Operating Systems – COC 3071L

SE 5th A – Fall 2025

1. Introduction

A **process** is simply a program in execution.

- When you type a command in Linux (like `ls`), the OS creates a process for it.
  - Every process has:
  - **PID (Process ID)** → unique number for each process.
  - **PPID (Parent Process ID)** → ID of the process that created it.

State → running, sleeping, stopped, zombie, etc.

In this lab, you will:

1. Learn Linux commands to monitor and manage processes.
2. Write C programs to create and observe processes.

---

2. Linux Process Commands

2.1 Viewing Processes

`ps` → Process Status

- Shows processes in the current terminal session.

```
ps
```

Output example:

```
  PID TTY          TIME CMD
 1234 pts/0    00:00:00 bash
 1256 pts/0    00:00:00 ps
```

- **PID** → Process ID
- **TTY** → terminal
- **TIME** → CPU time used
- **CMD** → command name

`ps -ef` → Full list of all processes

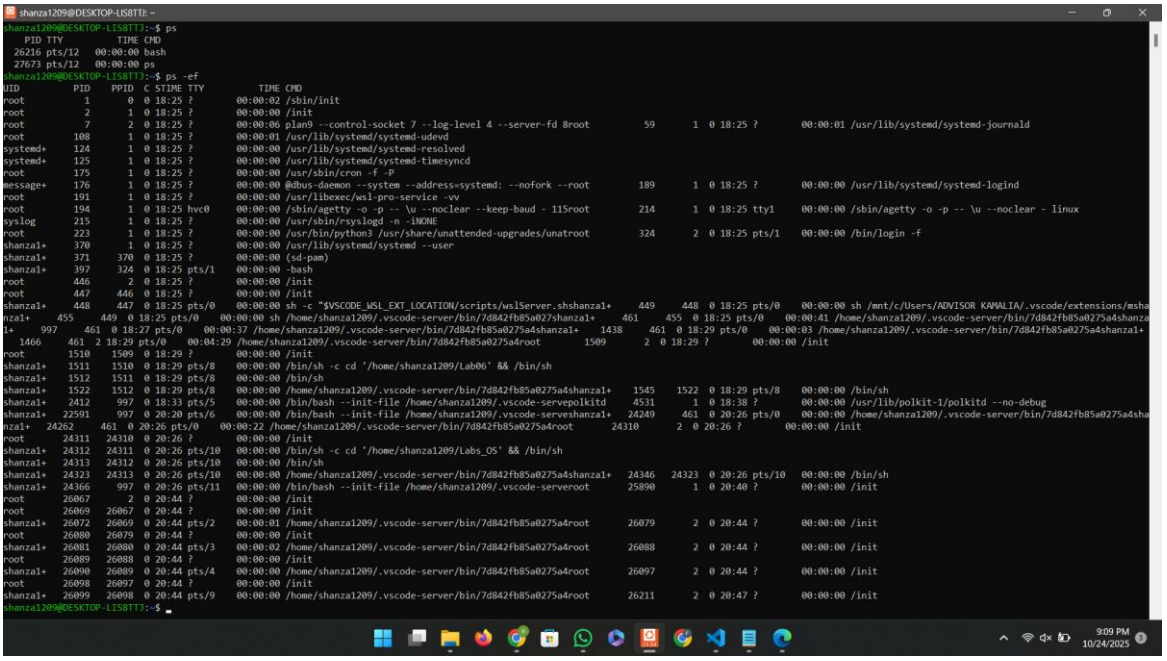
```
ps -ef
```

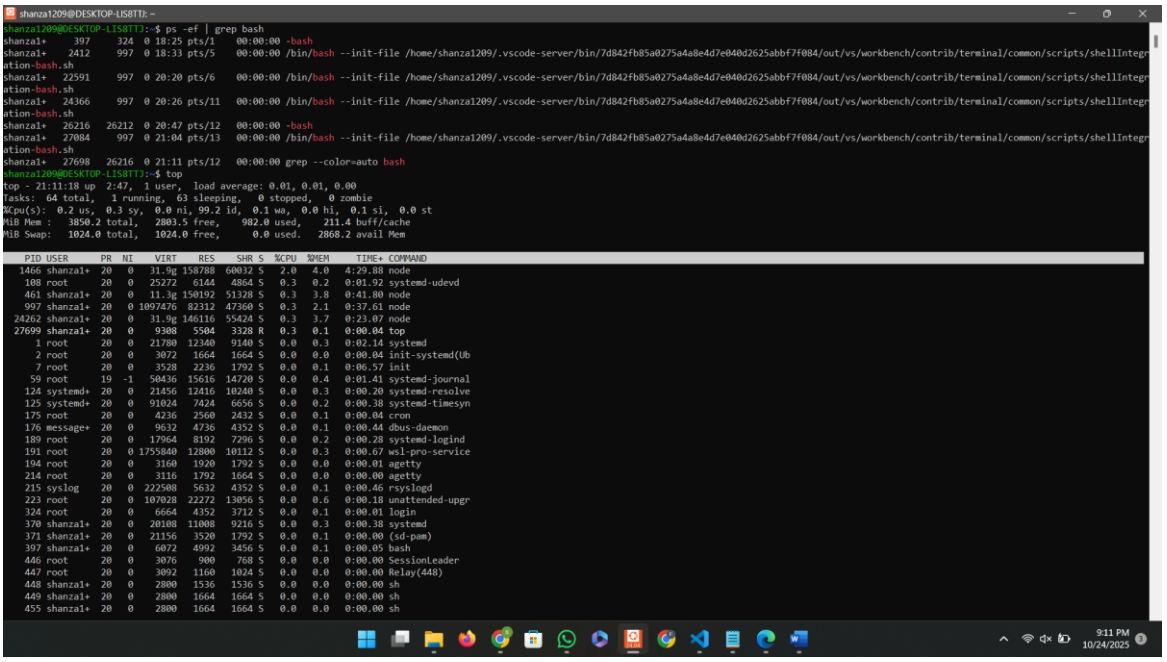
- `-e` → show all processes (not just yours).
- `-f` → full format with UID, PPID, etc.

Try:

```
ps -ef | grep bash
```

This finds all processes related to the `bash` shell.





2.2 Monitoring Processes Interactively

`top` → Dynamic process viewer

```
top
```

- Displays running processes with CPU and memory usage.
- Press `q` to quit.
- Press `k` inside `top` to kill a process (enter PID).
- Press `h` for help.

2.3 Foreground and Background Jobs

- **Foreground**: A process that takes control of the terminal until it finishes.

  ```
  sleep 30
  ```

  → You cannot type new commands until it finishes.

-

  ```
  sleep 30 &
  ```

Background: Add `&` to run without blocking.

  → Terminal is free while the command runs.

-

  ```
  jobs
  ```

Check background jobs:

- **Bring a job to foreground**:

```
fg %1
```

  `%1` means job number 1 (from `jobs` output).
- **Suspend a job**: Press **Ctrl + Z** while it runs.
- **Resume suspended job in background**:

```
bg %1
```

