

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
```

```
fatima1153@DESKTOP-RMTBL8S: ~
Windows Subsystem for Linux is now available in the Microsoft Store!
You can upgrade by running 'wsl.exe --update' or by visiting https://aka.ms/wslstorepage
Installing WSL from the Microsoft Store will give you the latest WSL updates, faster.
For more information please visit https://aka.ms/wslstoreinfo

fatima1153@DESKTOP-RMTBL8S:~$ ps
  PID TTY          TIME CMD
  3177 pts/1    00:00:00 bash
  3196 pts/1    00:00:00 ps
fatima1153@DESKTOP-RMTBL8S:~$
```

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.

```
fatima1153@DESKTOP-RMTBLS:~$ ps -ef
UID        PID     PPID  C STIME TTY          TIME CMD
root         1         0  0  11:43 ?        00:00:01 /init
root       15         1  0  11:43 ?        00:00:00 /init
root       16         1  0  11:43 ?        00:00:00 /init
fatima1+   17        16  0  11:43 pts/0    00:00:00 sh -c "SVSCODE_WSL_EXT_LOCATION/scripts/wslServer.sh" 385651c938df8a906869babe516bffd0ddb9829 stable code-server .vscode-server --host=127.0.0.1 --port=0 --connec
fatima1+   96        18  0  11:44 pts/0    00:00:00 sh /home/fatima1153/.vscode-server/bin/385651c938df8a906869babe516bffd0ddb9829/bin/code-server --host=127.0.0.1 --port=0 --connection-token=1143359964-75963636-3
fatima1+  100        96  1  11:44 pts/0    00:00:17 /home/fatima1153/.vscode-server/bin/385651c938df8a906869babe516bffd0ddb9829/node /home/fatima1153/.vscode-server/bin/385651c938df8a906869babe516bffd0ddb9829/out/
fatima1+   822       180  0  11:49 pts/0    00:00:03 /home/fatima1153/.vscode-server/bin/385651c938df8a906869babe516bffd0ddb9829/node /home/fatima1153/.vscode-server/bin/385651c938df8a906869babe516bffd0ddb9829/out/
root      1474         1  0  11:52 ?        00:00:00 /init
fatima1+  1476      1475  0  11:52 pts/4    00:00:01 /home/fatima1153/.vscode-server/bin/385651c938df8a906869babe516bffd0ddb9829/node -e const net = require('net'); process.stdin.pause(); const client = net.createCo
root     1483         1  0  11:52 ?        00:00:00 /init
fatima1+  1484       180  0  11:52 pts/0    00:00:00 /home/fatima1153/.vscode-server/bin/385651c938df8a906869babe516bffd0ddb9829/node /home/fatima1153/.vscode-server/bin/385651c938df8a906869babe516bffd0ddb9829/out/
root     1485      1483  0  11:52 ?        00:00:00 /init
fatima1+  1491      1485  0  11:52 pts/5    00:00:00 /home/fatima1153/.vscode-server/bin/385651c938df8a906869babe516bffd0ddb9829/node -e const net = require('net'); process.stdin.pause(); const client = net.createCo
fatima1+  1509       180  6  11:52 pts/0    00:00:33 /home/fatima1153/.vscode-server/bin/385651c938df8a906869babe516bffd0ddb9829/node --dns-result-order=ipv4first /home/fatima1153/.vscode-server/bin/385651c938df8a90
root     1593         1  0  11:53 ?        00:00:00 /init
root     1594      1593  0  11:53 ?        00:00:00 /init
fatima1+  1595      1594  0  11:53 pts/7    00:00:00 /bin/sh -c cd '/home/fatima1153/lab3-os' && /bin/sh
fatima1+  1596      1595  0  11:53 pts/7    00:00:00 /bin/sh
fatima1+  1606      1596  0  11:53 pts/7    00:00:00 /home/fatima1153/.vscode-server/bin/385651c938df8a906869babe516bffd0ddb9829/node /home/fatima1153/.vscode-remote-containers/dist/vscode-remote-containers-server-0
fatima1+  1630      1606  0  11:53 pts/7    00:00:00 /bin/sh
fatima1+  1646       822  0  11:53 pts/8    00:00:00 /bin/bash --init-file /home/fatima1153/.vscode-server/bin/385651c938df8a906869babe516bffd0ddb9829/out/vs/workbench/contrib/terminal/common/scripts/shellIntegratio
root     3175         1  0  12:00 ?        00:00:00 /init
fatima1+  3177      3176  0  12:00 pts/1    00:00:00 -bash
fatima1+  3297      3177  0  12:01 pts/1    00:00:00 ps -ef
fatima1153@DESKTOP-RMTBLS:~$
```

Try:

ps -ef | grep bash

This finds all processes related to the shell.

