Just as the physical machine has two modes, however, so must the virtual machine.

Consequently, we must have:

- A virtual user mode &
- A virtual kernel mode

Both of which run in a physical User mode

→ Protection of resources.

→ Particular area of disk will be assigned to each VM.

Operating System Generation & System Boot

- Design, code, and implement an operating system specifically for one machine at one site
- Operating systems are designed to run on any of a class of machines of a variety of sites with a variety of peripheral configurations.
  → run on a variety machine
- The system must be configured or generated for each specific computer site, a process sometimes known as system generation (SYSGEN) is used for this.
  ↳ generate O.S in a way compatible to specific computer
- The following kinds of information must be determined by the SYSGEN Program?
  → What CPU is to be used?
  → How much memory is available?
  → What devices are available?
  → What operating-system options are desired?

System Boot

- The procedure of starting a computer by loading the kernel is known as booting the system.
- On most computer systems, a small piece of code known as the bootstrap program or bootstrap loader locates the kernel.
- This program is in the form of read-only memory (ROM), because the RAM is in an unknown state at system startup.
  ROM is convenient → no need initialization, not infected by computer virus.
- Firmware
  → Small device
  → EPROM
  Erasable Programable ROM
  → program by explicitly command
- When the full bootstrap program has been loaded, it can tranverse the file system to find the operating system kernel, load it into memory, and start its execution. It is only at this point that the system is said to be Running

Process Management

(Processes and Threads)

High Level Program

↓ compiler

binary code (Allocate by O.S)

↓ execute ← Need Resources

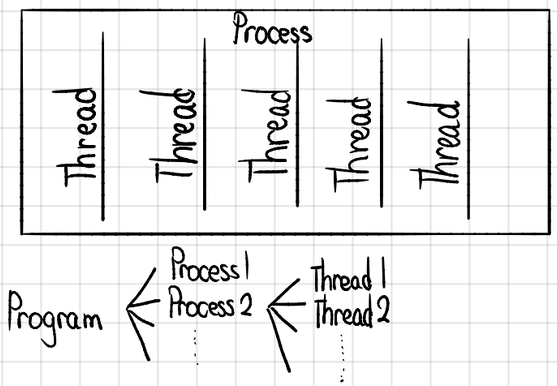
Process

A process can be thought of as a program in execution.

Idle → a pasive program

Thread

A thread is the unit of execution within a process. A process can have anywhere from just one thread to many threads.



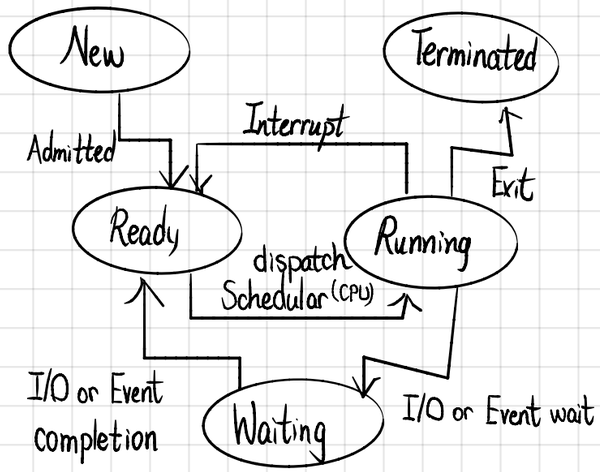
Process State

- As a process executes, it changes state.
- The state of a process is defined in part by the current activity of that process.

Each process can be in one of the following states:

- **New** The process is being created
- **Running** Instructions are being executed
- **Waiting** The process is waiting for some event to occur (Such as an I/O completion or reception of a signal)
- **Ready** The process is waiting to be assigned to a processor.
- **Terminated** The process has finished execution.

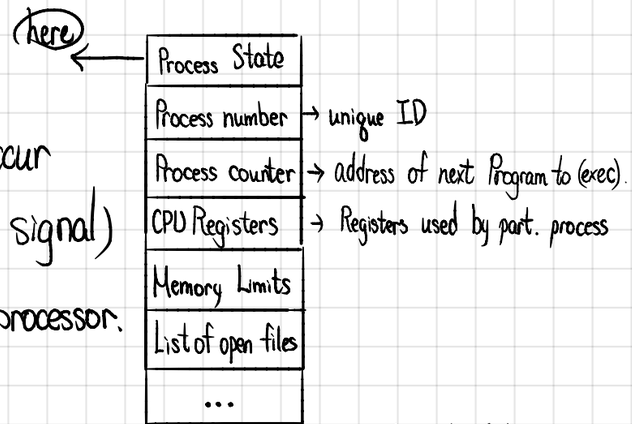
Diagram of process state



Process Control Block

Each process is represented in the operating system by a Process Control Block (PCB) - also called a task control block.

