Introduction to Concurrency

starts at 9:04 pm

useradd -m -G developers john

-m → /home/john_doe

→ Agenda.

- → setfacl
- → Interview question (task)
- → zombie process
- → Concurrency.

Process
Threads

Concurrency
Parallelism
Synchronization Primitives

Race Conditions
Livelocks

Deadlocks

Starvation

→ setfacl .

Blue group.

user :- blue 1 — blue 1.txt

blue 2 — blue 2.txt

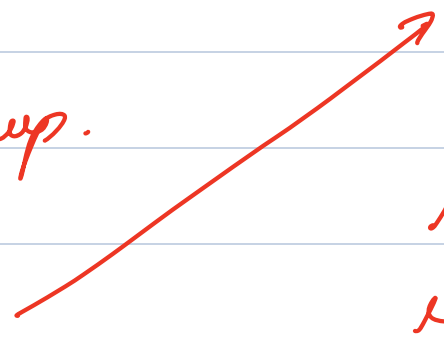
blue 3.txt

Red group.

red 1 — red 1.txt

red 2 — red 2.txt

red 3.txt



syntax

setfacl [options] <ACL> file

-m modify the ACL

-x removing

-R Recursive

ACL

→ setfacl -m u:username:rwx filename.

setfacl -m u:red2:rwx blue2.txt

verify the command
→ getfacl filename

Imagine you're a DevOps engineer at a tech company, and you're tasked with setting up a secure development environment for a new web application project. The project team consists of developers, testers, and a project manager.

You need to ensure that:

* Developers have access to the source code and can deploy to the development server.

* Testers can access the testing environment but not make changes to the source code.

* The project manager can view the progress and access reports but should not modify the environments or the code.

step 1 → create groups.
step 2 → create users and assign to groups.
→ give passwords.

sudo passwd username.

Step 3 → Setup directories and permissions.

/var/www/html/project_source.
" " /project_testing
" " /project_reports.

/var/project_source.

→ Change group ownership of directories we have created.

```
1  clear
2  sudo groupadd developers
3  sudo groupadd testers
4  sudo groupadd project_managers
5  cat /etc/groups
6  cat /etc/group
7  sudo useradd -m -G developers dev1
8  sudo useradd -m -G testers test1
9  sudo useradd -m -G project_managers pm1
10  sudo passwd dev1
```

```
11  sudo passwd test1
12  sudo passwd pm1
13  cat /etc/passwd
14  cat /etc/group
15  cd /var
16  ls -lrt
17  cd
18  clear
19  sudo mkdir -p /var/www/html/project_source
20  sudo mkdir -p /var/www/html/project_testing
21  sudo mkdir -p /var/www/html/project_reports
22  cd /var/www/html
23  ls -lrt
24  sudo chown -R :developers /var/www/html/project_source
25  ls -lrt
26  sudo chown -R :testers /var/www/html/project_testing
27  sudo chown -R :project_managers /var/www/html/project_reports
28  ls -lrt
29  sudo chmod -R 770 /var/www/html/project_source
30  sudo chmod -R 770 /var/www/html/project_testing
31  sudo chmod -R 770 /var/www/html/project_reports
```

→ create new user in project_managers

→ assign pm2 access to project_source.

→ getfacl

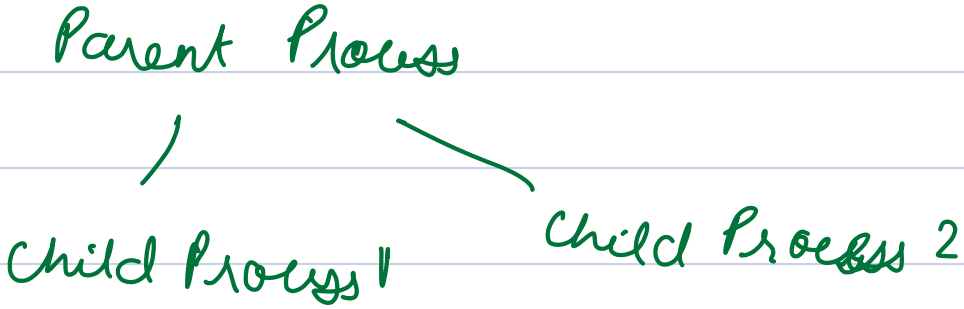
ACL → user:pm2:rwx

mask → rwx

setfacl -m m:rw project_source.

Break → 10:25 pm

Zombie Process.



Program. (Process)

→ Forks Process 2 (Child Process. (completed in 2 seconds)

Sleep 20 → for 20 seconds

Zombie.

Z → state

Stat

R

S

Z

ps -aux | grep Z

kill -9 pid forcefully.

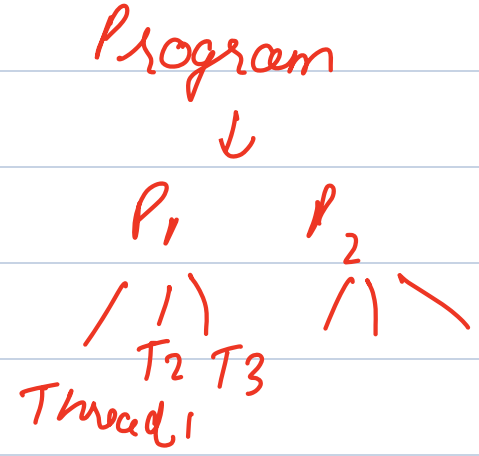
kill pid gracefully.

