Unit 2. The process.

- A $\boxed{\text{process}}$ is an instance of running pgm / executing pgm.

- Whenever a pgm get executed by a user, it gives rise to its copy & that is a process.
- These processes use memory. At some time, these memory gets exhausted, then the kernel moves the code & data to ~~the~~ different space called swap area → disk.

- Whenever these process get their time to execute, then a copy is recalled from swap area & processed.

→ Some important attributes are related to process

↳ PID → allotted by kernel at its creation

↳ user-id → owner of the process during its creation.

↳ Priority → value alloted to the process given by the kernel to decide when will it all run.

↳ Current directory from where the process is running.

Birth & Death ⟹ Parent & Child process.

↳ To create any process → one needs to boot the system that sets up the very first process with PID 0.

Eg:- To do a ppt.

GUI.
- → Boot the system (executing the code of OS / kernel → PID 0).
  ↓ given rise (PPT)
- → Open the MS ~~office~~ (executing the pgm internally → child of PID 0).
  ↓ given rise
- → In that you will select ~~PPT~~ New — another option in execution (child of PPT appl. process.)

This process remains alive / active till PPT is open. Once you click close 'button', the process dies.

UNIX: → multiuser / multitasking

↳ process is born when pgm is in execution
↳ remains alive till pgm is active, after execution process dies.

Eg: cat file1;
↳ cmd used in UNIX OS.
↳ ~~given rise by~~ sh ~~cmd of the system bcz it was started / booted.~~

sh is parent & cat is child.

→ We cannot have process without parent processes. (orphaned processes).

→ One process can have only one parent.
One process can create multiple child processes.

★ How a Process is Created?

↳ a process is a pgm in execution.
- ↳ code / statements] seq of
- ↳ data in vars/arrays that has to be used in the code.

↳

→ kernel ~~keeps~~ this a space called as user address space ← cant be interfered by any user.

- TXT SEGMENT
  - ↓ executable code/segment
- DATA SEGMENT
  - ↓ vars/arrays with data
- USER SEGMENT.
  - ↓ UIDs, GIDs & current dir. information.

→ Any process that gets created has 3 phases using 3 system calls.

- fork()
- exec()
- wait()

PHASES

1] FORK :: Any process in UNIX is first created by fork() system call.

→ This call creates a copy of the process that calls it & invokes it.

kes.

Eg:: 'cat' cmd on prompt
'sh' has called it. 'sh' creates copy of itself. & cat gets new PID.
cat ↓ ~~forked~~ called process.
~~forking~~ calling process.

2] EXEC :: In this phase, the parent process overwrites the image/copy with the pgm that has to be executed in real.

Eg:: 'sh' was calling process (parent process)
'cat' was called (child process).
Copy was made.

Now, in exec() ⟶ 'sh' image is replaced by by the pgm or cmd 'cat' entirely & executed.

WAIT :- In this the parent execu [illegible] () cmd to wait for the child process to complete its exe^n. Once it completes, it sends terminate signal to the parent.

* Child process inherits all parameters of user address space & ~~its~~ is able to make changes. But once the child dies, those modified parameters are unavailable to the parent.

---

* The Login shell :- The first user process

  ↳ This is the very first user process created in UNIX. When you boot the system & log in, ~~a parent process~~ kernel sets up a shell pgm → user loggin

  - → sh (Bourne shell)
  - → ksh (Korn shell)
  - → csh (C shell)
  - → bash (Bourne again sh)

→ prompt: Any cmd is i/p to shell pgm.

shell pgm is born when you login → It will die when you log out.

- login shell also gets PID. It is stored in special var. $$.

```
$echo $$
659
```

$\rightarrow$ This won't change till you log out.

$\rightarrow$ login shell PID.

Every time, new PIDs are assigned.

