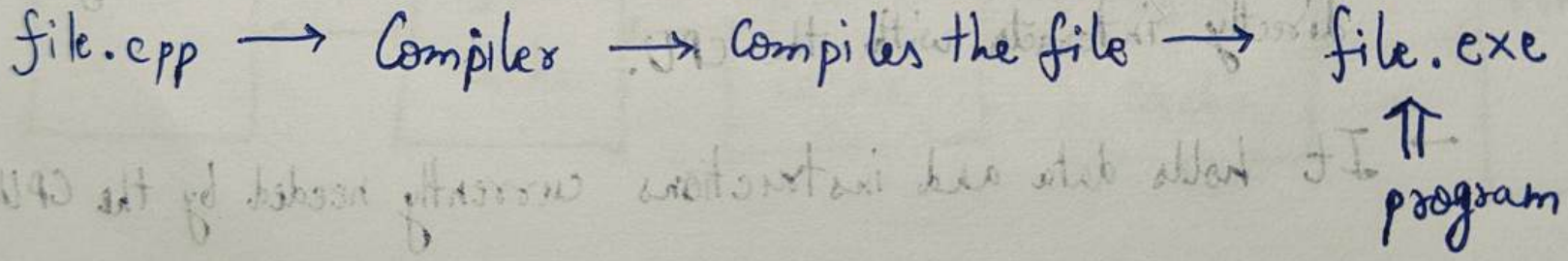
What is a program ?



→ Program is a compiled code that is ready to execute

→ Programs are stored on disk (secondary storage device).

eg :- Pubg is a program stored

↓

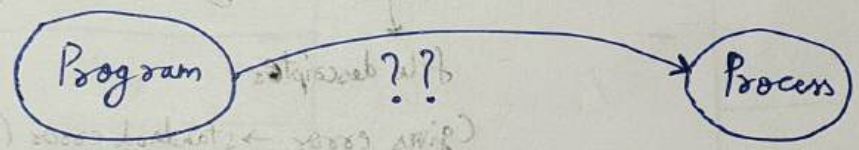
After clicking Pubg to open it, the OS converts the program to process.

What is a process?

→ Program under execution is a process

Why process?

→ Process is a way by which user can get the work done by the CPU.



How OS creates a process?

Step-1: Load the program & static data to memory (disk → RAM)

↓

used for initialization

eg :- char *name = "Laksh";

⇓

when converted to process the name variable is created and initialized with "Laksh".

⇓

Step-2 : Allocate runtime stack

→ Stack is a part of memory used for local variable, function argument & return value.

Step-3: Allocate heap

→ Heap is a part of memory used for dynamic allocation.

Step-4: I/O tasks

→ Allocate handles as to where to shows output, from where to take input, etc.

eg:- In UNIX-file descripters

i/p → handle

o/p →

error → handle

'OR'

in cpp

fprintf (stderr , "hello");

file descripter

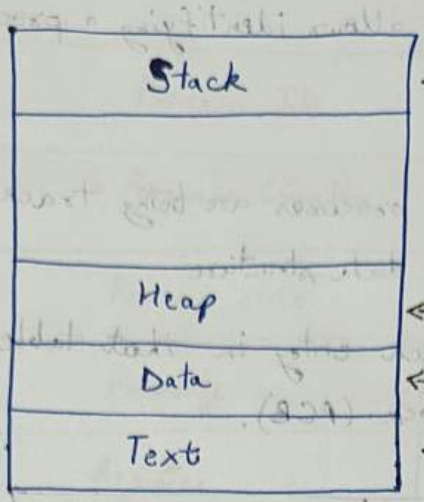
(gives error → standard errors (stderr))

Step-5: OS hands off the control to main()

As in cpp our program starts from main only as OS only knows about main() so it calls main() and gives the control so that it can execute.

So in main() in the end we return 0, this is done so that OS knows that the execution of program was successful or not.

