

Kernel (Types) → HW

Kernel: The kernel is the core part of the OS. It acts as a bridge between the hardware & software (applications).

→ It ~~manage~~ manage system resources like CPU, memory and device, ensuring everything works together smoothly and efficiently.

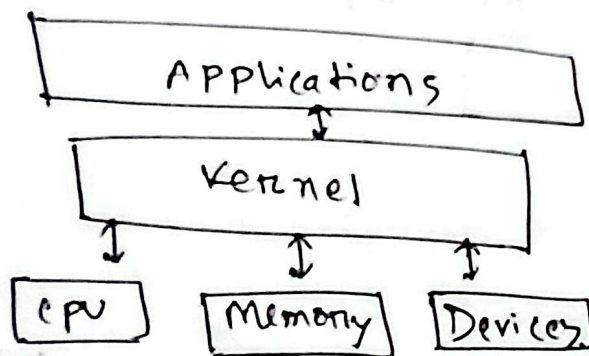


Figure — Kernel

Main Function: ^{load balancing}
→ ~~error handling~~ ^{and termination}
→ Process Management :- scheduling, execution of process
→ Memory " :- allocation/deallocation of RAM
→ Device " :- controlling hardware device
→ File system " :- manage files
→ System calls :-

Types:

① Monolithic kernel: — It is one of the type where all OS services operate in kernel space. ^{where all services run inside the kernel space} ~~large and complex~~. It has huge line of code which is complex.

Ex: Unix, Linux, Open VMS, XTS-400 etc.
^{early macOS.}

Advantages:

- High efficiency & speed
- communication faster | for tight integration
- easy to design
- low latency in handling system calls.

Disadvantages:

- Any bug can crash the whole system (less reliable)
- security risk
- Harder to maintain or update.
- less flexible

② Micro Kernel: It is kernel types which has minimalist approach. It has virtual memory and thread scheduling.

→ Only essential ^{core services run} in kernel space. Other ^{services} devices in user space.

→ It is use in small operating system.
ex: mach, minix, L4, QNX etc.

Advam:

→ more reliable (fault in services do not crash in full system)

→ High modular and flexible

→ Better isolation service

→ Easier to maintain

Disadvam:

→ slower performance due to context switching.

→ more complex communication (I/O)

→ use more resources (cpu, memory)

→ harder to develop OS with it.

③ Hybrid kernel : It is the combination of monolithic and micro kernel. It has fast and stability like microkernel and modular like monolithic.

ex: window NT, macOS (XNU), BeOS.

Advantages:

→ Good performance (due to fewer context switching)

→ more reliable than pure monolithic.

→ Flexible : can add/remove services.

Disadvantages:

→ more complex to design & maintain.

→ larger size than microkernel.

→ security risk due to monolithic parts.

① Exokernel: It is the type of kernel which follow end-to-end principle.

- Very minimal kernel ✓
- provides direct access of hardware resources to applications.

Ex:

ExOS, Nemesis, MIT

Adv:

- very flexible (application can customize)
- high performance
- strong security
- modular and efficient.

Dis:

- complex design and development
- application development is harder
- less practical.
- debugging more difficult.

⑤ Nano Kernel: It is type of kernel with extremely small kernel, only provides hardware abstraction (No-system services)

→ similar to micro kernel, but even smaller

Ex: EROS

Advantages:

- very small & lightweight.
- high modularity and customizable
- ~~secure~~ secure (small attack surface)
- portable across hardware.

Disadvantages:

- very limited functionality.
- harder to develop and maintain.

Kernel mode

- Privilege level - Full access to CPU, memory and hardware.
- Hardware can ^{directly} access I/O devices, memory, CPU instructions.
- Core of the OS runs here
- ~~Exec~~ Execute and handles system call.
- CPU can execute all type of instruction
- can access all memory (kernel + user space)
- If kernel mode can crashes then the whole system have crash
- debugging harder
- Enforces protection rules and permission
- Ex: OS kernel, device driver, memory manager

User mode.

- limited privilege
- cannot directly access hardware.
- user program run here
- must request services via system call.
- only non-privileged instructions are allowed.
- can only access its own memory space
- crash in user mode only.
- easier
- subject to restrictions set by kernel
- Ex: ms word, chrome, text editor.

What is OS?

A program that acts as an intermediary b/w a user of a computer and the computer hardware.

OS goals:

→ Execute user programs and make solving user problems easier.

→ use the computer hardware in an efficient manner.

W OS is a resource allocator.
• manage all resources

W OS is a control program
controls execution of programs to prevent errors and improper use of the computer.

W Note → KB : 1024 bytes

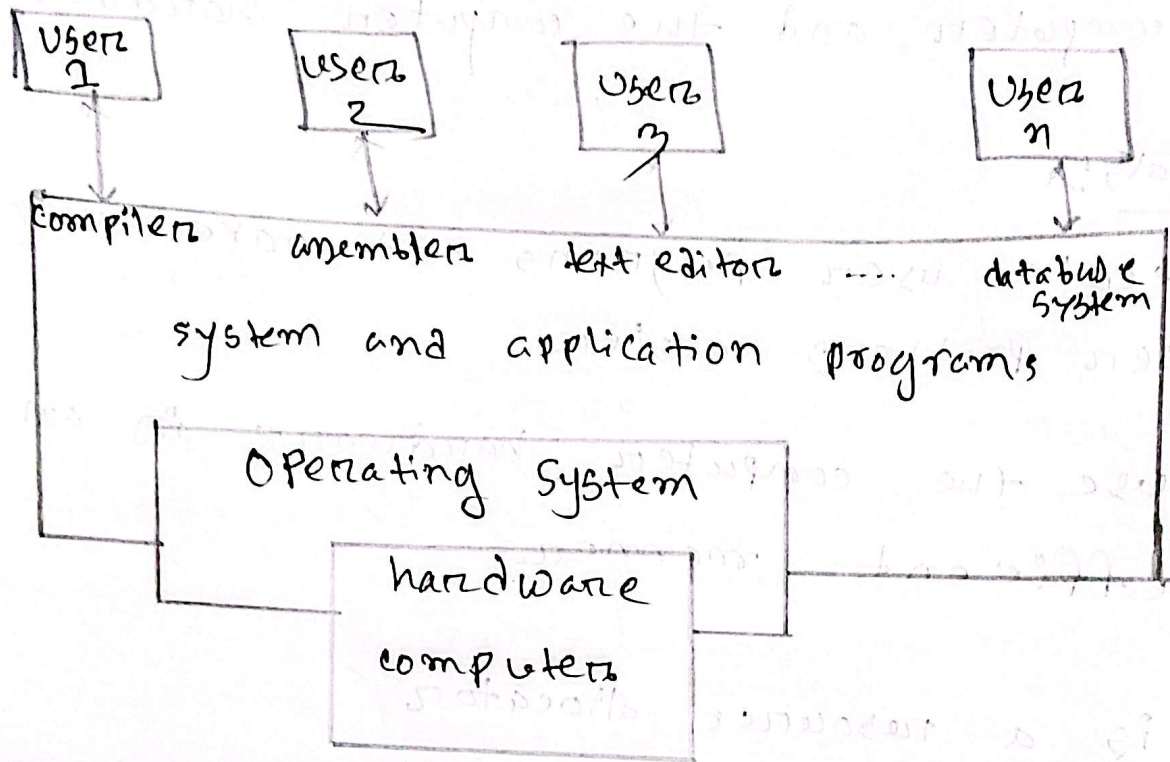
MB : 1024^2

GB : 1024^3

TB : 1024^4

PB : 1024^5

Discuss relationship between hardware, OS, system program and application program:—



Hardware: the physical component of the computer. such as CPU, I/O devices, memory, storage etc.

