①
```
# include <stdio.h>
main()
{
    Printf(" Hello ---\n");
    System (" ls");
    Printf (" Hai ---\n");
}
```

②
```
# include <stdio.h>
main()
{
    Printf (" Hello ---\n");
    execl (" /bin / ls ", "ls", NULL);
    Printf (" Hai ---\n");
}
```

Output

Hello ----

ls command

Hai --- is not printed.

→ after execl, the codes are not execute.

→ Its print a past new image on a Old Image.

→ execl Statement never pretreble in child process.

③
```
# include <stdio.h>
main()
{
    Printf (" Hello ---\n");
    execl (" /bin/ls ", "ls", " -l", " -i", NULL);
    Printf (" Hai ----\n");
}
```

① when bash is waiting for input

② user provide a command

③ Then bash work a one duplicate child process.

④ In child process duplicate process it will be call the exec.

$ ls ; pwd ; cal

① $ ls

② $ /bin/ls

③ $cal

command is execute

④ $ /bin/cal

no such file or directory.

→ If we supply a command to a bash including a path it will search only in that path.

Environmental Variable :-

(Shell Variable) (system Variable)

→ It's the concept for to know if does not provided a path where-where it will be searching for the command to know that variable need this concept.

→ The designer of shell they designed some variable for storing some environment variable.

① Command : env

→ All environment variable in capital letter.

② Environment Variable : Path (PATH)

→ If we does not provide a path it will be searching each path is seprated by colorum(;)

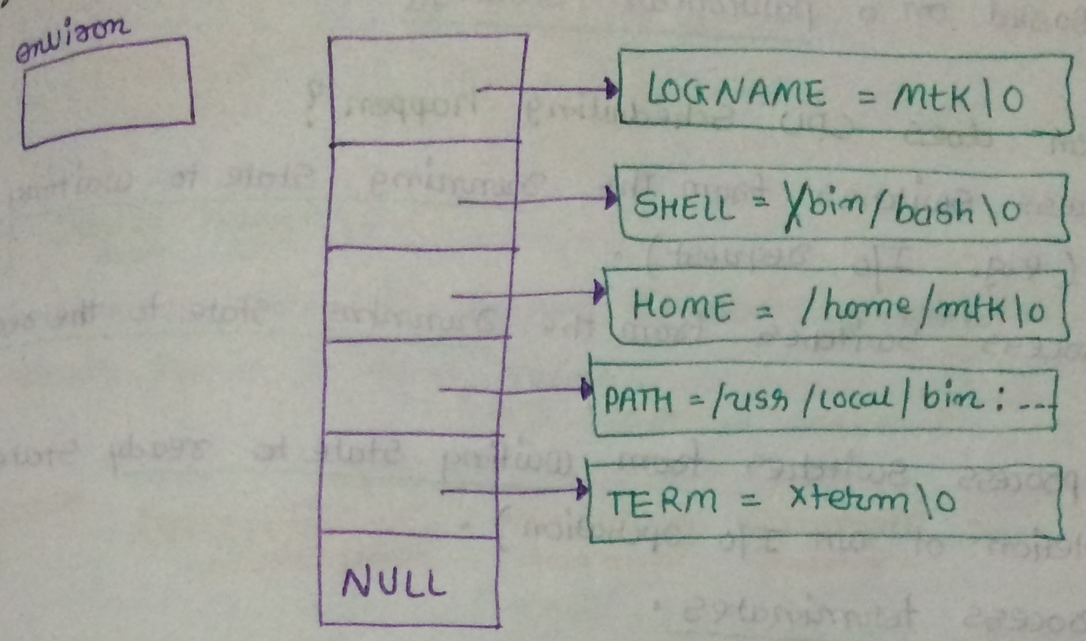
③ Command : echo $ PATH

↳ Shell Scripting word.

Example write a program to print all the environment variable after every 1 second.

```
# include <stdio.h>
main(int argc, char **argv, char **env)
{
    int i;
    for(i=0 ; env[i] ; i++)
```

```
    {   printf(" %s\n", env[i]);
        sleep(1);
    }
}
```



execl → Not provide a path

execlp → It is better than the execl.