```
fatima1153@DESKTOP-RMTBLS:~$ ps -ef
UID        PID     PPID  C STIME TTY          TIME CMD
root         1         0  0  14:54 ?        00:00:01 /init
root      118         1  0  14:55 ?        00:00:00 /init
root     300         1  0  14:55 ?        00:00:00 /init
root     301       300  0  14:55 ?        00:00:00 /init
fatima1+   302       301  0  14:55 pts/0    00:00:00 sh -c "SVSCODE_WSL_EXT_LOCATION/scripts/wslServer.sh" e3a5acfb517a443235981655413d56653310root 303 302 0 14:55 pts/0 00:00:00 sh /mnt/c/Users/HP/.vscode/extensions/ms-vs
code-remote.remote-wsl-0.104.fatima1+ 308 303 0 14:55 pts/0 00:00:00 sh /home/fatima1153/.vscode-server/bin/e3a5acfb517a443235981655413d56653310root 312 308 0 14:55 pts/0 00:00:09 /home/fatima1
53/.vscode-server/bin/e3a5acfb517a443235981655413d56653310root 323 1 0 14:55 ? 00:00:00 /init
root     324       323  0  14:55 ?        00:00:00 /init
fatima1+  325       324  0  14:55 pts/1    00:00:02 /home/fatima1153/.vscode-server/bin/e3a5acfb517a443235981655413d56653310root 342 1 0 14:55 ? 00:00:00 /init
root     343       342  0  14:55 ?        00:00:00 /init
fatima1+  347       343  0  14:55 pts/3    00:00:01 /home/fatima1153/.vscode-server/bin/e3a5acfb517a443235981655413d56653310root 353 312 0 14:55 pts/0 00:00:00 /home/fatima1153/.vscode-server/bin/e3a5acfb
517a443235981655413d56653310fatima1+ 374 312 4 14:55 pts/0 00:01:22 /home/fatima1153/.vscode-server/bin/e3a5acfb517a443235981655413d56653310root 402 1 0 14:55 ? 00:00:00 /init
fatima1+  403       402  0  14:55 ?        00:00:00 /init
fatima1+  404       403  0  14:55 pts/4    00:00:00 /bin/sh -c cd '/home/fatima1153/lab3-os' && /bin/sh
fatima1+  405       404  0  14:55 pts/4    00:00:00 /bin/sh
fatima1+  415       405  0  14:55 pts/4    00:00:00 /home/fatima1153/.vscode-server/bin/e3a5acfb517a443235981655413d56653310fatima1+ 438 415 0 14:55 pts/4 00:00:00 /bin/sh
fatima1+  576       312  0  14:57 pts/0    00:00:05 /home/fatima1153/.vscode-server/bin/e3a5acfb517a443235981655413d56653310fatima1+ 3628 576 0 15:16 pts/5 00:00:00 /bin/bash --init-file /home/fatima1153/.vsc
ode-server/bin/e3a5acfb517a44root 4866 1 0 15:26 ? 00:00:00 /init
fatima1+  4867      4866  0  15:26 ?        00:00:00 /init
fatima1+  4868      4867  0  15:26 pts/6    00:00:00 -bash
fatima1+  4909      4868  0  15:26 pts/6    00:00:00 ps -ef
fatima1153@DESKTOP-RMTBLS:~$ ps -ef | grep bash
fatima1+  3628 576 0 15:16 pts/5 00:00:00 /bin/bash --init-file /home/fatima1153/.vscode-server/bin/e3a5acfb517a443235981655413d56653310e92/out/vs/workbench/contrib/terminal/common/scripts/shellIntegratio
n-bash.sh
fatima1+  4868 4867 0 15:26 pts/6 00:00:00 -bash
fatima1+  4944 4868 0 15:26 pts/6 00:00:00 grep --color=auto bash
fatima1153@DESKTOP-RMTBLS:~$
```

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.