→ hardware only understand machine language (binary 0s and 1s)

OS: The intermediate layer between hardware and software.

→ manage hardware resources and control CPU scheduling, memory management, I/O operations etc.

system program:

→ special utility programs that work closely with the OS.

→ Provide basic function such as file management tools, compilers, assemblers,

text editor, database system etc.

→ help user & developers interact with the OS more easily.

Application Programs:

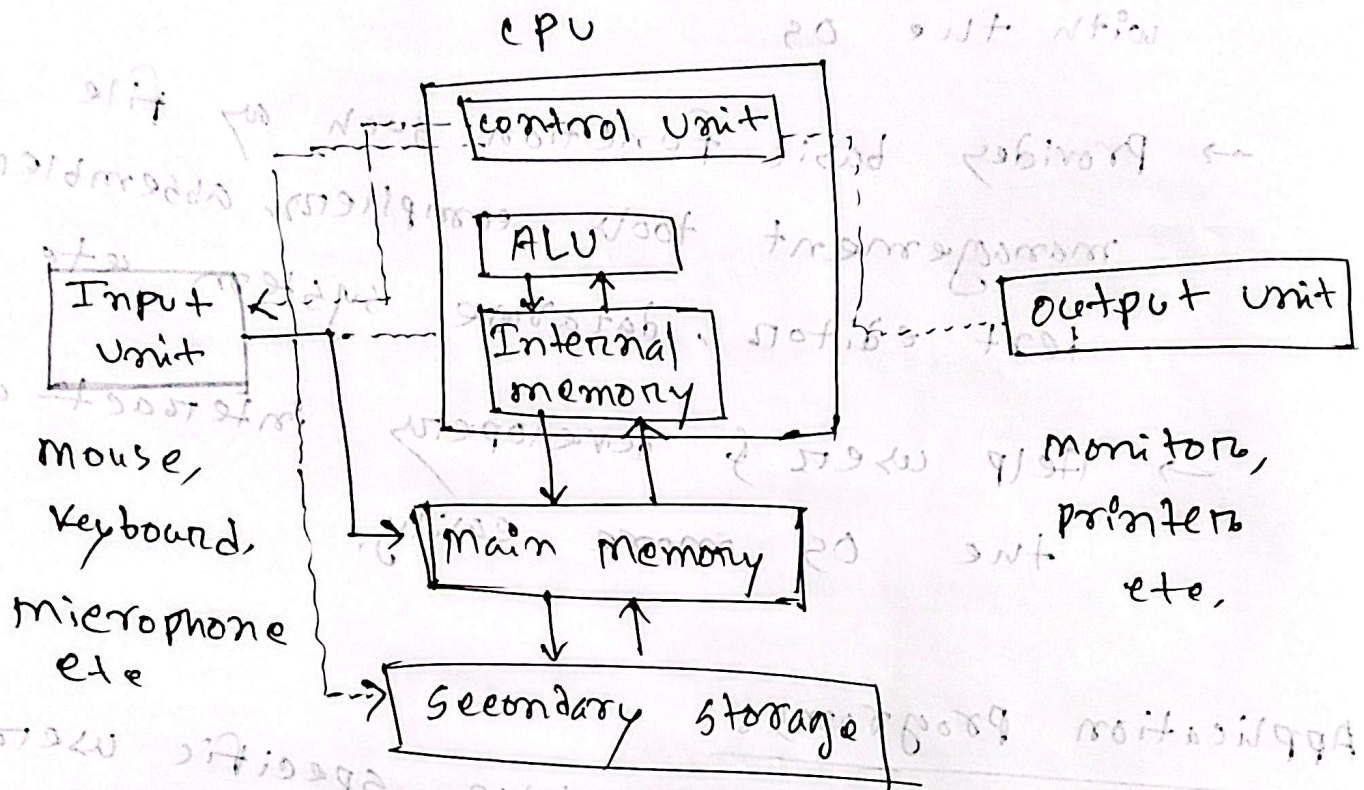
→ software design to solve specific user problem or performing task.

ex: ms word, Browser, media player, Game

Users: people, machine, other computer.

Output: Display or browser the results
of processing to the user
on monitors, printers, etc.

main components of in the logical organization of a computer system :-



Input: Accept data and instruction from the users or external devices.

Ex: Keyboard, mouse, scanner.

Output: Display or provides the results of processing to the users.

ex: Monitor, printer, speaker.

Memory Unit:

store data, instructions and intermediate results.

Types — Primary memory (RAM, ROM) and secondary ~~memory~~ storage (SSD, HDD)

ALU: perform all arithmetic operation and logical operation.