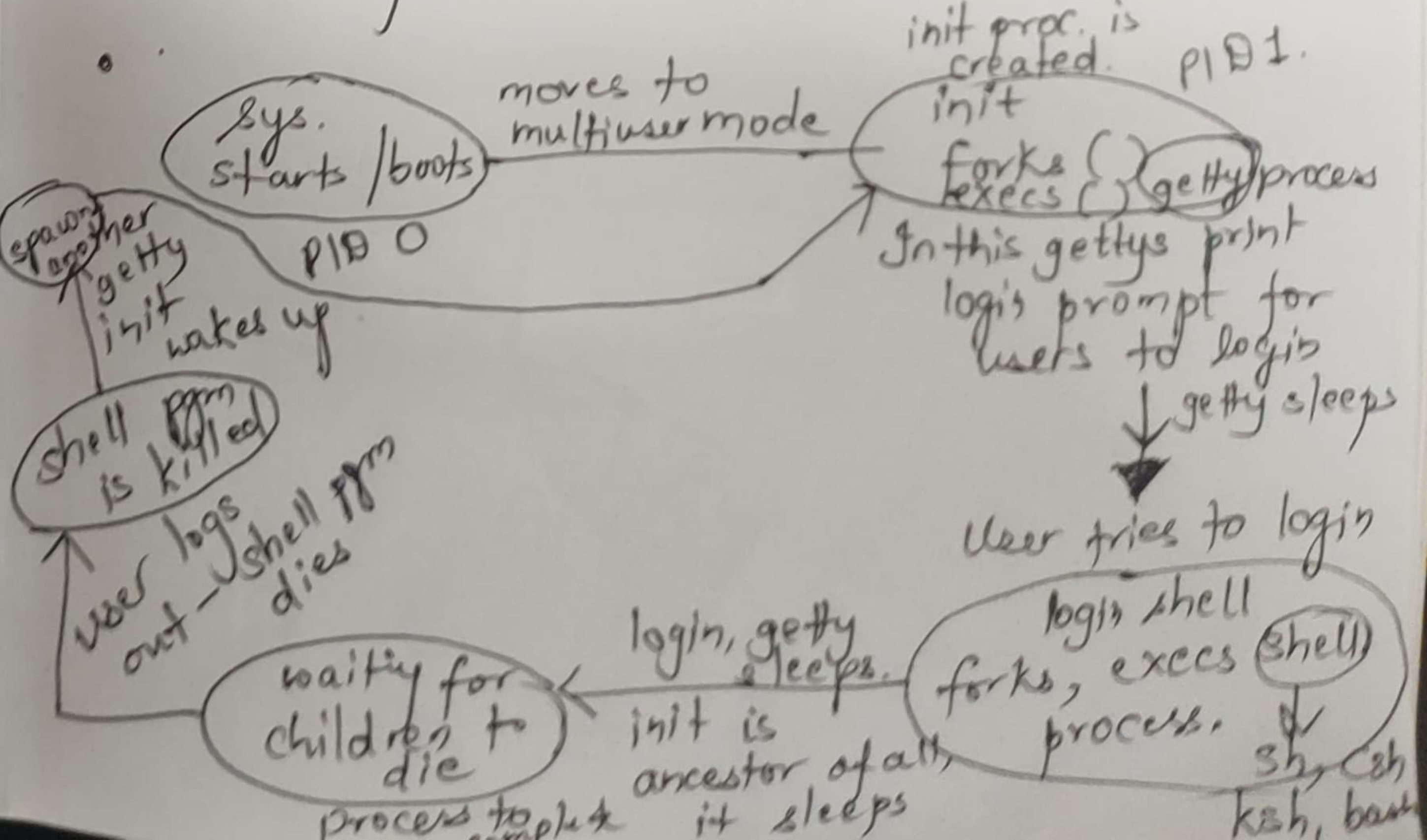
↓ under this many processes are started whose parent is login shell

* You can have many processes under login shell

The init Process. → parent of all processes.

- init is a system process. — parent of login shells
- And this is 2nd process of the system with PID 1.
- The role of init is

•



Sys. starts /boots
PID 0
moves to multiuser mode
init proc. is created. PID 1.
init forks { } execs { } getty process
In this gettys print login prompt for users to login
getty sleeps
User tries to login
login shell forks, execs shell process.
sh, csh ksh, bash
login, getty sleeps.
init is ancestor of all, it sleeps
waiting for children to die
process to complete
user logs out - shell pgm dies
shell pgm is killed
init wakes up
spawn another getty