Architecture of process



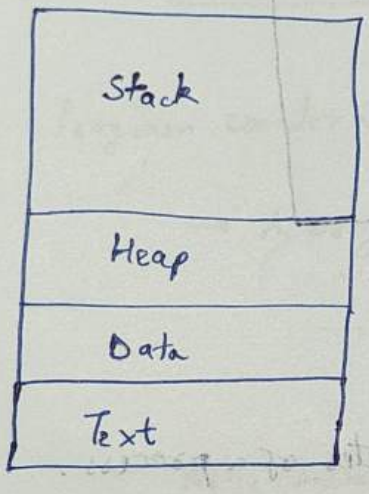
← Local variables, function arguments & return values

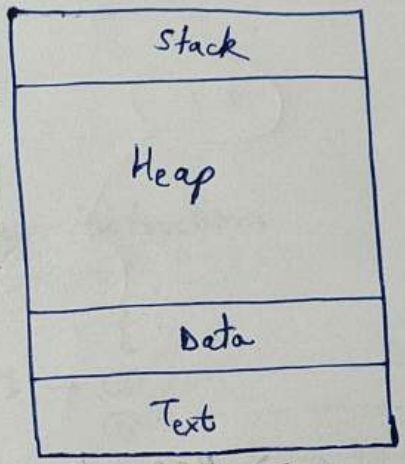
← Dynamically allocated variables

← Global & static data

← Compiled code loaded from disk

- Stack overflow
- Out of Memory errors





→ When the heap or stack memory touch the other's line or can say we cannot give more space to stack or heap as there's no space left then these errors occurs. It helps OS know that there was some problem while executing them.

→ To avoid these errors

① Stack overflow ← set base condition for recursive calls

② Out of memory ← deallocate memory to unnecessary objects.

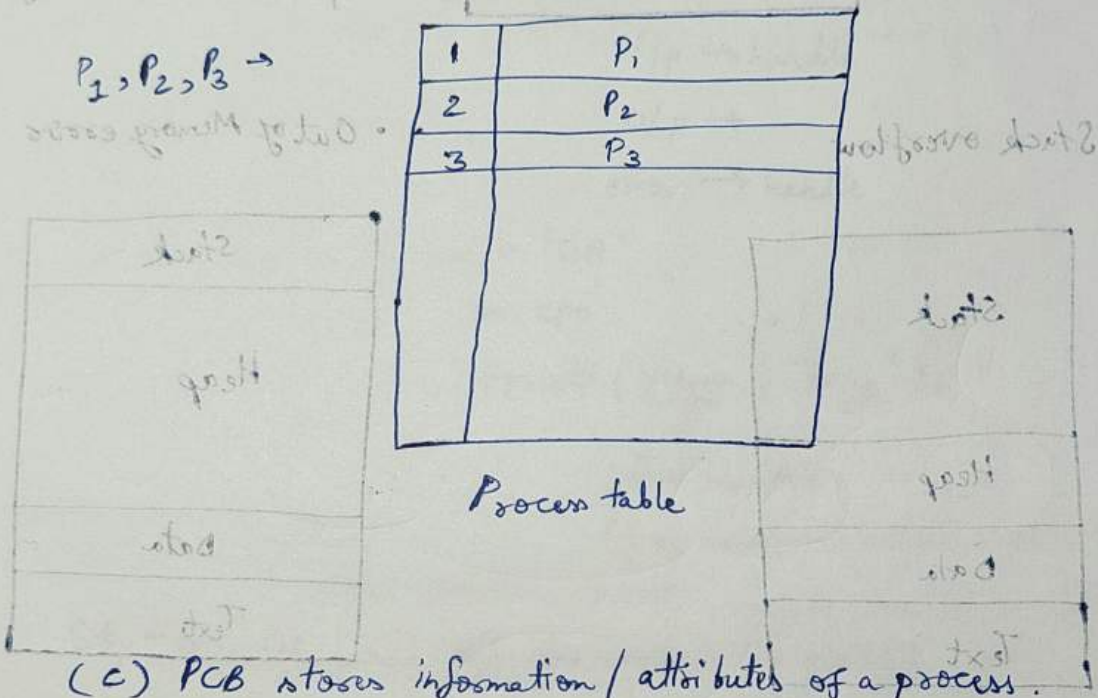
Attributes of Process

(a) Feature that allows identifying a process uniquely.

