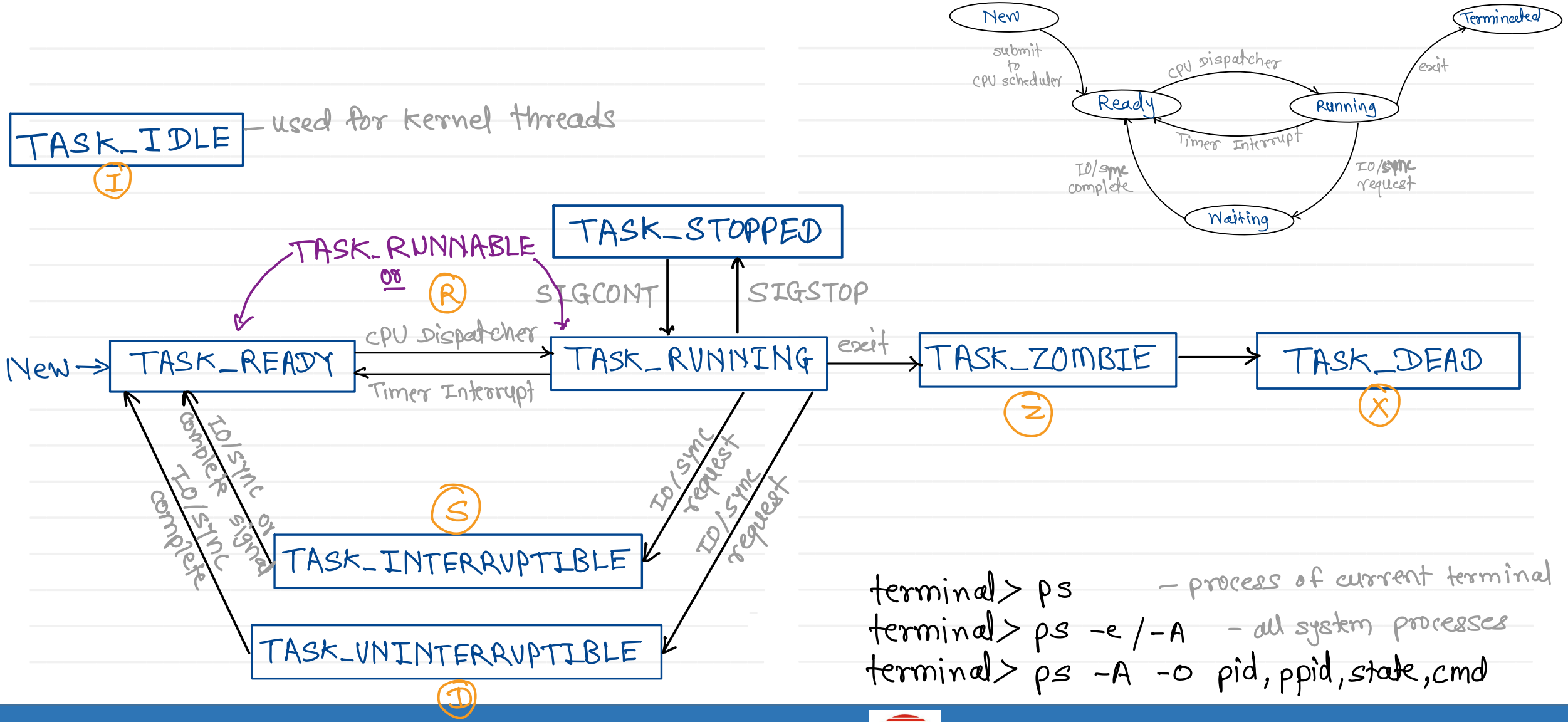
Sunbeam Institute of Information Technology
Pune and Karad

Module - Embedded Operating System

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Linux process life cycle




```
terminal> ps          - process of current terminal
terminal> ps -e /-A   - all system processes
terminal> ps -A -o pid,ppid,state,cmd
```

exit() vs _exit()

exit()
- C library function
- stdio.h
- internally calls _exit() & perform C runtime cleanup

_exit()
- system call
- unistd.h
- release process resources (memory, file....) and writes exit status in its PCB