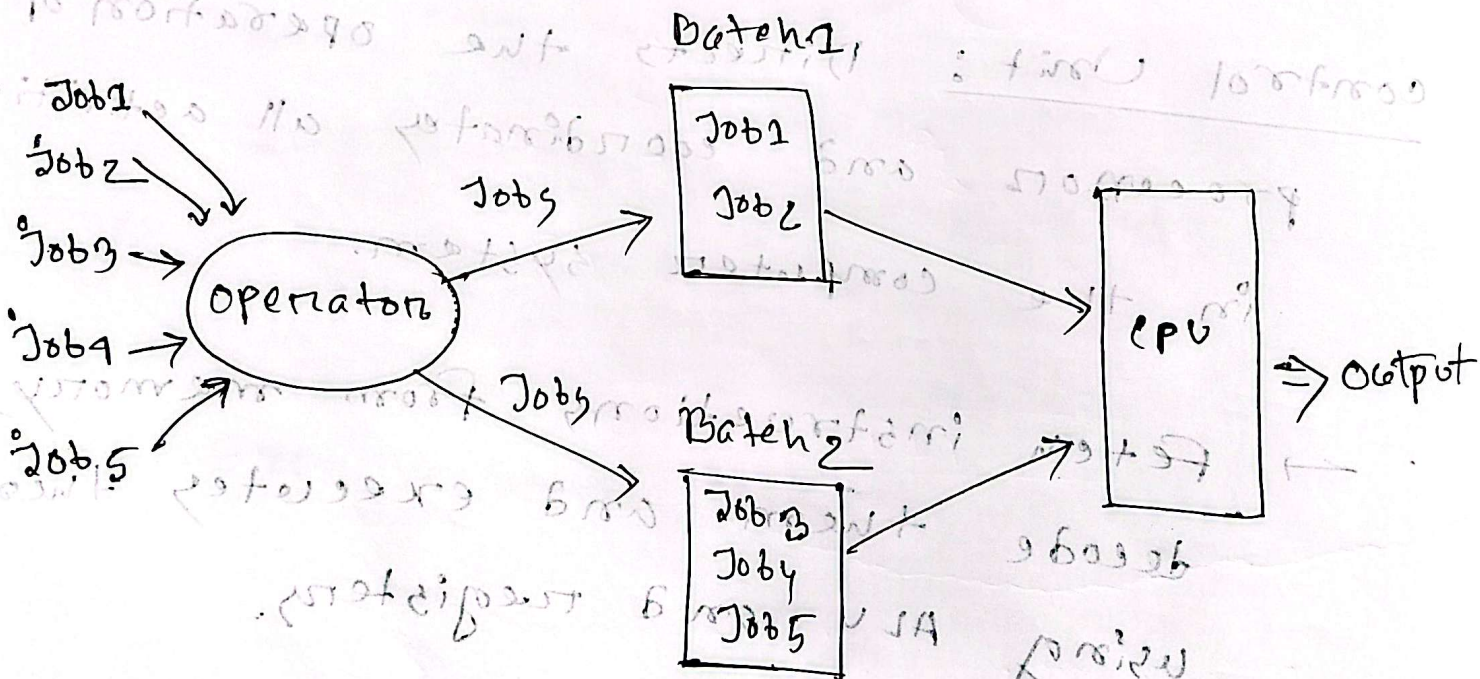
Control Unit: Directs the operation of the processor and coordinates all activities in the computer system.

→ Fetch instructions from memory, decode them, and execute them using ALU and registers.

Multi programming / Batch programming :-

Batch operating system:

This type of OS doesn't interact with the computer directly. There is an operator which takes similar jobs having same groups requirements and group them into batches.



purpose:

- Efficient CPU utilization (keep the CPU busy)
- Automate job processing (Execute similar job)
- Simplify job management (Group job into batch)

problem:

→ lack of ~~user~~ interaction between the user and the job

→ CPU is ~~open~~ often idle, because the speed of I/O slower than CPU.

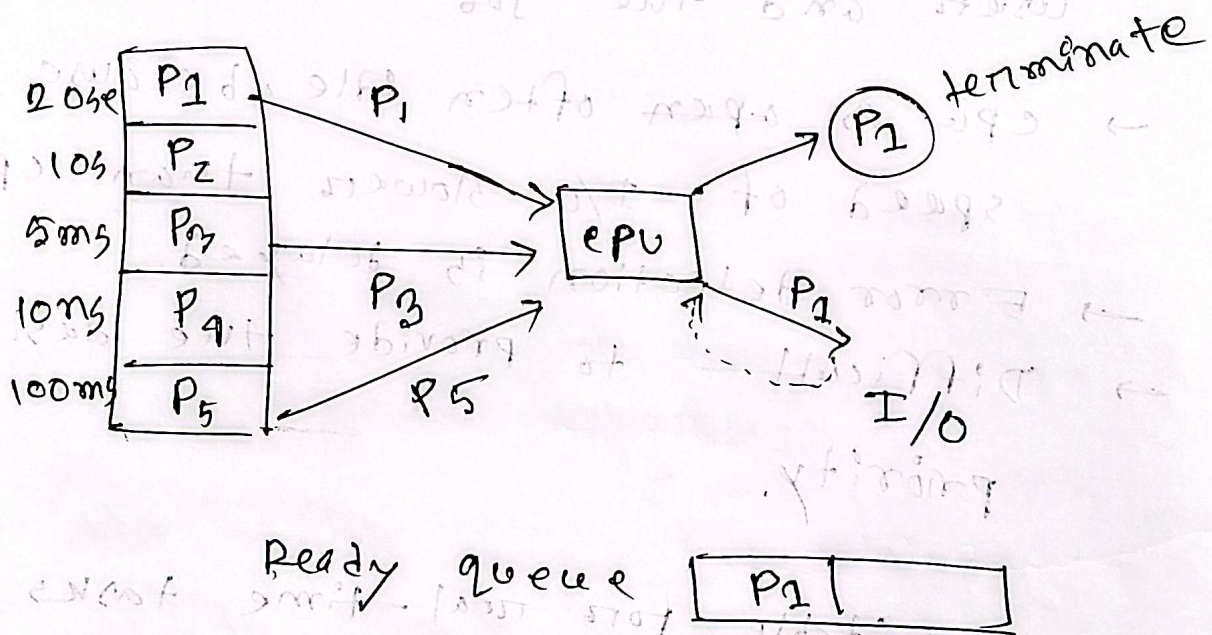
→ Error detection is delayed.

→ Difficult to provide the desired priority.

→ Not suitable for real-time tasks

✓ Multiprogramming:

multiprogramming is a technique in OS where multiple programs (jobs) are loaded into main memory simultaneously, allowing a single CPU to switch between them while some are waiting for I/O operation, thereby improving CPU utilization. It's also non-primitive.



→ firstly P1 processing, but somehow P1 going for I/O operation then P2 loaded in CPU. But, when P1 back then P1 start processing untill it's complete.

Advantage:

- High CPU utilization - CPU never goes to idle
- efficient memory utilization
- less waiting time than Batch OS
- Better response time.

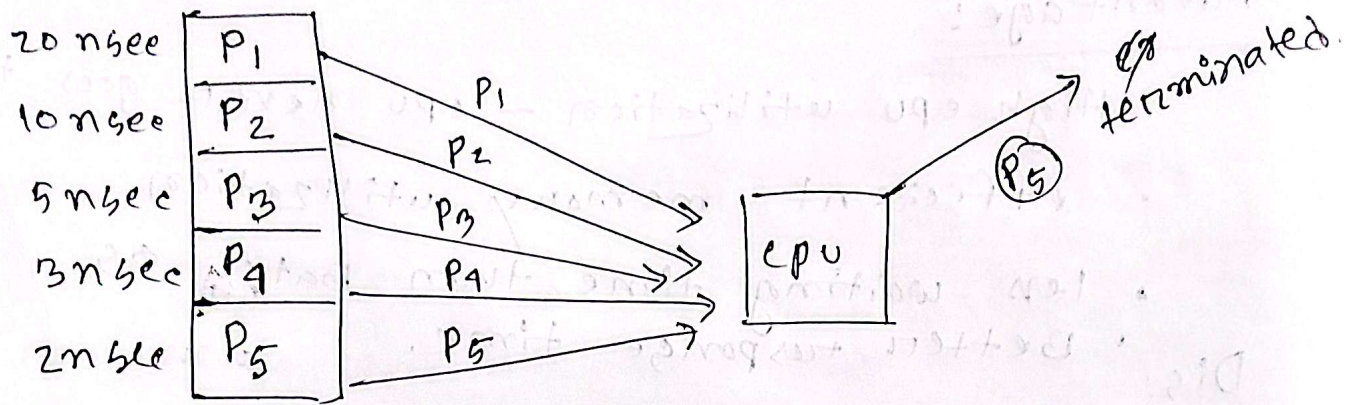
Dis:

- multiple job ready to run simultaneously
- Users cannot interact with a job while it is executing.
- programmers cannot change a program during its execution.