(b) Process table

(i) All processes are being tracked by OS using a table like data structure

(ii) Each entry in that table is process control block (PCB).



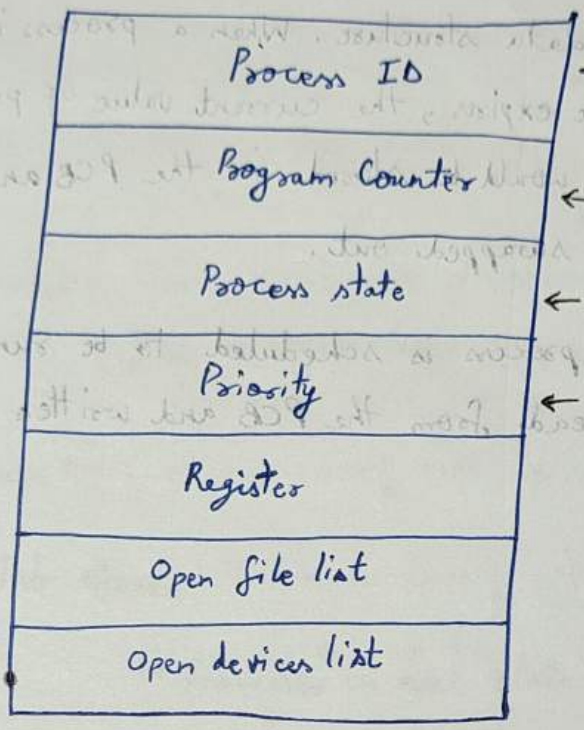
(c) PCB stores information / attributes of a process.

(i) Data structure used for each process that stores information of a process such as process id, program counter (PC), process state, priority, etc.

Process table: The process table is a data structure maintained by the OS. It is a table in main memory that stores information about every active process in the system.

The OS uses process table to manage and keep track of all processes currently in any state.

Process control block (PCB)



← Unique identifier

← Next instruction address of the program

← Stores process state

← Based on priority a process gets CPU time

- Program counter (PC)

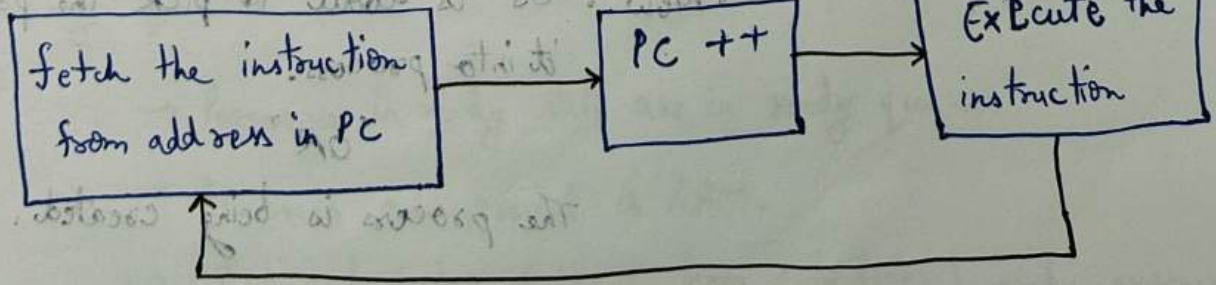
→ A program can have many instructions

PC → Instruction 1

1 2 3 4 5 6

After Instruction 1 is fetched PC ++ as PC points to next instruction that will be executed