```
fatima1153@DESKTOP-RMTBL8S: ~
top - 15:36:39 up 42 min, 0 user, load average: 0.00, 0.02, 0.00
Tasks: 28 total, 1 running, 27 sleeping, 0 stopped, 0 zombie
Cpu(s):  0.0 us,  2.5 sy,  0.0 ni, 97.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem : 12679.9 total, 11986.9 free,  594.8 used,  342.4 buff/cache
MiB Swap: 4096.0 total, 4096.0 free,  0.0 used, 12085.1 avail Mem

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
 374 fatima+  20   0 31.9g 154804 55564 S   5.3   1.2   1:47.22 node
    1 root      20   0 1128  736   476 S   0.0   0.0   0:00.08 init
 118 root      20   0  908   84    20 S   0.0   0.0   0:00.03 init
 300 root      20   0  908   84    20 S   0.0   0.0   0:00.00 init
 301 root      20   0  908   84    20 S   0.0   0.0   0:00.00 init
 302 fatima+  20   0 2796 1060   968 S   0.0   0.0   0:00.00 sh
 303 fatima+  20   0 2796 1120  1024 S   0.0   0.0   0:00.00 sh
 308 fatima+  20   0 2796 1084   992 S   0.0   0.0   0:00.00 sh
 312 fatima+  20   0 11.3g 114600 52936 S   0.0   0.9   0:10.10 node
 323 root      20   0  908   84    20 S   0.0   0.0   0:00.00 init
 324 root      20   0  908   84    20 S   0.0   0.0   0:00.83 init
 325 fatima+  20   0 1016488 61924 44148 S   0.0   0.5   0:02.20 node
 342 root      20   0  908   84    20 S   0.0   0.0   0:00.00 init
 343 root      20   0  908   84    20 S   0.0   0.0   0:00.60 init
 347 fatima+  20   0 1012312 55288 44040 S   0.0   0.4   0:01.75 node
 353 fatima+  20   0 1261964 65140 47920 S   0.0   0.5   0:00.76 node
 402 root      20   0  908   84    20 S   0.0   0.0   0:00.00 init
 403 root      20   0  908   84    20 S   0.0   0.0   0:00.01 init
 404 fatima+  20   0 2796 1116  1024 S   0.0   0.0   0:00.00 sh
 405 fatima+  20   0 2796 1916  1808 S   0.0   0.0   0:00.00 sh
 415 fatima+  20   0 1013016 55516 45236 S   0.0   0.4   0:00.15 node
 438 fatima+  20   0 2796 1092   996 S   0.0   0.0   0:00.00 sh
 576 fatima+  20   0 1142276 80676 48452 S   0.0   0.6   0:05.95 node
3628 fatima+  20   0  6200  5468  3660 S   0.0   0.0   0:00.09 bash
 4866 root      20   0  908   84    20 S   0.0   0.0   0:00.00 init
 4867 root      20   0  908   84    20 S   0.0   0.0   0:00.01 init
 4868 fatima+  20   0  6068  5308  3592 S   0.0   0.0   0:00.12 bash
 5789 fatima+  20   0  9308  5204  3052 R   0.0   0.0   0:00.01 top
```

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.

- **Background:** Add **&** to run without blocking.

sleep 30 &

→ Terminal is free while the command runs.

- **Check background jobs:**

jobs

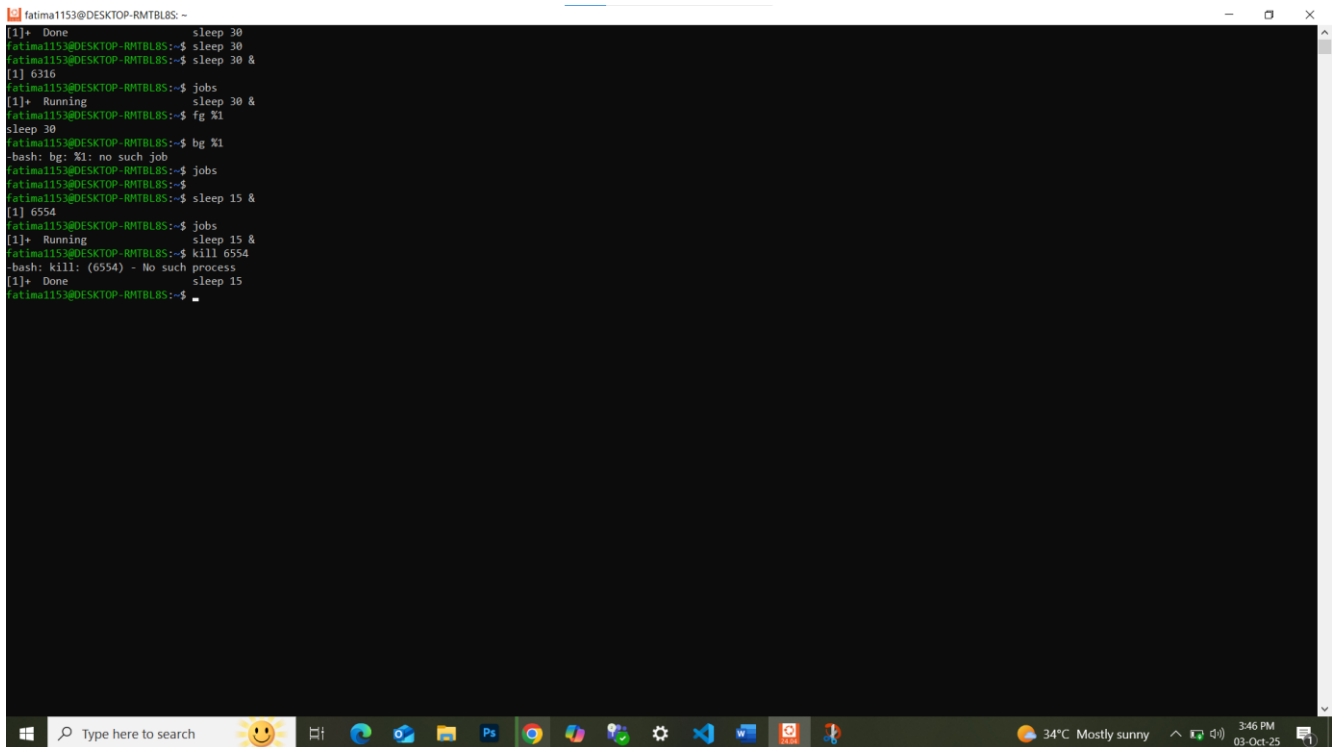
- ♦ **Bring a job to foreground:**