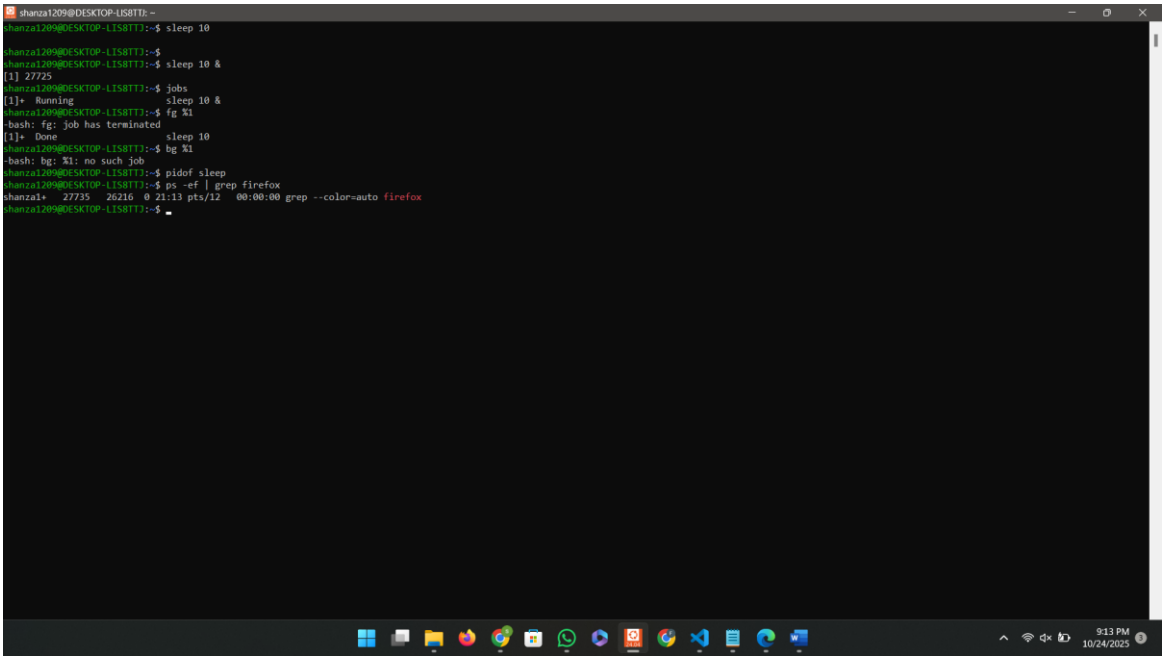
2.4 Process Identification

- **Get PID of a process by name**:

```
pidof sleep
```

  Example output: `3421` (PID of sleep command).
- **Search using `ps` and `grep`**:

```
ps -ef | grep firefox
```



2.5 Killing Processes

- **Kill by PID**:

```
kill -9 3421
```

  - `-9` → force kill (SIGKILL).

- **Kill all processes by name**:

```
killall sleep
```

Practice Task:

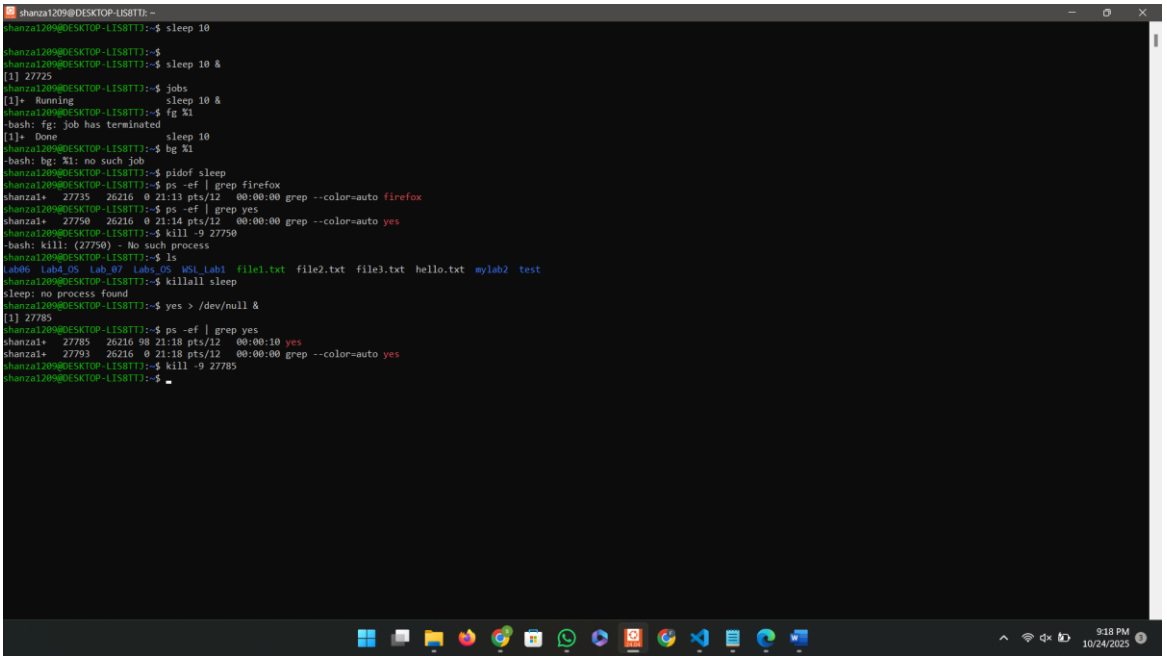
1. Run an infinite process:

```
yes > /dev/null &
```

   (`yes` prints “y” forever; redirected to `/dev/null` to hide output).