{ Instruction register (IR) holds the instruction currently being executed }

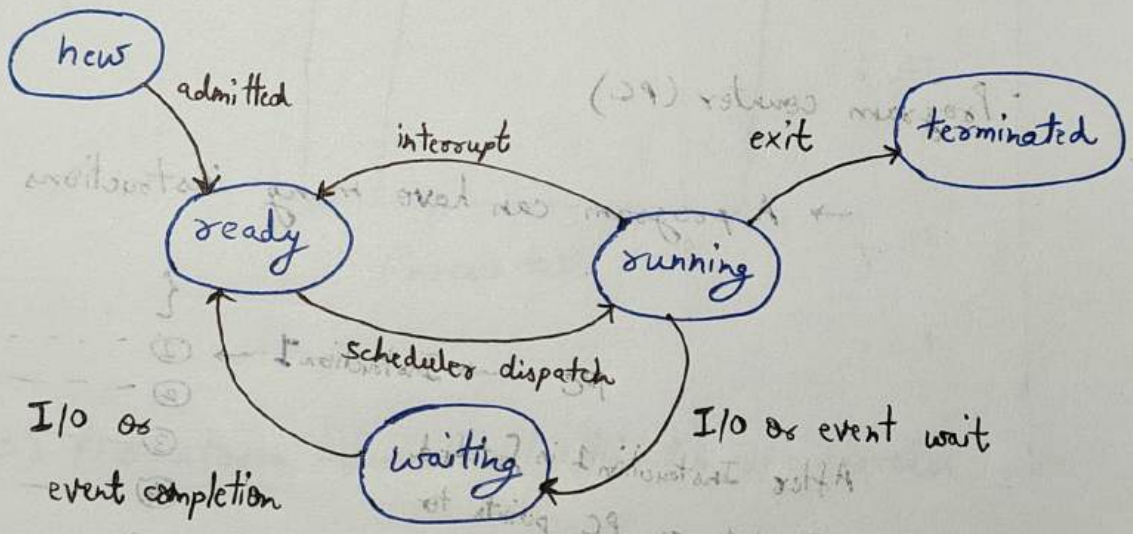


- Registers

→ It is a data structure. When a process is running and it's time slice expires, the current value of process specific registers would be stored in the PCB and the process would be swapped out.

→ When the process is scheduled to be run, the register value is read from the PCB and written to the CPU registers.

Process states



→ As process executes it changes state. Each process may be in one of the following states:

- New: OS is about to pick the program & convert it into process.

"OR"

The process is being created.

- Run: Instructions are being executed (CPU is allocated)
- Waiting: Waiting for IO.
- Ready: The process is in memory, waiting to be assigned to a processor.
- Terminated: The process has finished execution. PCB entry removed from process table.

Process Queues:

- Job Queue
  - → Processes in new state are in job queue.
  - → Present in secondary memory or disk
  - → Job scheduler (Long Term Scheduler (LTS)) picks process from the pool and loads them into memory for execution
    OR
    job pool
  - → The job queue is the list of all processes that have entered the system but are not yet loaded into RAM.
  - → These jobs are waiting for the long term scheduler to select them and admit them into memory so they can be executed later by the CPU.

- Ready queue
  - → Processes in ready state are in ready queue.
  - → Processes are present in RAM.
  - → CPU scheduler (Short term scheduler) picks process from ready queue and dispatch it to CPU.

→ The ready queue holds all the processes that are in main memory (RAM) and are ready to use the CPU but are waiting for their turn.

→ These processes have no issues, they're not waiting for input / output and they're not new. They're simply waiting for the CPU to become waiteble.

- Waiting Queue

→ Processes in this queue are in wait state.

→ The waiting queue holds all the processes that cannot continue right now because they are waiting for some event to happen.

like Input/Output
A file to be read or written
A signal from another process.

Eg:- When you open a file

- When we open a file OS creates a process for that program
- That process goes to the job queue first
- Then the job scheduler may move it to RAM (if there's space) then from ready queue the process is allocated CPU by short term scheduler

OR can say above example in other words

→ The OS creates a process for the program that will open the file.

→ That process is put into the job queue first (i.e it's waiting to be loaded into RAM).

→ The long term scheduler checks if there's space in memory. If yes, it moves the process into RAM.

→ Then the process enters the ready queue.

→ Finally it is assigned to the CPU by the short term scheduler for execution

{In case of a folder}

→ The OS sends a file system request to the disk

→ The disk returns a list of file names, sizes, types (metadata).

→ The OS displays that list in the File explorer window.

Definition

Long Term Scheduler

→ Selects which jobs from the job queue are admitted into RAM.

→ Controls the degree of multiprogramming (how many processes are in memory).

Short term scheduler

→ Picks one process from ready queue and assigns it to the CPU.

→ Happens very frequently - whenever a process finishes, waits or is preempted.

Medium or Middle term scheduler

→ Temporarily removes processes from RAM to free up memory.