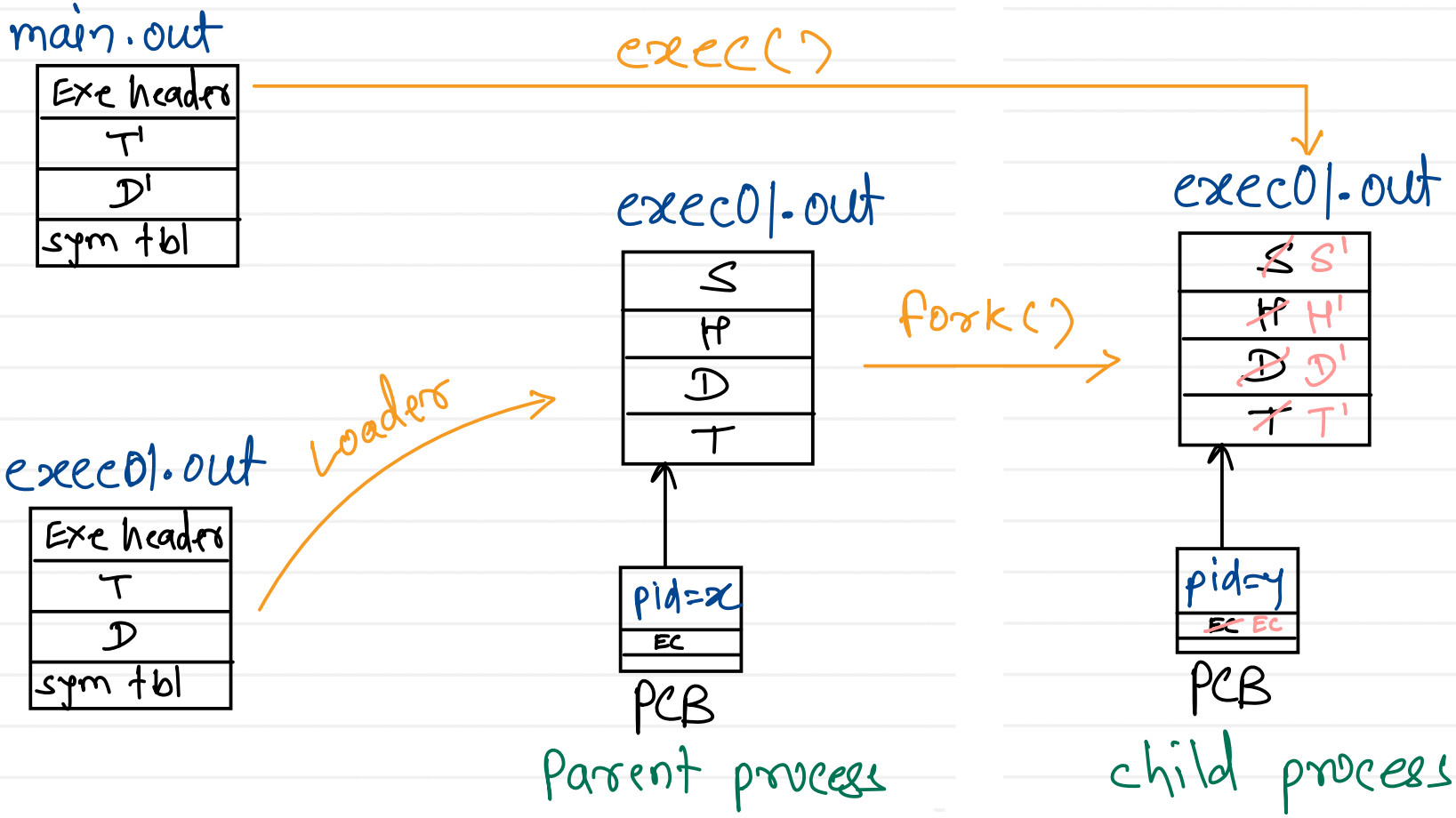
wait() vs waitpid()

wait()
1) block execution of current process until one of its child is terminated.
2) after child is terminated, get its exit status from its PCB & put it into out parameter.
3) release PCB of child.

waitpid()
- similar to wait(), but wait for a given child (pid - arg1) & given flags (default=0)

waitpid(-1, &s, 0) <==> wait(&s)