2. Find it with:
3. Kill it with:

```
kill -9 <PID>
```



```
ps -ef | grep yes
```

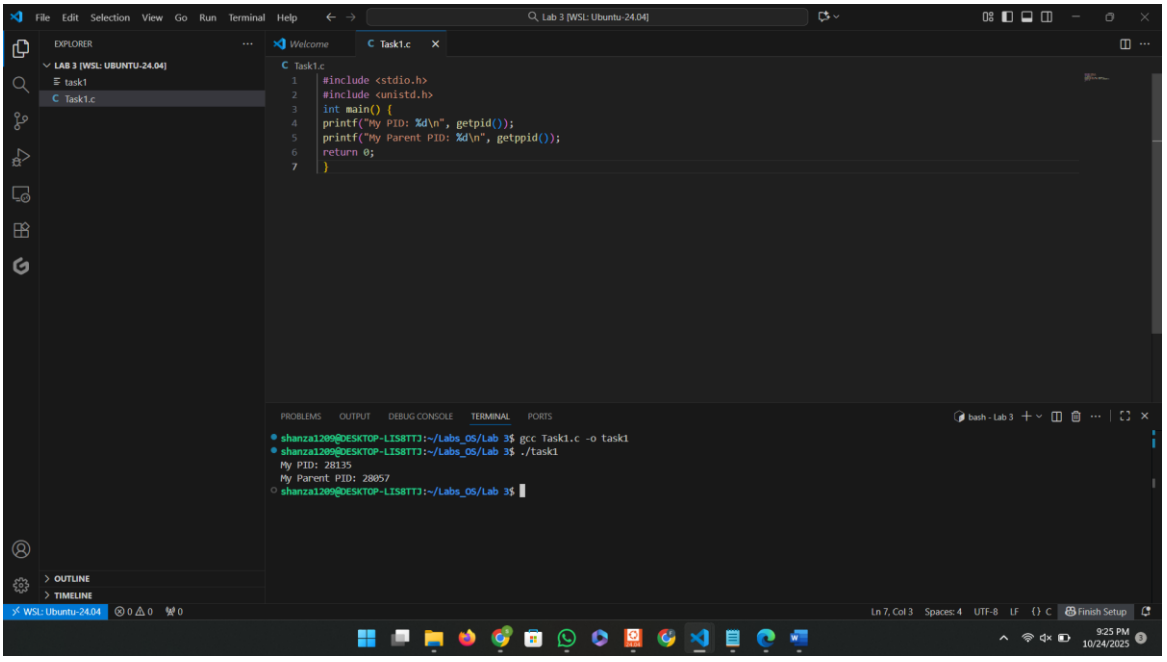
3. C Programs on Processes

Program 1: Print PID and PPID

```c
#include <stdio.h>
#include <unistd.h>

int main() {
    printf("My PID: %d\n", getpid());
    printf("My Parent PID: %d\n", getppid());
    return 0;
}
```

- `#include <unistd.h>` → contains process-related functions like `getpid()` and `getppid()`.
- `getpid()` → returns the unique **process ID** of the current process.
- `getppid()` → returns the **parent's PID**.
- Every process in Linux has a parent (except the very first process, usually `init` or `systemd`).
  Run and compare with `ps -ef`.



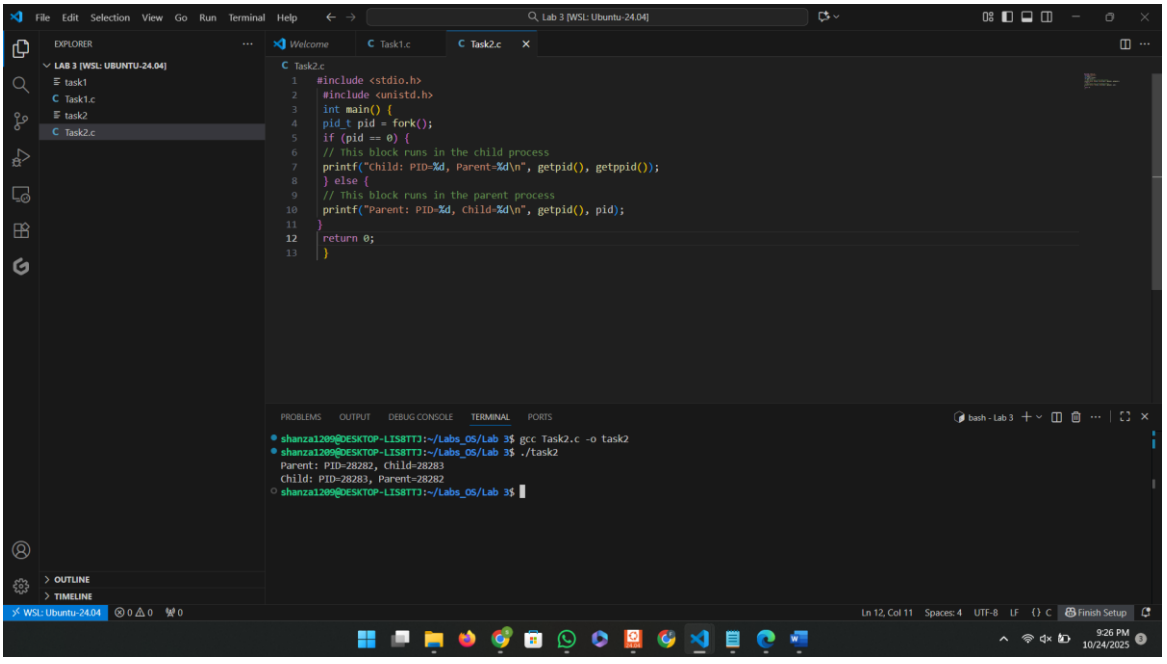
Program 2: Fork – Creating Child Process

```c
#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        // This block runs in the child process
        printf("Child: PID=%d, Parent=%d\n", getpid(), getppid());
    } else {
        // This block runs in the parent process
        printf("Parent: PID=%d, Child=%d\n", getpid(), pid);
    }
    return 0;
}
```