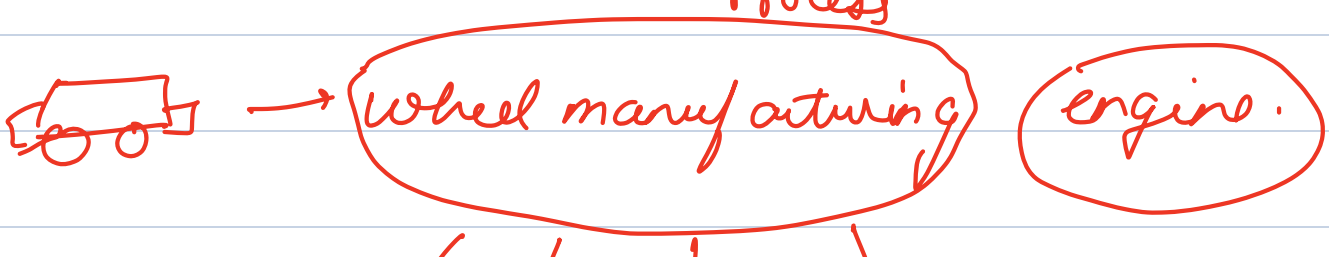
-15 → default

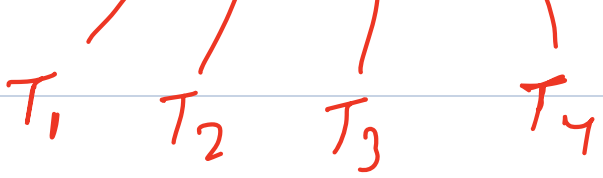
kill -15

kill pid

Threads.



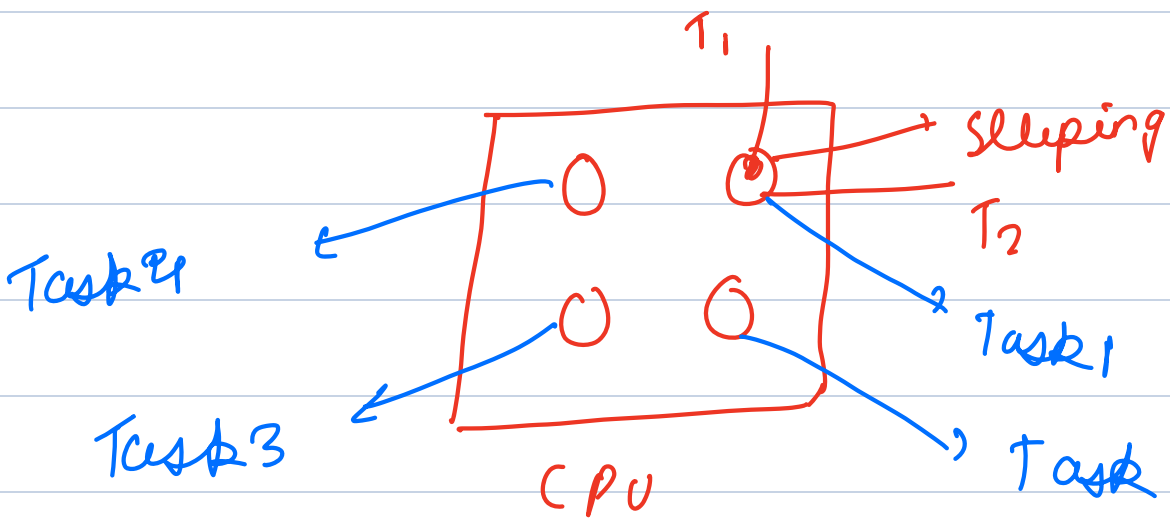




In simple terms if a process is a program that's running, a thread is what's happening inside the program.

It's like a single task that the program is performing, and there can be many tasks happening at once."

Feature	Process	Thread
Definition	An instance of a program running independently.	The smallest unit of processing that can be performed in an OS.
Memory Space	Has its own separate memory space.	Shares memory space with other threads of the same process.
Creation Time	Creating a process is more resource-intensive and time-consuming.	Threads can be created more quickly and efficiently than processes.
Control	Each process operates independently and is isolated from other processes.	Threads are easily controlled by the operating system, allowing for faster context switching.
Resources	Each process has its own resources (files, memory, etc.).	Threads share resources within the same process, which can lead to conflicts if not properly managed.
Use Case	Suitable for applications where tasks are relatively independent.	Ideal for tasks that are closely related and need to share data frequently.



→ Concurrency. (multitasking)

Core 1 Person 1 → Task 1
Core 2 Person 2 → task 2
Core 3 Person 3 → task 3
} Parallelism.

Person → Concurrency.
task 1 task 2 task

Jenkins pipeline.

5 stages.

Stage 1.
Stage 2
Stage 3
— Stage 4 dependent on stage 1
— Stage 5 dependent on stage 2.

Parallely { Stage 1 → Stage 4 [Concurrently]
Stage 2 → Stage 5 [Concurrently]
Stage 3

Feature	Concurrency	Parallelism
Execution	Tasks start, run, and complete in overlapping time periods, not necessarily simultaneously.	Multiple tasks run at exactly the same time.
Processors	Can occur on a single processor by task switching.	Requires multiple processors or cores.
Main Goal	Utilize the computing resources efficiently by task switching.	Reduce the overall time to complete multiple tasks by running them concurrently.
Use Case	Managing multiple connections on a web server.	Processing large datasets or performing complex calculations over multiple CPU cores.
Example	A single-threaded web server handling multiple requests.	A multi-threaded application performing calculations across different threads on a multicore processor.

Context Switching -> The CPU will execute one task for a short period (a few milliseconds),

save its state,

and then switch to another task.

This happens so quickly that to a human user, it feels like the tasks are happening at the same time.