W Multitasking / time sharing :-

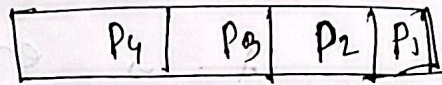
is an operating system technique where multiple users or programs share the CPU simultaneously by dividing CPU time into small slices called time quanta.

- response time should be < 1 second
- Each user has at least one program executing in memory $\Rightarrow$ process
- Time sharing



cpu time = 2 nsec

Ready queue



↓ ↓ ↓ ↓ decrease time
1 3 8 18 nsec

Advantage :

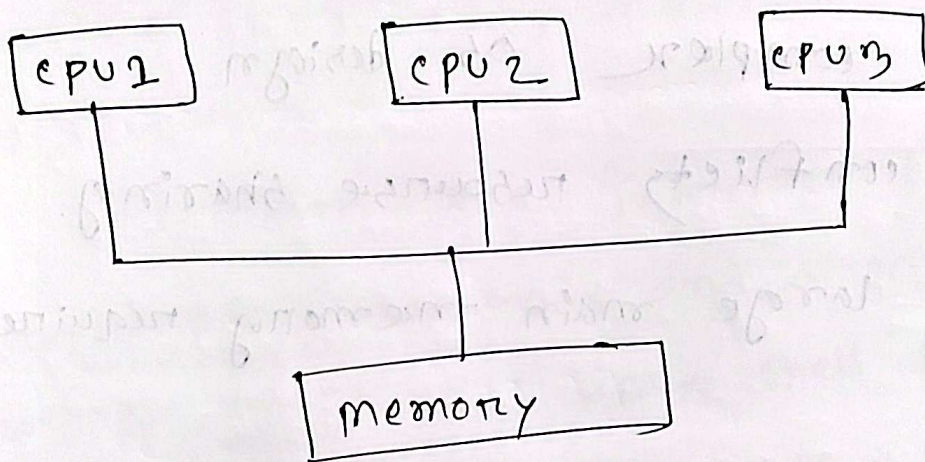
- Allow multiple users to use the system concurrently
- efficient cpu utilization
- Quick response time.

Dis :

- complex scheduling
- Not-suitable for real-time critical task

Multiprocessing: is an OS technique where two or more CPUs work together simultaneously to execute multiple processes.

→ It also refers to a computer system's ability to support more than one process at the same time.



Multiprocessing are two types:-

1. Symmetric:

Each processor performs all tasks.

2. Asymmetric: This system uses a master-slave relationship, where one processor (the master) manages the others.
→ Each processor assigned a specific task.

Advam:

- more reliable (if one epe fail, other continue)
- Enhance throughput; more faster process
- More economic system.

Disadvan:

- more expensive hardware
- complex OS design
- conflict resource sharing.
- large main memory required.

operating system service :-

An OS provides an environment for execution of programs and services to programs and users.

Five services are :-

1. User Interface: ~~All~~ Almost all OS have a user interface (UI). This interface can take several forms -

(i) Common-line-Interface (CLI), which ~~uses~~ uses text commands and a method for entering them. Ex: Linux shell, MS-DOS.

(ii) Batch-Interface, which commands and directives to control those commands are entered into files, and those files are executed.

(iii) most commonly, Graphical-user-Interface (GUI). Ex: Windows, Mac OS, Linux GNOME.

Convenience for users: users can interact with the system using simple command or graphical menu. No need understand hardware-level operation.

② Program Execution:

The OS is responsible for loading programs into memory, executing them and managing their termination.

convenience: users can run multiple programs without manually managing CPU scheduling or memory allocation.

③ I/O operation: A running program may require I/O, which may involve a file or I/O devices. (like monitor, keyboard, printer and disks)

convenience:

users and programs can perform I/O without needing to directly control hardware, which is complex and error-prone.

④ File-system Manipulation:-

The file system is of particular interest. The programs need to read and write files and directories, create & delete them, search them, list file information, permission management.

convenience: Users can store & retrieve data easily without worrying about the physical storage layout or disk management.

⑤ Error Detection:

The OS need to be constantly aware of possible errors. Errors may occur in the CPU and memory hardware, I/O devices, in users program.

→ For each type of error, OS should be take the appropriate action to ensure correct and constant computing.

convenience: users are informed of errors, and the system can take corrective actions to maintain reliability and prevent crashes.

⑥ communication: process may be exchanging information on the same computer or between computers over a network.

→ communication may be shared memory resource or passing message.

Another set of OS services which are impossible for user-level programs. They are given below.

(i) Resource allocation: The OS distributed resource like CPU, memory, I/O devices among multiple users or programs.

For example, CPU scheduling routines decide which job run next based on CPU speed while other routines manage printers, USB drives and storage.

~~why users can~~

Impossible for user-level: Program cannot safely control shared resources or coordinate multiple jobs. Only OS can manage this.

(ii) Accounting: The OS keeps track of how much and what type of resources each user consumes, for billing or usage statistics.

Impossible for user: Only the OS can monitor all users and programs at a system-wide level.

(iii) Protection & security: The OS prevent program from interfering with each other & protect data from unauthorized access using access control.

Impossible for user: User cannot enforce security across the system or control hardware access, only OS has control privilege.



Chapter - 3

Processes

✓ A process is a program in execution.

→ It is an active entity that include not only the program code but also its current state, memory, cpu registers and resource needed to execute.

→ The program code, called text section.

→ data section containing global ^{and static} variables

→ heap containing memory

dynamically allocated during run time

→ stack contain temporary data.

such as func parameters, return address, local variables.

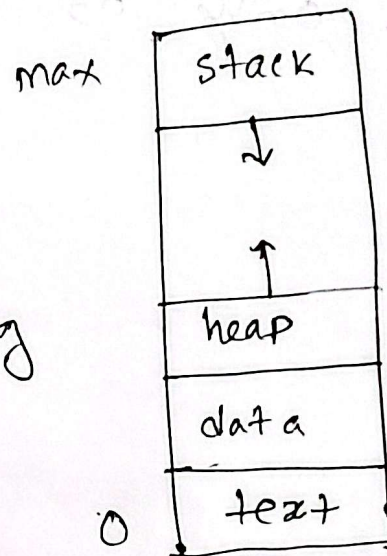


Fig:- Process memory

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.