PCB diagram:



- CPU scheduling inform. → { priority of the process; pointer to scheduling queue; scheduling parameter }
  - scheduling → which process will be executed
- memory management inform.
  - → memory used by part. process
- accounting Inform.
  - → the resources that has being used
- Input / Ouput inform.
  - I/O status assigned to part. process

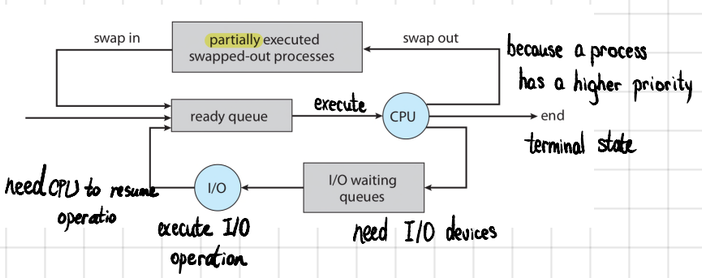
Process Scheduling

- The objective of multiprocessing is to have some process running at all times, to maximize CPU utilization.
  → prevent wasting resources
- The objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running.
- To meet these objectives, the process scheduler selects an available process (possibly from a set of several available processes) for program execution on the CPU.
  - For a single-processor system, there will never be more than one running process.
  - If there are more processes, the rest will have to wait until the CPU is free and be rescheduled.

Scheduling Queues

Job Queue As processes enter the system, they are put into a job queue, which consists of all processes in the system.

Ready Queue The process that are residing in the main memory and are ready and waiting to execute are kept on a list called the ready queue.



Context Switch

- Interrupts cause the operating system to change a CPU from its current task & to run a kernel routine.
- Such operations happen frequently an general-purpose systems.

When an interrupt occurs, the system needs to save the current Context of the process currently running on the CPU so that it can restore that context when its processing is done, essentially suspending the process and then resuming it.

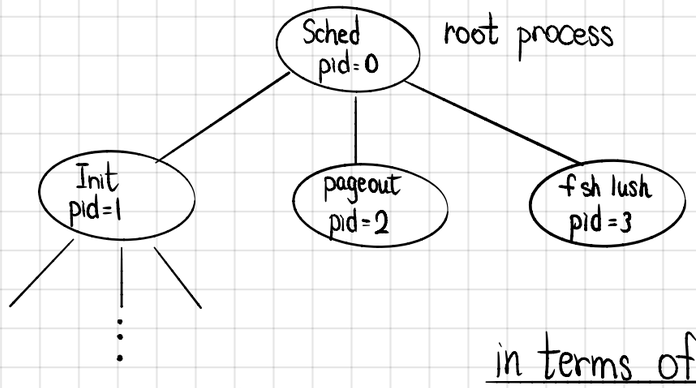
- The context is represented in the PCB of the process
  ↳ the current PCB
  PCB = Process Control Block
- Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process.
  → This task is known as a context switch
- Context-switch time is pure overhead, because the system does no useful work while switching. → no useful task happens (means the switch part)
  the cost of doing something
- Its speed varies from machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions (such as single instruction to load or store all registers)
- Typical speeds are a milliseconds.

Operation on Process

(Process Creation)

- A Process may create several new processes, via a create-process system call, during the course of execution.
- The creating process is called a parent process, and the new processes are called the children of that process.
- Each of these new processes may in turn create other processes, forming a tree of processes.

Figure: A tree of processes on a typical Solaris system



in terms of execution

When a process creates a new process, two possibilities exists

1. The parent continues to execute concurrently with its children
2. The parent waits until some or all of its children have terminated.

→ the subprocess may have the resource (all or some) from the parent

There are also two possibilities in terms of the address space of the new process

1. The child process is a duplicate of parent process (it has the same program and the data as the parent.
2. The child process has a new program loaded into it.

Operations on Processes (Process Termination)

- A process terminates when it finishes executing its final statement and asks the operating system to delete it by using exit() System calls.
- At that point, the process may return a status value (typically an integer) to its parent process (via the wait() system call).
- All the resources of the process - including physical and virtual memory, open files, and I/O buffers - are deallocated by the operating system.