- `fork()` creates a new process by duplicating the current one.
- Return value of `fork()`:
  - `0` → you are inside the **child** process.
  - Positive number (child PID) → you are in the **parent** process.
- After `fork()`, both parent and child run **the same code**, but in different branches of the `if`.



Program 3: Execl – Replacing a Process

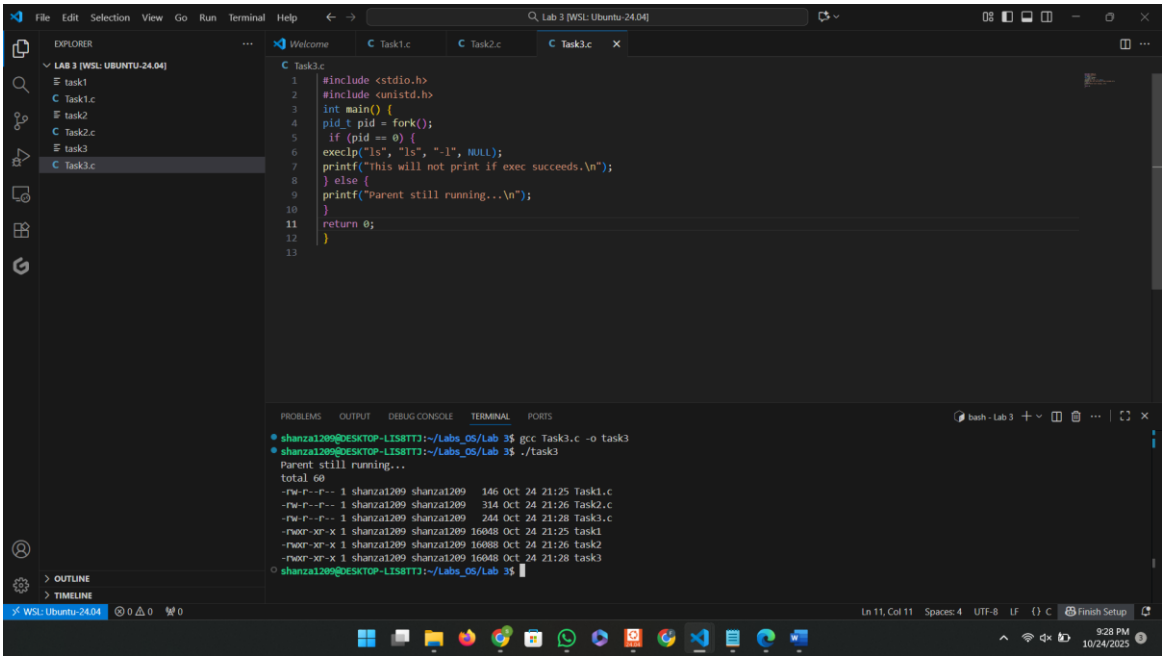
```c
#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        execlp("ls", "ls", "-l", NULL);
        printf("This will not print if exec succeeds.\n");
    } else {
        printf("Parent still running...\n");
    }
    return 0;
}
```

- `fork()` → creates child.
- In the child:
  - `execlp("ls", "ls", "-l", NULL);`
    - Replaces the **current process image** with the `ls` program.
    - First `"ls"` = name of the program, second `"ls"` = argument 0 (how program sees itself).
    - `"-l"` = argument for `ls`.
    - `NULL` marks end of arguments.
-
- Parent is unaffected and continues normally.

After exec() , the child **no longer runs our C code** – it becomes ls .



Program 4: Wait – Synchronization

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

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        execlp("ls", "ls", "-l", NULL);
        printf("This will not print if exec succeeds.\n");
    } else {
        waitpid(pid, NULL, 0);  // Wait for the child process to finish
        printf("Parent still running...\n");
    }
    return 0;
}
```

-
  - `fork()` → creates child.
- `sleep(3)` → child "works" for 3 seconds.
- `wait(NULL)` → parent pauses until child exits.
- Without `wait()`, parent may finish early and child could become a **zombie process**.