~~State~~ → states :-

- ① New : The process is being created
- ② Running : Instructions are being executed
- Waiting : The process is waiting for some event to occur.
- ③ Ready : The process is waiting to be assigned to a processor
- Terminated : The process has finished execution.

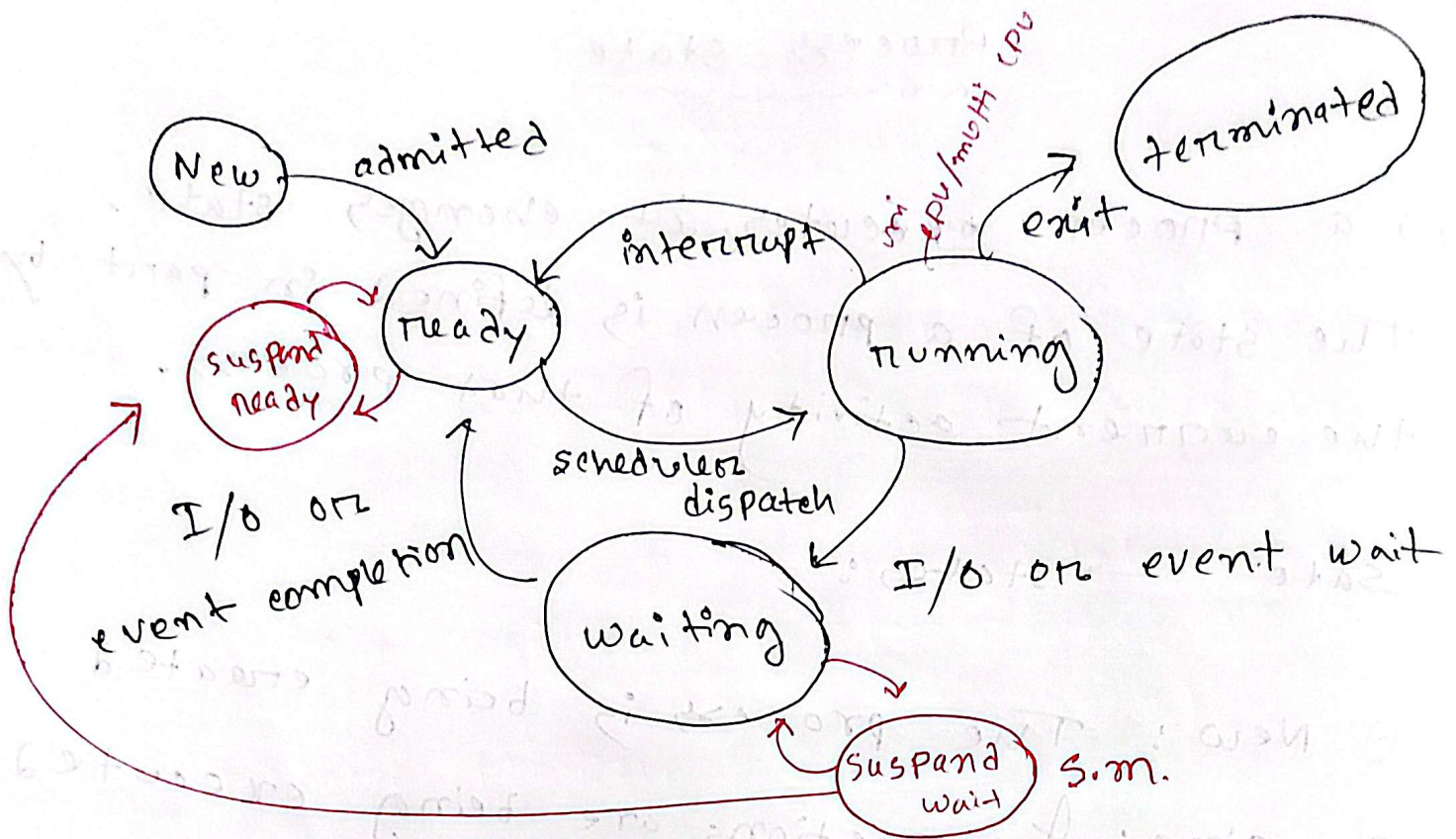
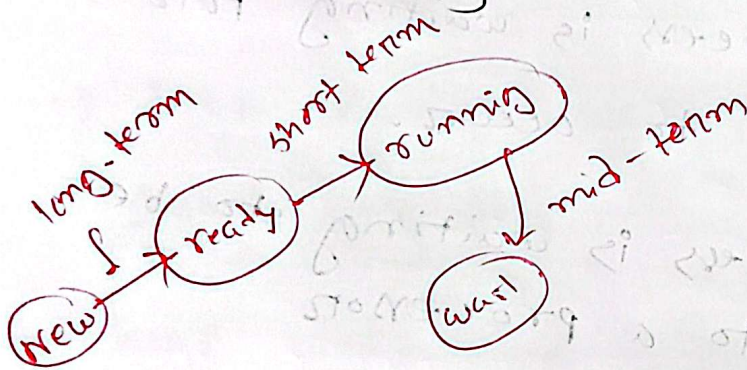


Fig:- process state.

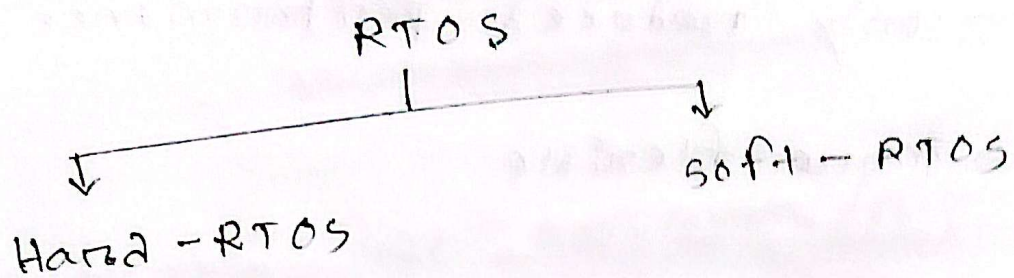


✓ A real time OS is designed to handle task that must be executed within a specific time limit (deadline)

Real time OS → (RTOS)

→ Real-time OS intended to serve real time applications.

→ Response should be guaranteed within a specified timing constant or system should meet the specified deadline.



Hard-RTOS: This type of ~~as~~ RTOS can never miss its deadline.

→ task must be complete within the deadline

→ Even a small delay is considered a system failure.