Termination can occur in other circumstances as well:

- A process can cause termination of another process via an appropriate system call.
- Usually, such a system call can be invoked only by the parent of the process that is to be terminated.
- Otherwise, users could arbitrarily kill each other's jobs.

A parent may terminate the execution of one of its children for a variety of reasons, such as these:

- The child has exceed its usage of some of the resources that it has been allocated. (To determine whether this has occured, the parent must have a mechanism to inspect the state of its children.)
- The task assigned to the child is no longer required.
- The parent is existing, and the operating system does not allow a child to continue if its parent terminates.

Interprocess Communication

- Processes executing concurrently in the operating system may be either independent processes or cooperating processes.
- Independent processes - They cannot affect or be affected by the other processes executing in the system.
- Cooperating processes - They can affect or be affected by the other processes executing in the system.

⇒ Any process that shares data with other processes is a cooperating processes

There are several reasons for providing an environment that allows process cooperation:

- Information sharing.
- Computation speedup → break a task into diff. several subtasks.
- Modularity → divide system into diff. modules to prevent designing system from top-down.
- Convenience → able process run concurrently.

Cooperating processes require an interprocess communication (IPC) mechanism that will allow them to exchange data and information

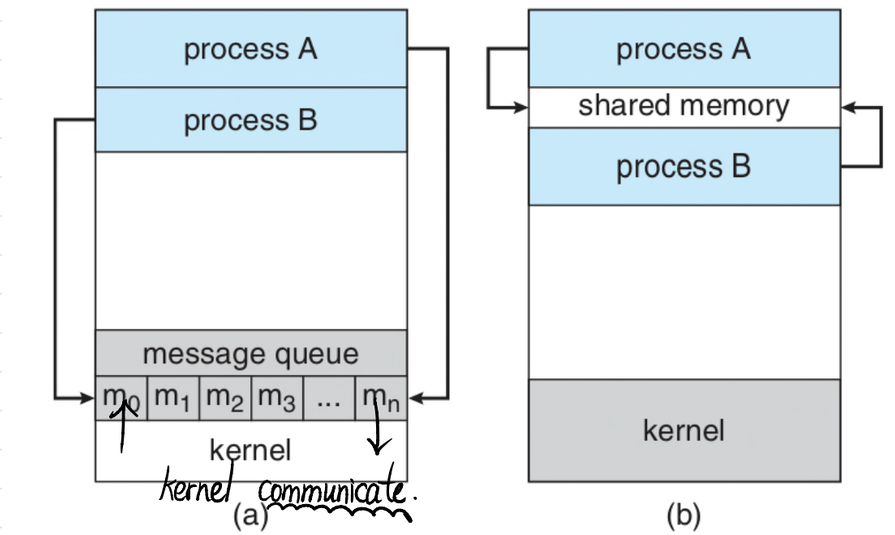
There are two fundamental models of interprocess communication:

(1) Shared memory

→ In the shared-memory model, a region of memory that is shared by cooperating processes is established. Processes can then exchange information by reading and writing data to the shared region.

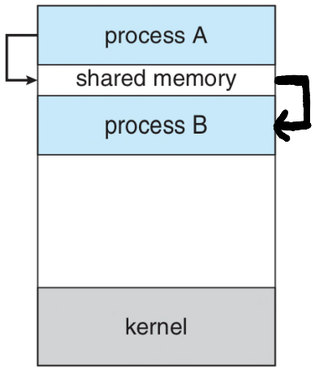
(2) Message passing

→ In the message passing model, communication takes place by means of messages exchanged between the cooperating processes.



message passing　　Shared Memory

Shared Memory System



- Interprocess communication using sharing requires Communicating processes to establish a region of shared memory.
- Typically, a shared-memory region resides in the address space of the process creating the shared shared-memory segment ↳ If Program create it then the shared memory is in Process A

- Other Processes that wish to communicate using this shared-memory segment must attach it to their address space.

  ⇒ the memory now is shared (owned) both A and B.
- Normally, the operating system tries to prevent one process from accessing another processor's memory.
- Shared memory requires that two or more processes agree to remove the restriction.

To have a better understanding:

Produces Consumer Problem

A producer process produces information that is consumed by a consumer process

For example, a compiler (producer) may produce assemble code, which is consumed by an assembler. (Consumer)

The assembler (producer), in turn, may produce object modules, which are consumed by the loader. (consumer)

⇒ The problem : consumer only consumes what producer produces not otherwise, and also they may have action concurrently.