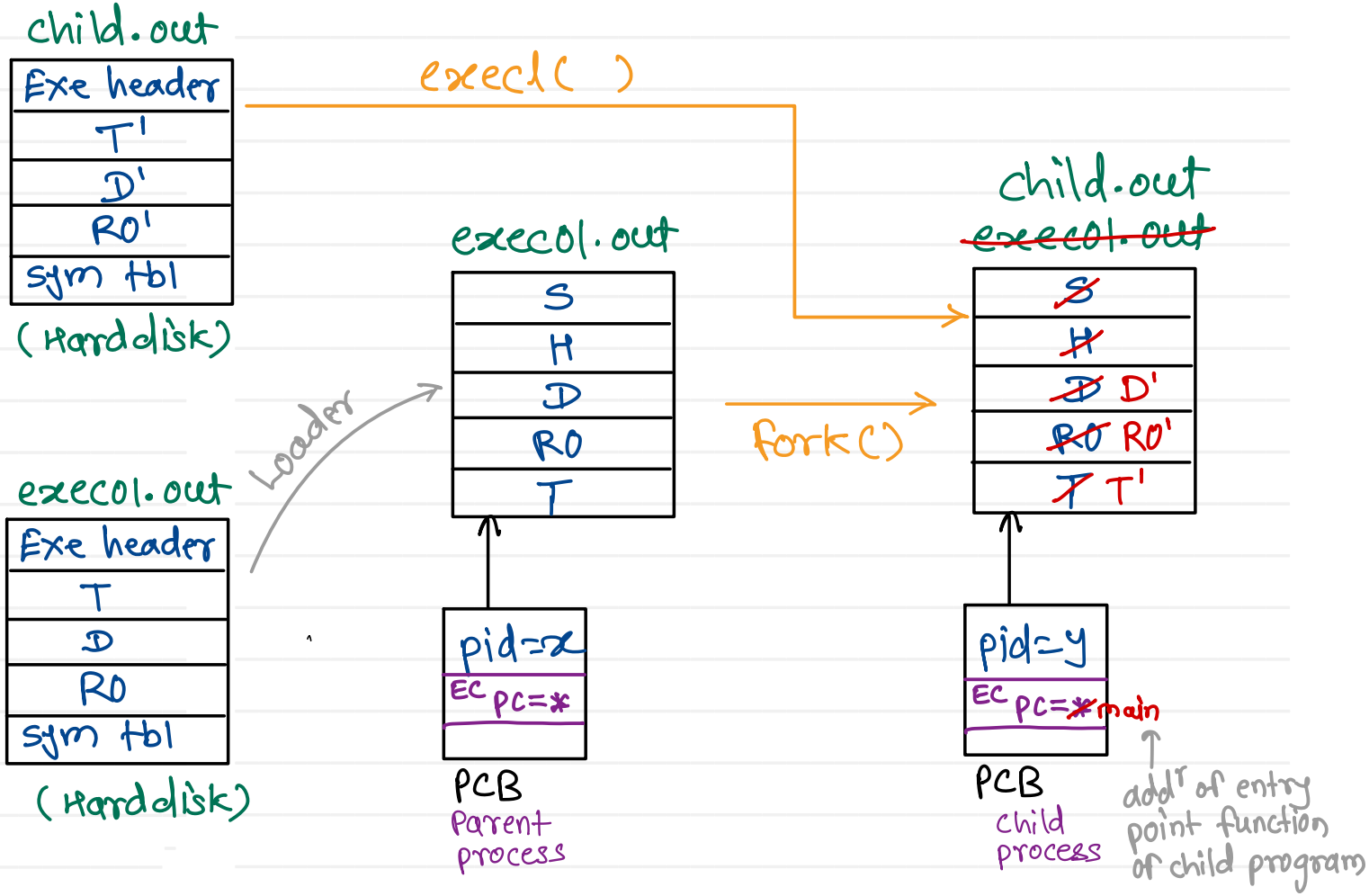
-1 : any child
0 : no flags


main.out
Exe header
T'
D'
sym tbl
exec()
exec01.out
S
H
D
T
pid=x
EC
PCB
Parent process
fork()
exec01.out
S S'
H H'
D D'
T T'
pid=y
EC EC
PCB
child process
exec01.out
Exe header
T
D
sym tbl
loader

exec() syscall

- execc() loads a new program in calling process memory by replacing old program image.

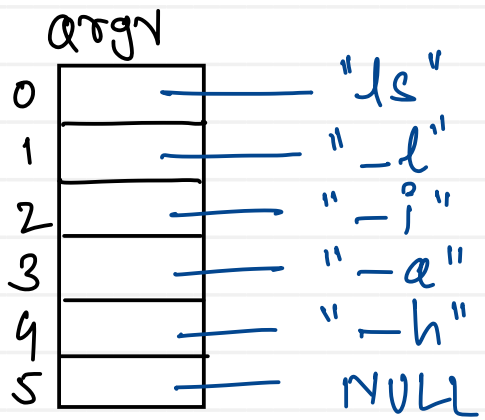
```
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>

int main() {
    int pid,ret,s;
    pid=fork();
    if(pid==0) {
        ret=exec("/path/of/child",args);
        if(ret<0) {
            printf("exec() failed");
            exit(-1);
        }
        else
            waitpid(ret,&s,0);
        return 0;
    }
```



shell $ ls -l -i -a -h

program : ls

cmd line args : ls, -l, -i, -a, -h, NULL



int execve(pathname, argv[], envp[])

pathname : path of program
argv : array of cmd line args
envp : array of env vars.