```
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
```



```
fatima1153@DESKTOP-RMTBLS: ~  
[1]+  Done                  sleep 30  
fatima1153@DESKTOP-RMTBLS:~$ sleep 30  
fatima1153@DESKTOP-RMTBLS:~$ sleep 30 &  
[1] 6316  
fatima1153@DESKTOP-RMTBLS:~$ jobs  
[1]+  Running              sleep 30 &  
fatima1153@DESKTOP-RMTBLS:~$ fg %1  
sleep 30  
fatima1153@DESKTOP-RMTBLS:~$ bg %1  
-bash: bg: %1: no such job  
fatima1153@DESKTOP-RMTBLS:~$ jobs  
fatima1153@DESKTOP-RMTBLS:~$  
fatima1153@DESKTOP-RMTBLS:~$ sleep 15 &  
[1] 6554  
fatima1153@DESKTOP-RMTBLS:~$ jobs  
[1]+  Running              sleep 15 &  
fatima1153@DESKTOP-RMTBLS:~$ kill 6554  
-bash: kill: (6554) - No such process  
[1]+  Done                  sleep 15  
fatima1153@DESKTOP-RMTBLS:~$
```

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
```

```
fatima1153@DESKTOP-RMTBLS:~  
fatima1153@DESKTOP-RMTBLS:~$ pidof sleep  
fatima1153@DESKTOP-RMTBLS:~$ sleep 100 &  
[1] 6965  
fatima1153@DESKTOP-RMTBLS:~$ pidof sleep  
6965  
fatima1153@DESKTOP-RMTBLS:~$ ps -ef | grep firefox  
fatima+ 7139  4868  0 15:51 pts/6    00:00:00 grep --color=auto firefox  
[1]- Done  
fatima1153@DESKTOP-RMTBLS:~$ sleep 100  
fatima1153@DESKTOP-RMTBLS:~$
```

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:

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

3. Kill it with:

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

```
fatima1153@DESKTOP-RMTBLS: ~  
fatima1153@DESKTOP-RMTBLS:~$ kill -9 3421  
-bash: kill: (3421) - No such process  
fatima1153@DESKTOP-RMTBLS:~$ sleep 10 &  
[2] 7597  
fatima1153@DESKTOP-RMTBLS:~$ kill -9 7597  
-bash: kill: (7597) - No such process  
[2]+ Done sleep 10  
fatima1153@DESKTOP-RMTBLS:~$ yes > /dev/null &  
[2] 7664  
fatima1153@DESKTOP-RMTBLS:~$ ps -ef | grep yes  
fatima1+ 7489 4868 99 15:55 pts/6 00:02:26 yes  
fatima1+ 7664 4868 99 15:57 pts/6 00:00:29 yes  
fatima1+ 7708 4868 0 15:58 pts/6 00:00:00 grep --color=auto yes  
fatima1153@DESKTOP-RMTBLS:~$ kill -9 <7597>  
-bash: syntax error near unexpected token `7597'  
fatima1153@DESKTOP-RMTBLS:~$ kill -9 <7489>  
-bash: syntax error near unexpected token `7489'  
fatima1153@DESKTOP-RMTBLS:~$ kill -9 7489  
fatima1153@DESKTOP-RMTBLS:~$ ps -ef | grep yes  
fatima1+ 7664 4868 99 15:57 pts/6 00:02:05 yes  
fatima1+ 7851 4868 0 15:59 pts/6 00:00:00 grep --color=auto yes  
[1]- Killed yes > /dev/null  
fatima1153@DESKTOP-RMTBLS:~$
```

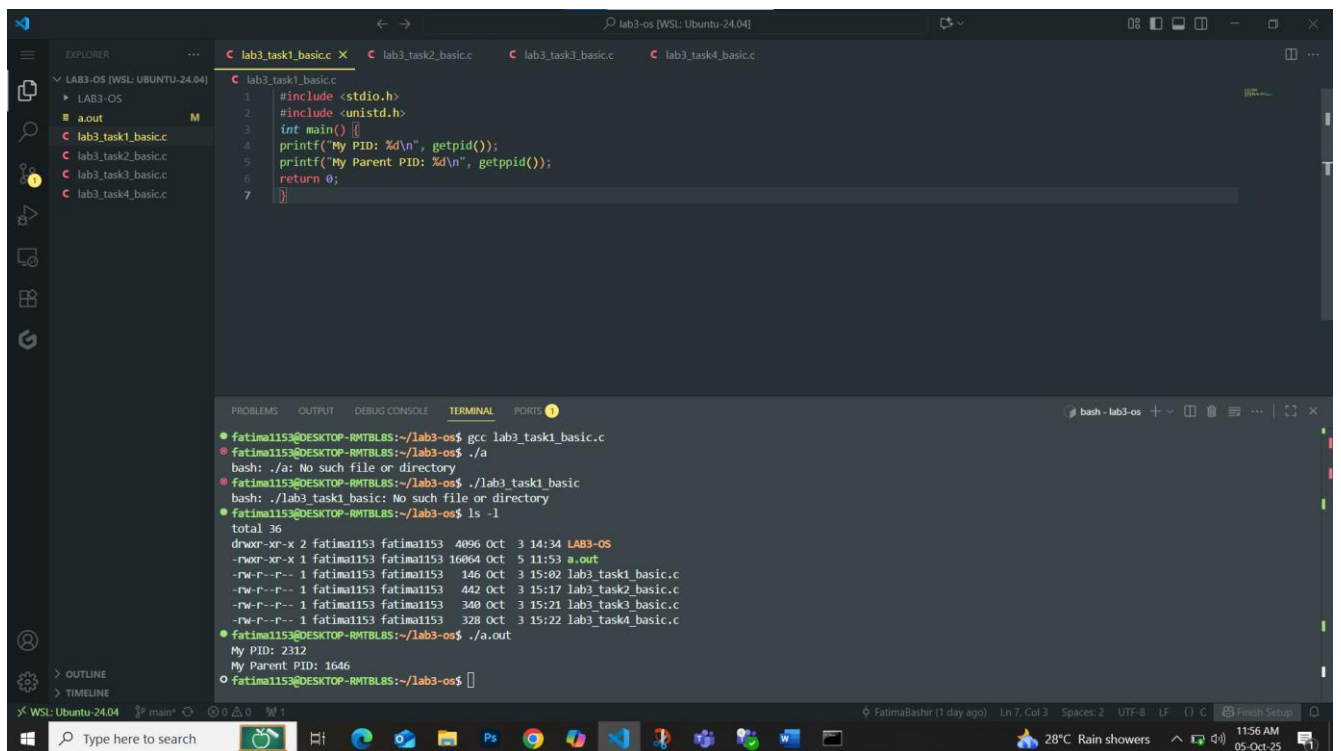
3. C Programs on Processes

Program 1: Print PID and PPID

```
#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

```
#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** .