→ Later it brings them back into RAM when resources are avilable.

→ If RAM is full and a new process needs to be loaded, the medium term scheduler may pause and swap out a low priority process to disk.

- Working

① → OS decides that a process must be suspended (maybe it's low priority or inactive)

② → The process's memory contents are copied to swap space on hard disk.

Storage space on hard disk used like a backup for RAM.

③ → It's state and information are updated to suspended (in PCB)

④ → Process is removed from RAM to free space

⇒ This is called swapping out.

① → OS decides that the process can now resume (RAM is free)

② → It loads the process's memory contents from disk back into RAM.

③ → Process state is changed from suspended to ready (or waiting if it was waiting)

④ → Process is placed into the ready queue to wait for CPU time.

⇒ This is called swapping in.

{ swap in swap out }

{ If it swaps processes on the basis of priority
⇓
roll in roll out }

Degree of multi-programming

→ The number of processes in the memory

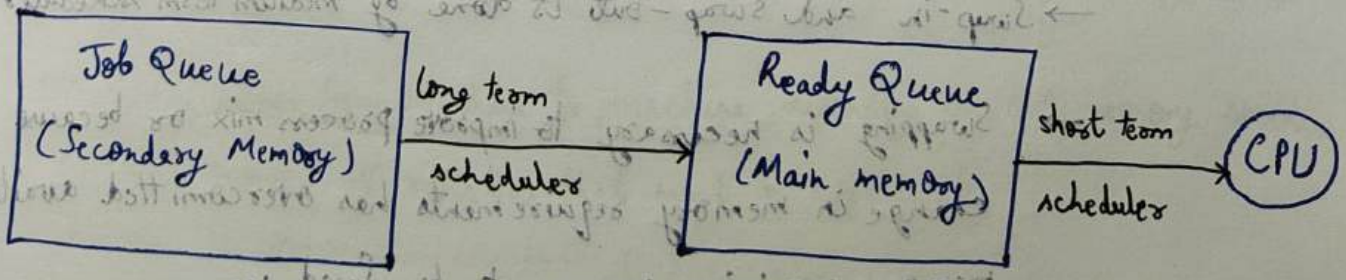
→ long term scheduler controls degree of multi-programming

Dispatcher

→ The dispatcher is the small part of OS that is responsible for giving control of the CPU to the process selected by the short-term scheduler.

Dispatch

→ Dispatch is the action or operation performed by the dispatcher (the actual process of switching the CPU to the selected process.)



→ We always want a mix of process to go for execution.

eg :- P_1, P_2 → CPU intensive

P_3, P_4 → I/O intensive

P_5 → CPU intensive or Memory intensive

So if we only give CPU intensive process the CPU to execute then it will cause starvation.

So we want a mix of processes

{we want mix jobs}

→ So this work is done by job scheduler.

{Medium term scheduler more theory}

⇓

Swapping

→ Time sharing may have medium term scheduler (MTS).