→ used in safety critical-system

- ex:
- Airbag system in cars
 - medical devices (pacemaker, ICU)
 - Industrial Robot and control system.

Soft-RTOS: This type of RTOS can miss its deadline occasionally with some acceptable low probability.

- Task should be completed within the deadline but occasional delays are acceptable.
- Delays does not cause total system failure only reduced performance.
- more flexible

Ex: .. multimedia system (video/audio stream)
• online transaction (banking, E-commerce)
• Virtual reality and gaming.

Hard - RTOS

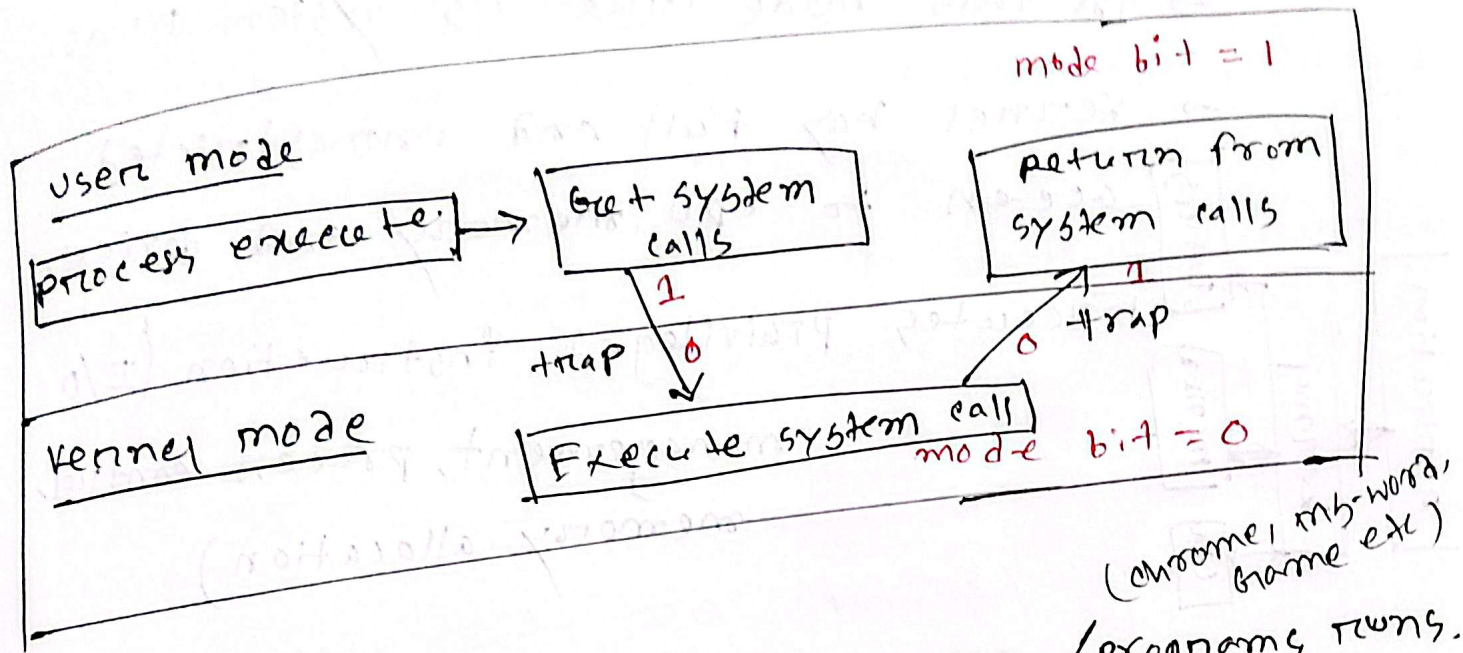
- must always be met at deadline
- delay not acceptable
- life - safety criticality

ex: Airbag, Pacemaker, Robots

Hard - RTOS

- can be missed occasionally
 - some time accept
 - performance criticality
- ex: Streaming, gaming, Banking.

user mode and kernel mode: —



User mode: In this mode Application / programs runs.

- application cannot access directly hardware (cpu, memory etc)
- cannot modify OS memory.
- if they need hardware services, they must request the OS through a system call.

→ safe mode (if a program ~~fails~~ crashes, the OS remain safe)

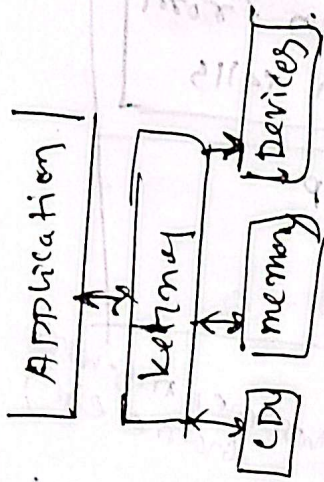
ex: run, ms word, chrome, game etc.

- if a user mode program crashes, it does not crash the entire OS.

Kernel mode:

→ In this mode the OS system runs.

→ Kernel has full and unrestricted access to CPU, memory and devices.



Executes privileged instruction (I/O management, process control, memory allocation)

→ Risky mode → if something goes wrong, the whole system can crash.

ex: when we save a file, the OS goes to kernel mode to actually write data to the disk.

How mode bit works: The mode bit is a special CPU bit that tells whether the CPU is running in user mode (0) or kernel mode (1).

binary value :-
1 = user mode
0 = kernel mode.

Working:

W when a user program runs $\rightarrow$ the mode bit = 1 (user mode)

$\rightarrow$ Program can only execute non-privileged instruction

$\rightarrow$ If it needs I/O, it must request the OS through a system call.

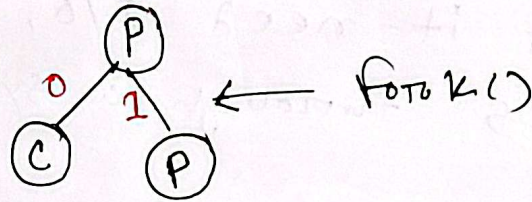
W When a system call then the CPU switches the mode bit = 0 (kernel mode)

$\rightarrow$ Now the ^{OS (kernel)} OS takes over, perform privileged operation (like I/O, memory etc)

W After the OS finishes $\rightarrow$ The mode bit is reset back to 1 (user mode) & control returns to the program.



Fork() system call: is used for creating a new process, which is called child process. Here child process will run concurrently with its parent process.



→ fork() returns an integer value, where :-

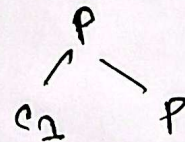
Negative value: creation of a child process was unsuccessful.

Zero: Returned to the newly created child process

positive value: Return to parent or caller.