The screenshot shows a VS Code editor with a C program in `lab3_task2_basic.c`. The program uses `fork()` to create a child process. The terminal output shows the execution of the program, which prints the parent and child PIDs.

```

1 #include <stdio.h>
2 #include <unistd.h>
3 int main() {
4     pid_t pid = fork();
5     if (pid == 0) {
6         // This block runs in the child process
7         printf("Child: PID-%d, Parent-%d\n", getpid(), getppid());
8     } else {
9         // This block runs in the parent process
10        printf("Parent: PID-%d, Child-%d\n", getpid(), pid);
11    }
12    return 0;
13 }

```

```

fatima1153@DESKTOP-RMTBLS8:~/lab3-os$ gcc lab3_task1_basic.c
/usr/bin/ld: cannot find gcc: No such file or directory
collect2: error: ld returned 1 exit status
fatima1153@DESKTOP-RMTBLS8:~/lab3-os$ gcc lab3_task2_basic.c
fatima1153@DESKTOP-RMTBLS8:~/lab3-os$ ./a.out
Parent: PID=2685, Child=2686
Child: PID=2686, Parent=2685
fatima1153@DESKTOP-RMTBLS8:~/lab3-os$

```

Program 3: Exec1 – Replacing a Process

```

#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;
}

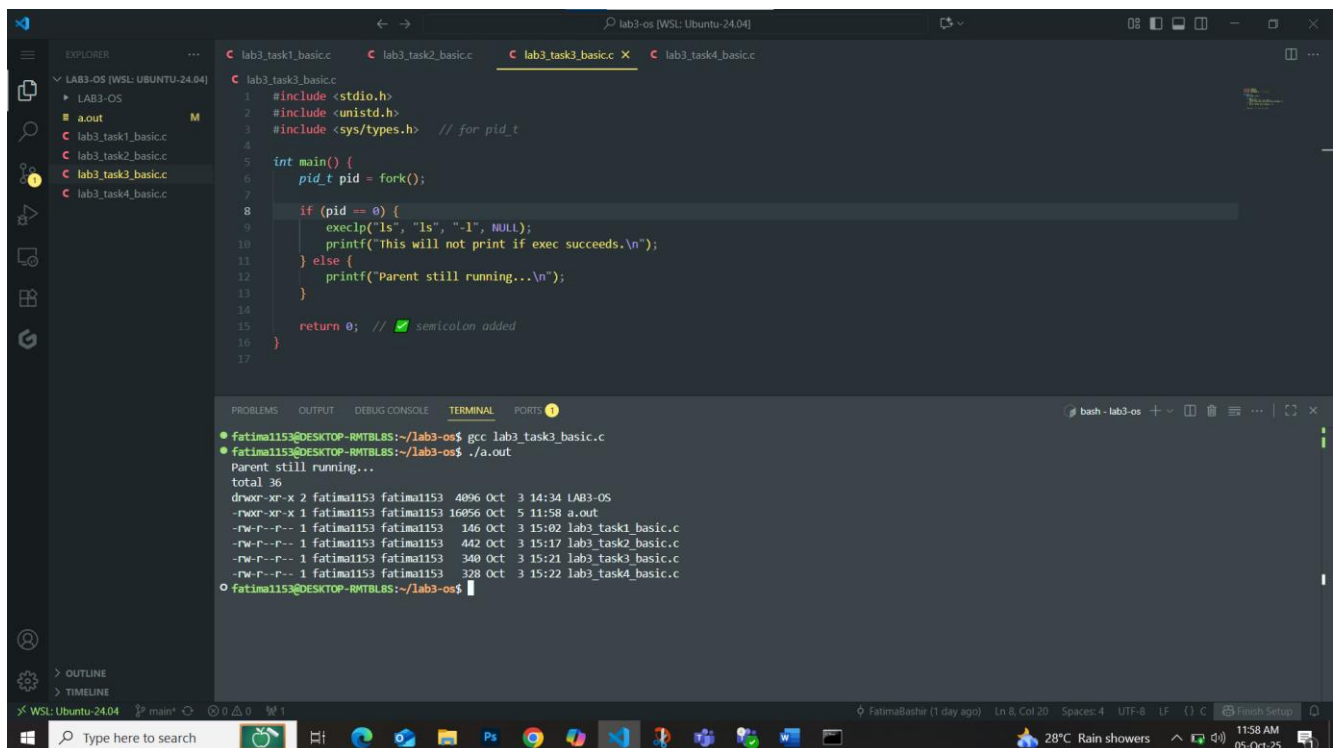
```