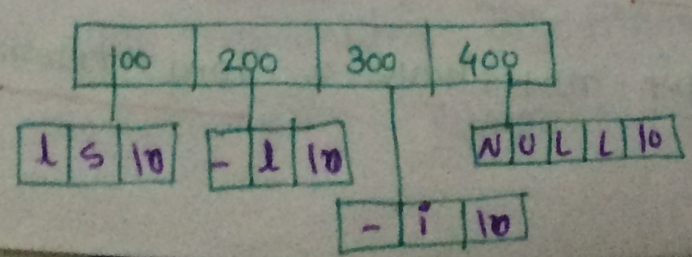
⇒ These two Variable argument list function at compile time only we know How many argument to pass.
(to overcome this using execvp, execlp).

②
```
# include <stdio.h>
main()
{
    char *p[] = {"ls", "-l", "-i", NULL};
    execvp (p[0], p);
}
```

⇒ Size of p is 16 bytes

⇒ p[0] = holding the address of ls.

p[3] = holding it self NULL.



CPU Schedular :-

→ A CPU schedular, running in the dispatcher is responsible for selecting the next running process.

- Based on a particular strategy.

(*) when does CPU Scheduling happen?

→ A process switches from the running state to waiting state (e.g. I/O request).

→ A process switches from the running state to the ready state.

→ A process switches from waiting state to ready state. (completion of an I/O operation).

→ A process terminates.

Scheduling queues :-

① Job queue :- consits of all processes.

→ All Jobs, one submitted are in the Job queue.

→ Some processes can not be executed.

② Ready queue :-

→ All processes that are ready and waiting for execution are in ready queue.

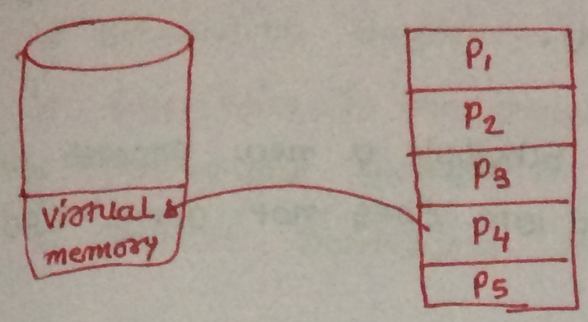
→ usually a long term Schedular selects processes from the Job queue to the ready queue.

1) Short term schedular
2) middle term Schedular
3) long term schedular

→ CPU schedular / Short term Schedular Selects a process from the ready queue from execution.

→ Simple system may not have a long-term Job Schedular.

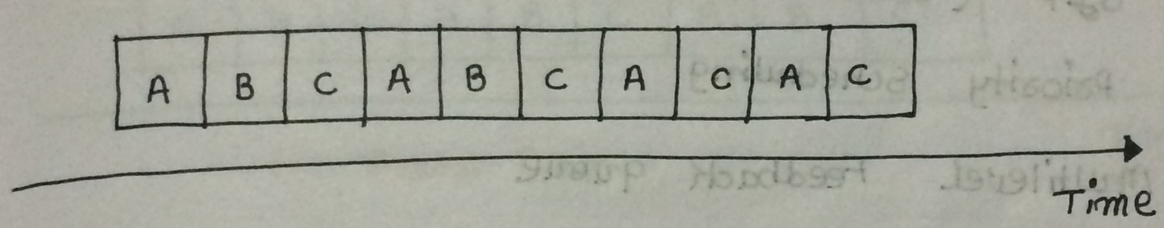
→ If no space in RAM then unload a process from RAM to virtual memory by medium term schedular.



③ Device queue :-

→ when a process is blocked in an I/O operation, it is usually put in a device queue.

Deterministic modeling example :-



	Response time	Wait time	Total time
A	0	5TS + 8CST	9TS + 8CST
B	1TS + 1CST	3TS + 4CST	5TS + 4CST
C	2TS + 2CST	2TS + 1CST	

TS = time slice

CST = Contact switch time

Preemptive Versus nonpreemptive Scheduling :-

→ Many CPU scheduling algorithm have both preemptive and non preemptive versions.

① **Preemptive** :- Schedule a new process even when the current process does not intend to give up the CPU.

② **non-preemptive** :- only schedule a new process when the current one does not want CPU any more.

Scheduling Policies :-

① FIFO (First in first out)

② Round Robin

③ SJF (Shortest Job first)

④ Priority scheduling

⑤ Multilevel Feedback queue

① **FIFO** :- (disadv.)

→ non premptive.

advant.

→ Also known as FCFS (First come first serve)

→ simple

→ Short Job can get stuck, behind long Jobs.

→ Turnaround time is not ideal.

② **Round Robin** :-

→ Periodically releases the CPU from long running Jobs.

→ Based on timer interrupts.

* **Preemptive** :- a process can be forced to leave its running state.