Ex:

```
int main()
{
    fork();
    printf("Hello");
}
```



→ output: Hello
Hello

Ex:

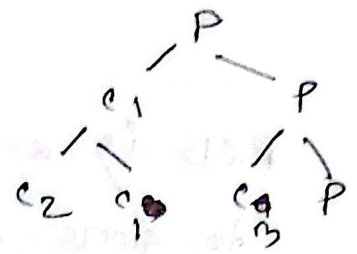
```
int main() {
```

```
    fork();
```

```
    fork();
```

```
    printf("Hello");
```

```
}
```



→ output :

Hello

Hello

Hello

Hello

→ fork() works only linux/unix, macOS, BSD

→ No. of process = 2^n ; n = no. of fork()

∴ $2^2 = 4$ process

child process = $2^n - 1$

= $2^2 - 1 = 3$ child

process

PCB → Process control Block.

PCB is a data structure used by the OS to store all the important information about a process.

→ Every process have an unique process ID.

Process ID
Process state
Process number
Program counter
Register
memory limits
list of open files
...

→ Process state:- new, ready, running, waiting etc.

→ Program counter: Address of the next instruction to be executed.

Register: Store the current value of CPU register when a process is not running.

→ Ensure process can resume correctly after being paused.

memory limit: store the memory allocation boundaries of a process. for protection.

→ The OS uses this to ensure the process stays within its assigned memory & doesn't access other process memory.

List of open files: contains a list of ^{all} open files and I/O devices currently open by the process

→ Helps the OS track I/O operation for the process.

Three Types of scheduler :

① long-term scheduler :

→ Decides which processes are admitted into ready queue for processing.

→ control degree of multiprogramming

→ run less frequently. (s. min)

purpose: select job from job pool $\rightarrow$ ready queue.

ex: Batch system.

② medium-term scheduler:

$\rightarrow$ Temporarily remove (suspend) or resume process from main memory.

$\rightarrow$ used to improve performance and balance CPU & I/O.

$\rightarrow$ Runs occasionally.

ex: swapping process in/out of memory

③ short-term scheduler: (CPU)

$\rightarrow$ Decides which ready queue process going to running state for processing.

$\rightarrow$ run very frequently (ms)

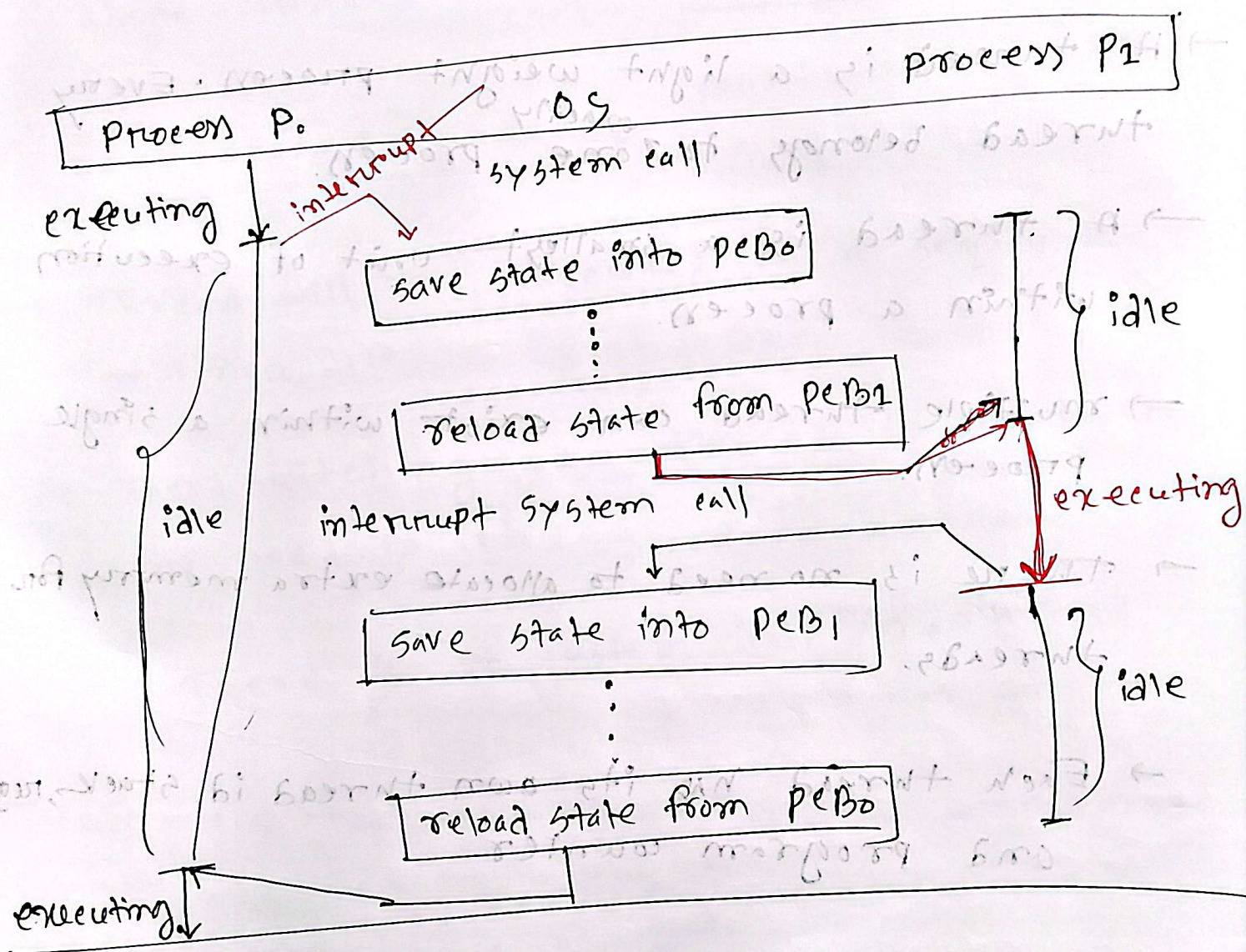
Goal: maximum CPU utilization and response time.

ex: time-sharing system, real-time systems

context switch

when cpu switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch.

→ context of a process represented in the PCB.



cpu switch.

Threads

Process → heavy weight

Thread → light weight

A process is a program that perform a single thread of execution.

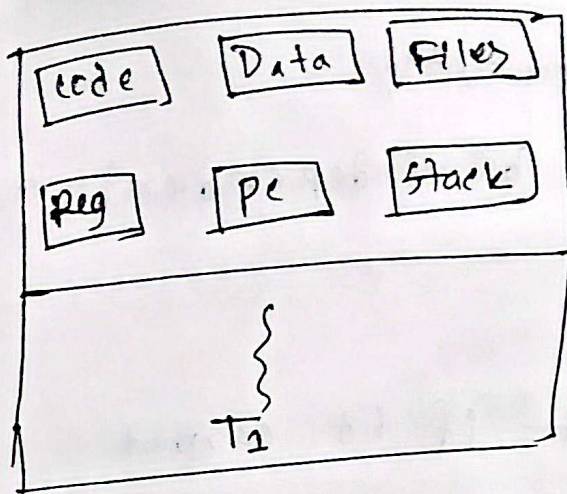
→ A thread is a light weight process. Every thread belongs to ^{exactly} one process.