→ Remove processes from memory to reduce degree of multi-programming

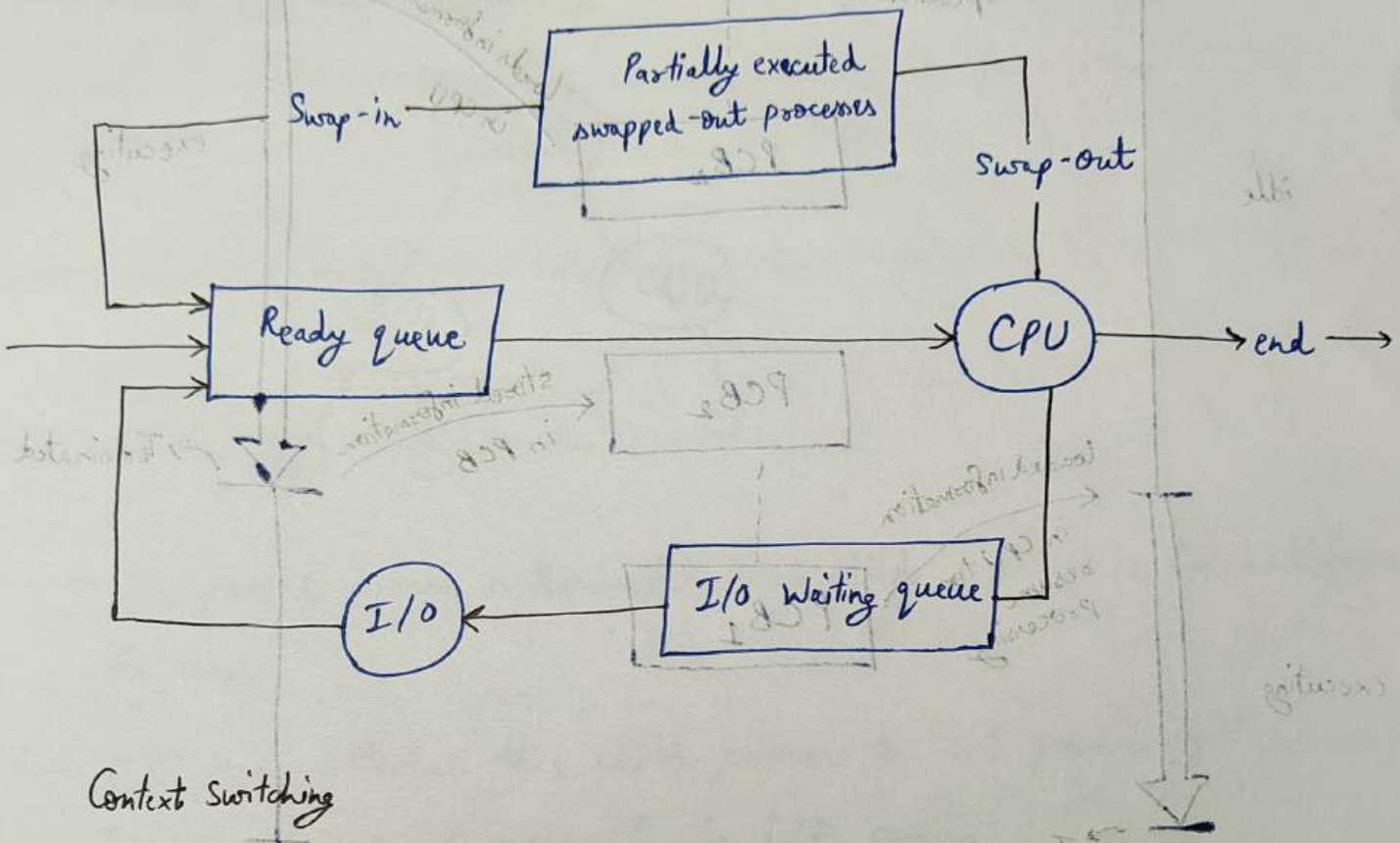
→ These removed processes can be reintroduced into memory and it's execution can be continued where it left off. This is called swapping.

→ Swap-in and swap-out is done by medium term scheduler (MTS).

→ Swapping is necessary to improve process mix or because a change in memory requirements has overcommitted available memory requiring memory to be freed up.

→ Swapping is a mechanism in which a process can be swapped temporarily out of main memory to secondary storage

and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.