- Solution One : using-shared memory

  → To allow producer & consumer processes to run concurrently, we must must have available a buffer of items (region of shared memory) that can be filled by the producer and emptied by the consumer.

  → This buffer will reside in a region of memory that is shared by the producer and consumer process.

  → A produce can produce one item while consumer is consuming another item

  → The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

Two kinds of buffer

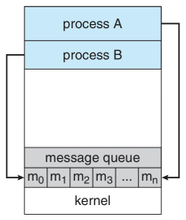
Unbounded Buffer

Places no practical limit on the size of the buffer. The consumer may have to wait for new items, but the producer can always produce new items.

Bounded Buffer

Assumes a fixed buffer size. In this case, the consumer must wait if the buffer is empty, and the producer must wait if the buffer is full.

Message Passing System



Message passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space and is particular useful in a distributed environment, where the communicating processes may reside on different computers connected by a network.

⇒ difficult to reside a shared memory.

- A message-passing facility provides at least two operations:
  - Send Message
  - Receive

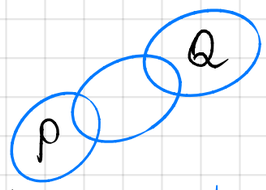
Messages sent by a process can be of either fixed or variable size.

Fixed Size: The system-level implementation is straightforward.
But makes the task of programming more difficult.
(the way and how the task is be down)

Variable Size: Requires a more complex system-level implementation.
But the programming task becomes simpler, (more flexible)

If processes P and Q want to communicate, they must send messages to and receive messages from each other

A communication link must exist between them



This link can be implemented in a variety of ways. There are several methods for logically implementing a link and the send()/receive() operations, like:

- Direct or Indirect Communication
- Synchronous or Asynchronous Communication
- Automatic or explicit buffering

} There are several issues related with features like:
→ Naming
→ Synchronization
→ Buffering

Naming

Processes that want to communicate must have a way to refer to each other. They can use either direct or indirect communication

Under direct communication - Each process that wants to communicate must explicitly name the recipient or sender of the communication.

- Send (P, message) - Send a message to process P
- Receive(Q, message) - Receive a message from process Q

A communication link in this scheme has following properties:

- A link is established automically between every pair of processes that want to communicate. The processes need to know only each other's identiy to communicate.
- A link is associated with exactly two processes.
- Between each pair of processes, there exists exactly one link.

This scheme exhibits symmetry in addressing; that is, both the sender process and the receiver process must name the other to communicate.

$\begin{cases} P \xrightarrow{name} Q \\ Q \xrightarrow{name} P \end{cases}$

Another variant of Direct Communication

Here, only the sender names the recipient; the recepient is not required to name the sender

- Send (P, message) - Send a message to P
- Receive (id, message) - Receive a message from any process; the variable id is set to the name of the process with which communication has taken place.

This scheme employs asymmetry in addressing

- This disadvantage in both these schemes (symmetric and asymmetric) is the limited modularity of the resulting process definitions Changing the identifier of a process may necessitate examining all other process process definitions.

With indirect communication:

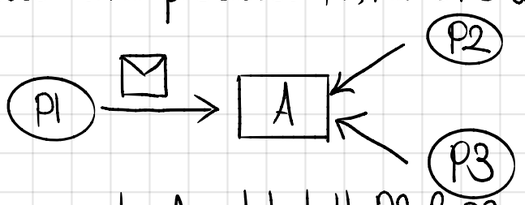
The message are sent to and received from mailboxes or port

- A mailbox can be viewed abstractly as an object into which messages can be placed by processes and from which messages can be removed.
- Each mailbox has a unique identification
- Two processors can communicate only if the processes have a shared mailbox
  - send(A, message) - Send a message to mailbox A.
  - receive(A, message) - Receive a message from mailbox A.

A communication link in this scheme has the following properties:

- A link is established between a pair of processes only if both members of the pair have a shared mailbox.
- A link may be associated with more than two processes.
- Between different each pair of communicating processes, there may be a number of different link, with each link corresponding to one mailbox

Now Suppose that processes P1, P2 & P3 all share mailbox A



Process P1 sends a message to A, while both P2 & P3 execute a receive()

from → Which Process will receive the message sent by P1?