→ A thread is a smallest unit of execution within a process.

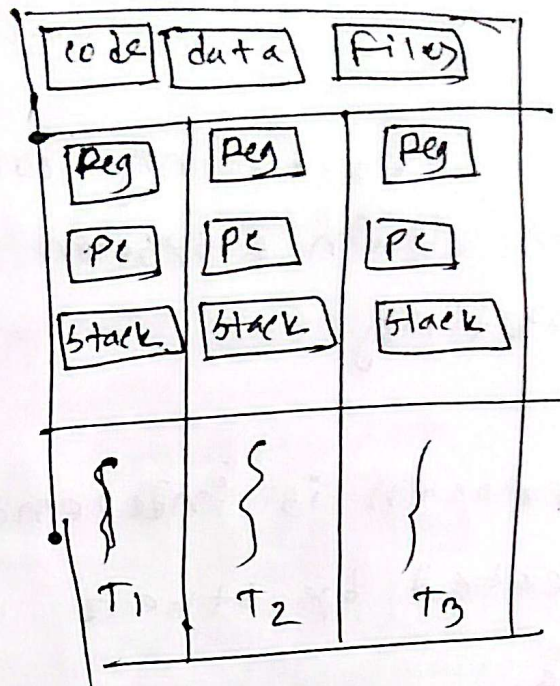
→ multiple thread can exist within a single process.

→ There is no need to allocate extra memory for threads.

→ Each thread has its own thread id, stack, register and program counter.



single-thread process



multi-thread process

→ Thread will be executed in a sequence manner.

→ For switching one thread to another no need to interact with OS.

→ There is no system call in a thread.

... (faint, mostly illegible handwritten notes) ...

Interprocess communication

→ Executing concurrently
Processes within a system may be independent or cooperating.

→ A process is independent if it cannot be affected by other processes, including sharing data.

→ A process is cooperating if it can be affected by other processes.

Reason for cooperating process:

→ Information sharing

→ computational speed up

→ modularity

→ convenience

Cooperating processes need interprocess communication (IPC) mechanism that will allow them to exchange data & information

→ Two models of IPC

① shared memory ② message passing

① shared - memory model: multiple process share a common memory area for direct data access.
→ In the shared memory model, A region of memory is created that is shared by cooperating process is established.

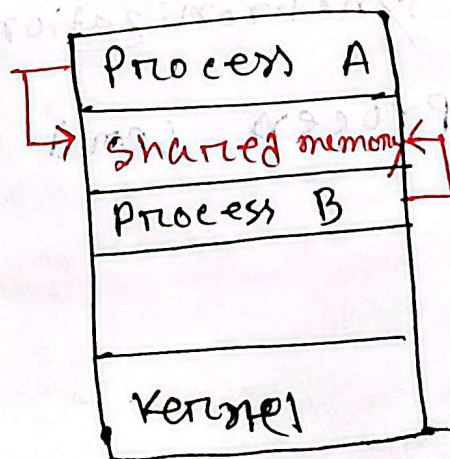
→ Process can then exchange information by reading and writing to the shared region.

→ communication is very faster than message passing, because process directly access memory without going through the OS.

→ However, synchronization is required to avoid race around condition or data inconsistency.

ex: Two process sharing a buffers in memory.

Figure:



② Message Passing Model:

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

→ process do not share memory

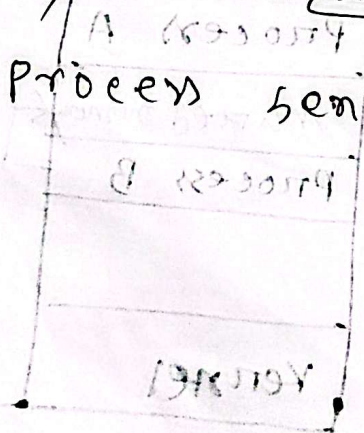
→ communicate only by sending and receiving messages.

→ The OS provides messages queue or communication primitives (send(message) and receive(message)).

→ communication is ~~slower~~ slower, because every message must go through the OS.

→ but it is safer and simpler, because the OS handles synchronization.

ex: One process send a message, another receive it.



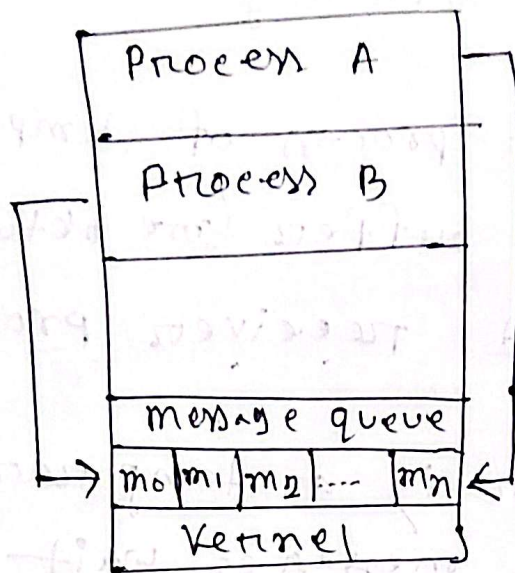


Fig: message passing.

✓ Implementation of communication link

→ Physical: ~~direct~~

- Shared memory
- Hardware bus

(~~Physical~~) Network

→ Logical:

- Direct or indirect
- Synchronous or asynchronous
- Automatic or explicit buffering

Direct communication:

send (P, ~~receive~~ message) — send a message to process P

receive (Q, message) — receive a message from process Q

Buffering

Buffering is the process of temporarily storing messages in a buffer ~~time~~ between a ~~buffer~~ sender and receiver process.

→ buffer is a temporary memory area where messages wait until the receiver is ready.

→ This allow sender & receiver to not be active at the same time.

Types: Three

① Zero capacity: (no buffering)

→ The queue has maximum length of zero.

→ The link cannot have any ~~link~~ message waiting in it.

→ Sender must block until the receiver receive the messages.

→ This is essentially a rendezvous.

i) Bounded capacity (finite buffer)

- The queue has finite length n .
- If the queue is not full, when a new message is sent, the ^{message is} placed in the queue and sender can continue execution without waiting.
- If the link is full, the send must block until space is available in the queue.

Limited space = limited message.

iii) Unbounded capacity (infinite buffering)

- The queue length is considered infinite.
- The sender can ~~receive~~ keep sending message without ever blocking.

→ The sender never block.

→ The receiver consumes messages at its convenience.

So, the zero capacity case is referred to as message system with no buffering. Other cases are referred for automatic buffering.