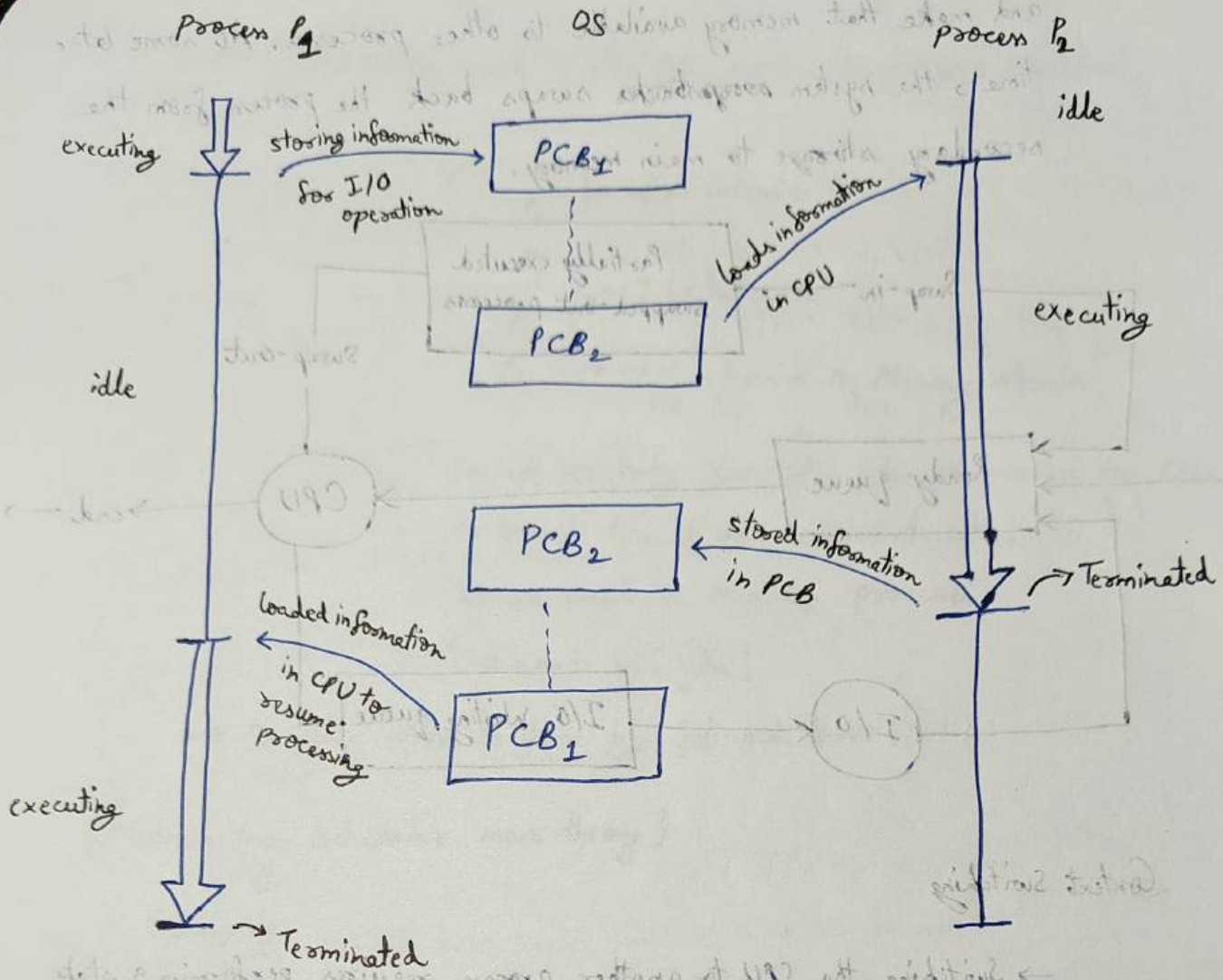
Context Switching

→ Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process.

→ When this occurs the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.

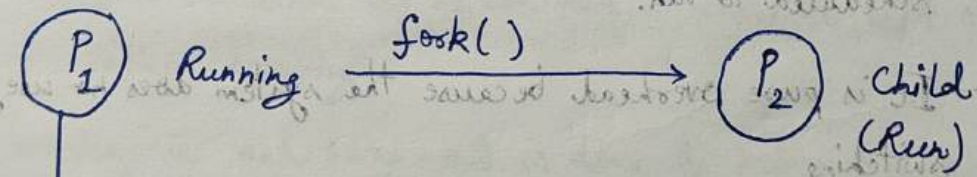
→ It is pure overhead because the system does no useful work while switching

→ Speed varies from machine to machine depending on the memory speed, the number of registers that must be copied etc.



Orphan process

{In Linux ↳ as example init process is the first process with PID 1}

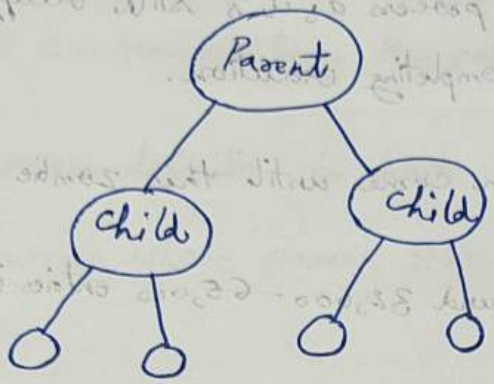


→ In above diagram P_1 is parent process and P_2 is child process. It is like a room that parent process must wait for child process to terminate. (to note the child process exit state)

↳ like if it was executed successfully or there was some error.