exec() family functions

exec() lib functions (3)
1) execl()
2) execv()
3) execlp()
4) execvp()
5) execle()
6) execvpe()

exec() sys call (2)
7) execve()

l - variable arg list
v - variable arg vector/array
p - find exe in PATH var automatically.
e - env parameters

1. `ret = execl("exePath", "exeName", "arg1", ..... NULL);`

2.
```
char *args[] = {"exeName", "arg1", .... NULL};
ret = execv("exePath", args);
```

3. `ret = execlp("exeName", "exeName", "arg1", ... NULL);`

4.
```
char *args[] = {"exeName", "arg1", ... NULL};
ret = execvp("exeName", args);
```

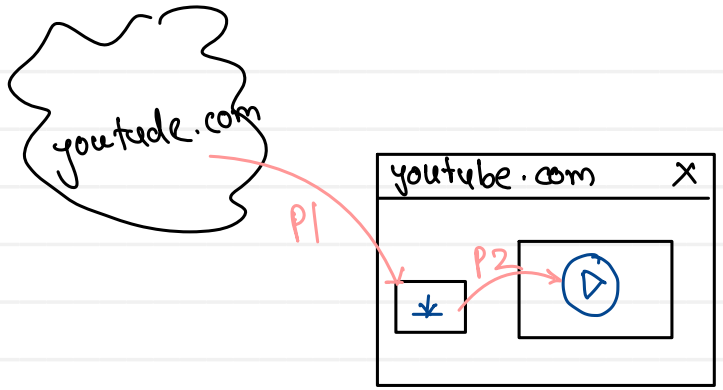
5.
```
char *envp[] = {"K1=V1", "K2=V2", .... , NULL};
ret = execle("exePath", "exeName", "arg1", .... NULL, envp);
```

6. `ret = execve("exePath", args, envp);`

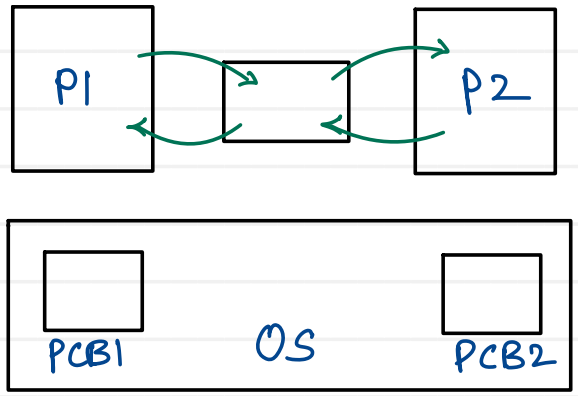
7. `ret = execvpe("exeName", args, envp);`

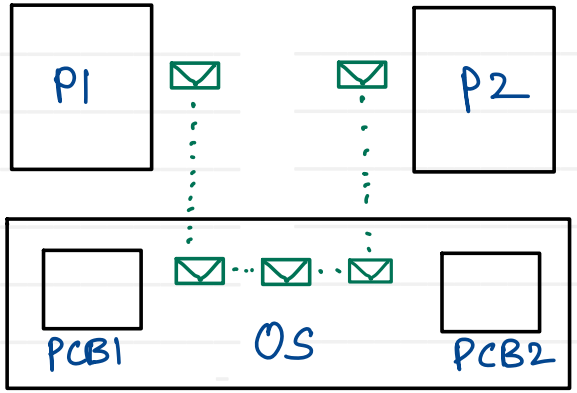
IPC

- single system can have multiple processes running together (multitasking)
- those processes can communicate / share / exchange information through IPC mechanisms.
- OS provides few ways to do comm^{n}.



i) Shared Memory model ii) Message Passing Model





Linux IPC

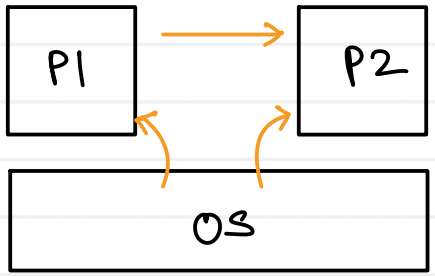
i) 1. Shared Memory

ii)
2. Signal
3. Message queue
4. pipe
5. socket

Signal

- set of predefined signals which processes need to follow.
- approximate 64 signals are available