The answer depends on which of the following methods we chose:

- Allow a link to be associated with two processes at most.
  → P1 → P2, P3 only P2 and P1 has the link
- Allow at most one process at a time to execute a recv() operation
- Allow the system to select arbitrarily which process will receive the message (that is either P2 or P3, but not both, will receive the message). The system also may define - scheduling - an algorithm for selecting which process will receive the message (that is, round robin where processes take turns receiving messages) The system may identify the receiver to the sender.

→ A mailbox may be owned either by a process or by the operating system discuss in scheduling

Synchronization

Communication between processes takes place through calls to send() and receive() primitives. There are different design options for implementing each primitive.

Message passing may be either blocking or nonblocking

also known as ↳ synchronous ↳ asynchronous

Blocking send: The sending process is blocked until the message is received by the receiving process or by the mailbox.

Nonblocking send: The sending process sends the message and resumes operation

Blocking receive: The receiver blocks until a message is available

Nonblocking receive The " retrieves either a valid message or a null

The synchronous is called because it waits until {send / receive} the message.

Buffering

Whether communication is direct or indirect, messages exchanged by communicating processes resides in a temporary queue.
Basically, such queue can be implemented in three ways:

#1) Zero capacity: The queue has a maximum length of zero; thus, the link cannot have any messages waiting in it. In this case, the sender must block until the recipient receives the message.
↳ acts like a link

#2) Bounded capacity: The queue has finite length n; thus, at most n messages can reside in it. If the queue is not full when a new message is sent, the message is placed in the queue and the sender can continue execution without waiting. The links capacity is finite, however. If the link is full, the sender must block until space is available in the queue.

#3) Unbounded capacity: The queues length is potentially infinite; thus, any number of messages can wait in it. The sender never blocks

If establish another connection than we will assign new port.

important

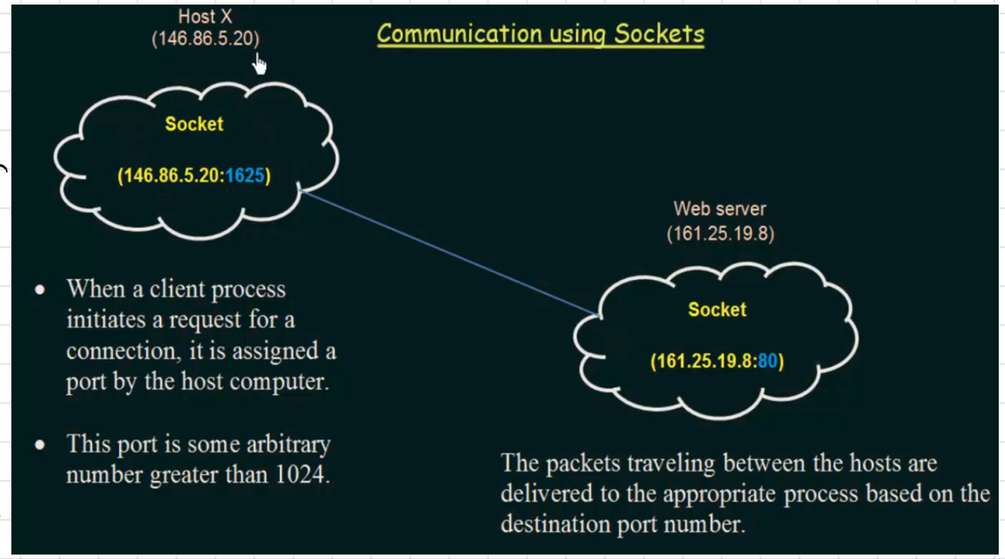
Sockets

Used for communication in Client-Server Systems

- A socket is defined as an endpoint for communication.
- A pair of processes communicating over a network employ a pair of sockets—one for each process.
- A socket is identified by an IP address concatenated with a port number.
- The server waits for incoming client requests by listening to a specified port. Once a request is received, the server accepts a connection from the client socket to complete the connection.

Client ask inform.
↑ give
server

- Servers implementing specific services (such as telnet, ftp, and http) listen to well-known ports
(a telnet server listens to port 23, an ftp server listens to port 21, and a web, or http, server listens to port 80).
- All ports below 1024 are considered well known; we can use them to implement standard services



Threads & Concurrency

Metaphor

A thread is like a worker in a toy shop

- an active entity
  → executing a unit of toy order
- works simultaneously with others
  → many threads executing
- require coordination
  → sharing I/O devices, CPU, memory.

- an active entity
  → executing a unit of toy order
- works simultaneously with others
  → many workers completing toy orders
- require coordination
  → sharing of tools, parts, workstation