→ But in above diagram the parent process is terminated before child process due to exception.

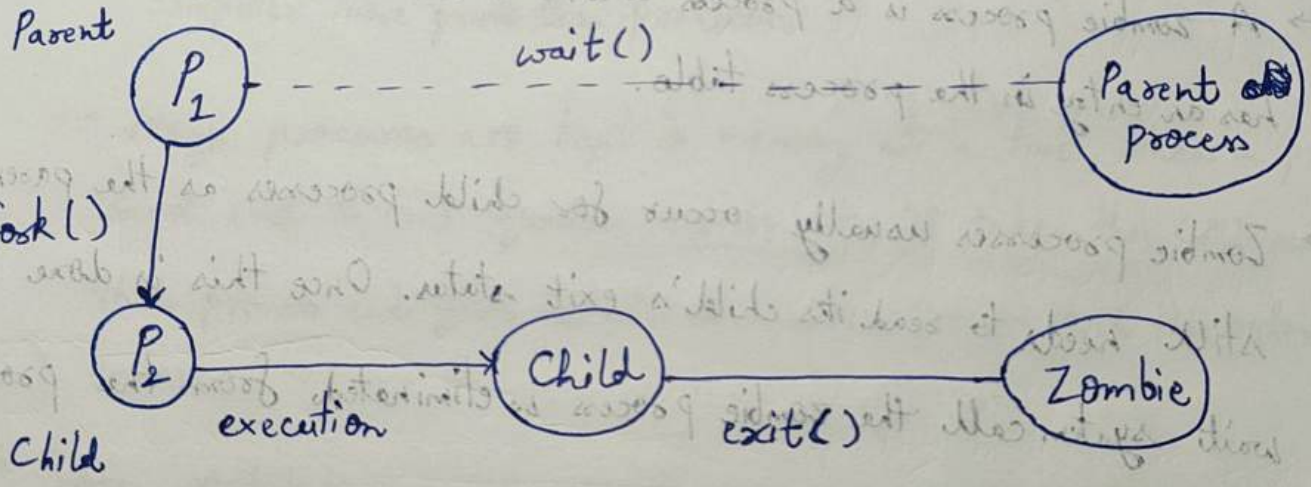
→ Hence the child process P_2 is orphan process.



→ So parent process is terminated hence child process is detached from the tree.

→ OS then attaches this child process to init process (1st process). So now parent of process P_2 is init. process.

Zombie process (or Defunct process)



→ In the above diagram we have put P_1 process on wait() for 5 mins but child process P_2 is executing and finished in just 2 mins.

→ So after 2 mins the child process P_2 has released the resources but is still occupying space in process table as it has to return to parent process before terminating.
↳ or to return exit status to parent P_1.

→ So process P_2 is zombie process as it's still occupying space in process table even after completing execution.

→ So no other process can come until this zombie process is gone.

{ Typical OS default has around 32,000 - 65,000 entries in process table }
↳ Confirm once with someone

→ In other case parent process might not have called wait() so there will be no resource leak but if parent process exits before as it doesn't wait so in this case OS might have a bug due to which there will be resource leak.

→ A zombie process is a process whose execution is completed but it still has an entry in the process table.

→ Zombie processes usually occur for child processes as the parent process still needs to read its child's exit status. Once this is done using the wait system call the zombie process is eliminated from the process table. This is known as reaping the zombie process.

→ It is because parent process may call wait() on child process for a longer time duration and child process got terminated much earlier

→ As entry in the process table can only be removed after parent process reads the exit status of child process. Hence the child process remains a zombie till it is removed from the process table.

Orphan process definition in short

→ The process whose parent process has been terminated and it is still running.

→ Orphan processes are adopted by init process.

→ Init is the first process of OS.