terminal > kill -l → list of all signal



kill / pkill - used to send signals to the processes

terminal> kill -signo PID
terminal> kill -signame PID
> process ID

terminal> pkill -signo cmd
terminal> pkill -signame cmd
> process name

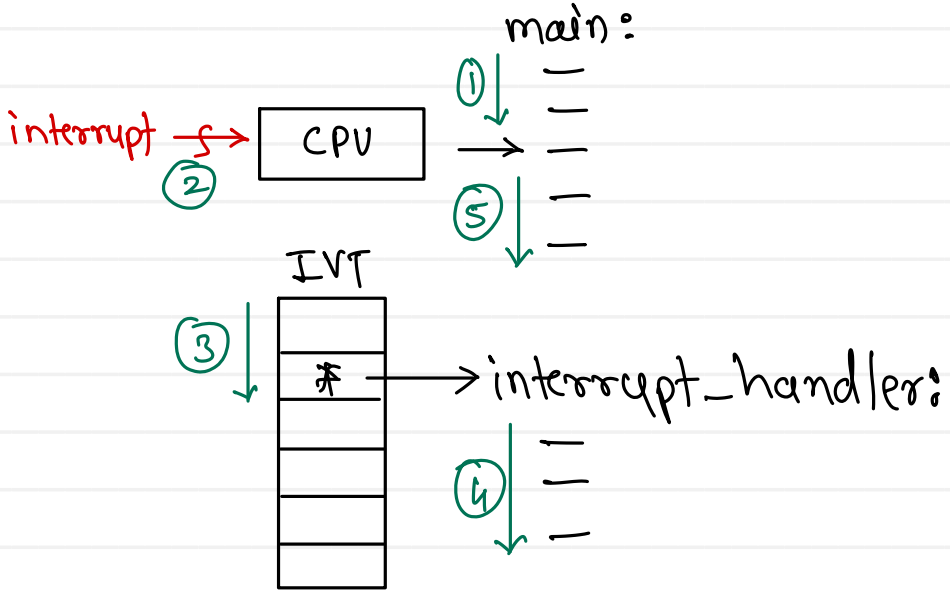
- both commands internally calls kill() system call

kill(pid, sig)

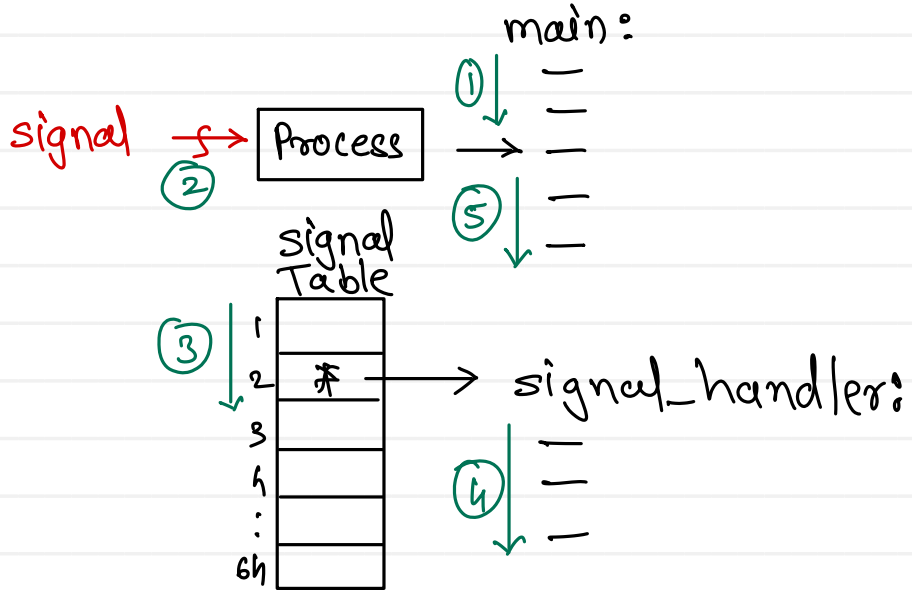
1) SIGHUP - send to processes of terminal when we close it
2) SIGINT - used to interrupt process
   ctrl + C : terminate
9) SIGKILL - used to terminate forcefully.
11) SIGSEGV - on invalid memory access
    core dump
15) SIGTERM - used to terminate process
    (process gets chance to release resources)
17) SIGCHLD - when child terminates, it is given to parent (ignore)
19) SIGSTOP - pause/suspend the process
    ctrl + s : stopped
18) SIGCONT - resume the process
    ctrl + q : continued.

Signal

- signals are software counter part of hardware interrupts



step 1: implement interrupt handler
step 2: register your interrupt handler



step 1: implement signal handler

```
void my_sigHandler(int);
```

step 2: register your signal handler

```
signal( sigNum , my_sigHandler)
```

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com