- `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.
- After `exec()`, the child **no longer runs our C code** – it becomes `ls`.
 - Parent is unaffected and continues normally.



The screenshot shows the Visual Studio Code editor with a C program in `lab3_task3_basic.c`. The program uses `fork()` to create a child process. In the child process (`pid == 0`), it calls `execlp("ls", "ls", "-l", NULL);` to replace the process image with the `ls` command. The parent process prints "Parent still running..." and then returns 0. The terminal output shows the execution of the program, including the output of the `ls` command in the child process.

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <sys/types.h> // for pid_t
4
5 int main() {
6     pid_t pid = fork();
7
8     if (pid == 0) {
9         execlp("ls", "ls", "-l", NULL);
10        printf("This will not print if exec succeeds.\n");
11    } else {
12        printf("Parent still running...\n");
13    }
14
15    return 0; // semicolon added
16 }
17
```

```
fatima1153@DESKTOP-RMTBLS:~/lab3-os$ gcc lab3_task3_basic.c
fatima1153@DESKTOP-RMTBLS:~/lab3-os$ ./a.out
Parent still running...
total 36
drwxr-xr-x 2 fatima1153 fatima1153 4096 Oct 3 14:34 LAB3-OS
-rwxr-xr-x 1 fatima1153 fatima1153 16056 Oct 5 11:58 a.out
-rw-r--r-- 1 fatima1153 fatima1153 146 Oct 3 15:02 lab3_task1_basic.c
-rw-r--r-- 1 fatima1153 fatima1153 442 Oct 3 15:17 lab3_task2_basic.c
-rw-r--r-- 1 fatima1153 fatima1153 340 Oct 3 15:21 lab3_task3_basic.c
-rw-r--r-- 1 fatima1153 fatima1153 328 Oct 3 15:22 lab3_task4_basic.c
fatima1153@DESKTOP-RMTBLS:~/lab3-os$
```

Program 4: Wait – Synchronization

```

#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**.

The screenshot shows a Visual Studio Code editor with a C program in `lab3_task4_basic.c`. The program uses `fork()` to create a child process that runs `ls -l`. The parent process then calls `waitpid()` to wait for the child to finish. The terminal output shows the execution of the program, including the directory listing and the parent's status.

```

1 #include <stdio.h>
2 #include <unistd.h>
3 #include <sys/wait.h>
4 int main() {
5     pid_t pid = fork();
6     if (pid == 0) {
7         execlp("ls", "ls", "-l", NULL);
8         printf("This will not print if exec succeeds.\n");
9     } else {
10        waitpid(pid, NULL, 0); // Wait for the child process to finish
11        printf("Parent still running...\n");
12    }
13    return 0;
14 }

```

```

fatima1153@DESKTOP-RMTBLS:~/lab3-os$ gcc lab3_task4_basic.c
fatima1153@DESKTOP-RMTBLS:~/lab3-os$ ./a.out
total 36
drwxr-xr-x 2 fatima1153 fatima1153 4096 Oct 3 14:34 LAB3-OS
-rwxr-xr-x 1 fatima1153 fatima1153 16096 Oct 5 11:59 a.out
-rw-r--r-- 1 fatima1153 fatima1153 146 Oct 3 15:02 lab3_task1_basic.c
-rw-r--r-- 1 fatima1153 fatima1153 442 Oct 3 15:17 lab3_task2_basic.c
-rw-r--r-- 1 fatima1153 fatima1153 340 Oct 3 15:21 lab3_task3_basic.c
-rw-r--r-- 1 fatima1153 fatima1153 328 Oct 3 15:22 lab3_task4_basic.c
Parent still running...
fatima1153@DESKTOP-RMTBLS:~/lab3-os$

```