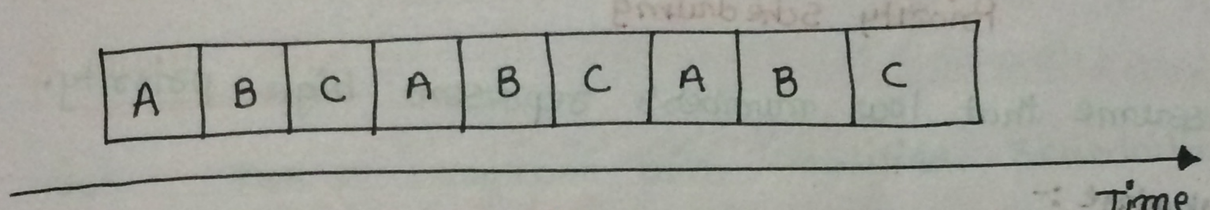
time slice :-

* if time slice is too long :-
  - scheduling degrades to FIFO.
* if time slice is too small :-
  - Throughput suffers.
  - Context switching cost.

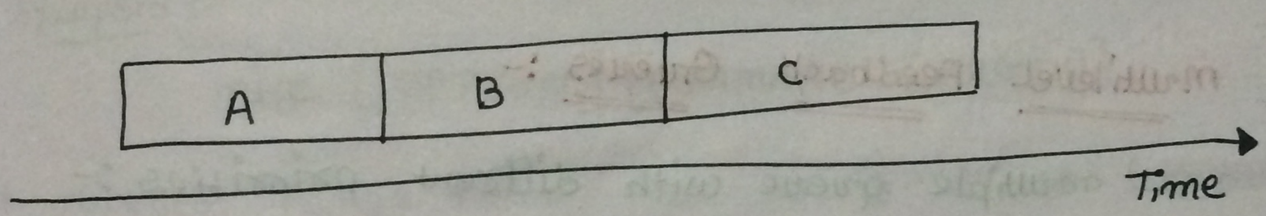
Que :- With zero-cost context FIFO is best or Round Robin ?

Ans Round Robin.

Que :- FIFO Vs Round Robin :-



(a) Round Robin



(b) FIFO

→ Total time is equal in both Round Robin and FIFO.

→ Suppose we have three Jobs of equal length (FIFO is best).

Compare to Response time (Round Robine is best).

③ Shortest Job first :- (SJF)

→ SJF runs whatever Job puts the least demand on the CPU also known as STCF (Short time to Completion first).

drawback :-

1) Starvation time.

Solution :-

1) user cheats, the Job is Killed.

2) Ask the user, who may not be better.

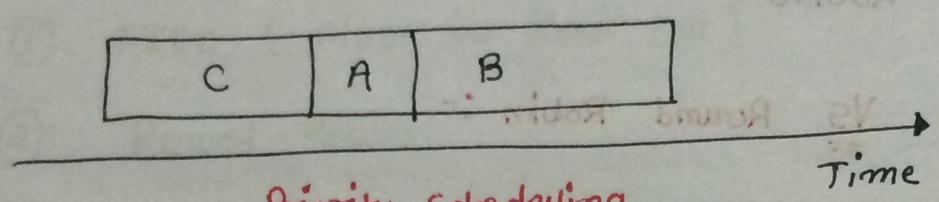
④ Priority Scheduling (multilevel Queues) :-

⇒ The process with the highest priority runs first.

Priority 0 : C

Priority 1 : A

Priority 2 : B



Priority Scheduling

⇒ Assume that low numbers represent high priority.

disadvantage :-

1) low priority is Stucked.

⑤ Multilevel Feedback Queues :-

⇒ use multiple queue with different priorities :-

① Round Robin at each priority level.

② Run highest priority jobs first.

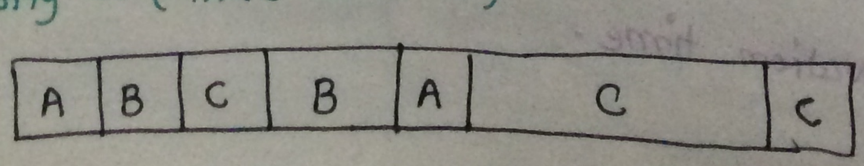
③ once those finish, run next highest priority.

④ Jobs Start in the highest priority queue.

⑤ If time slice expired, drop the job by one level.

⑥ If time slice does not expire push the job up by one level.

Priority 0 (time Slice = 1)

Priority 1 (time Slice = 2)

Priority 2 (time Slice = 4)