

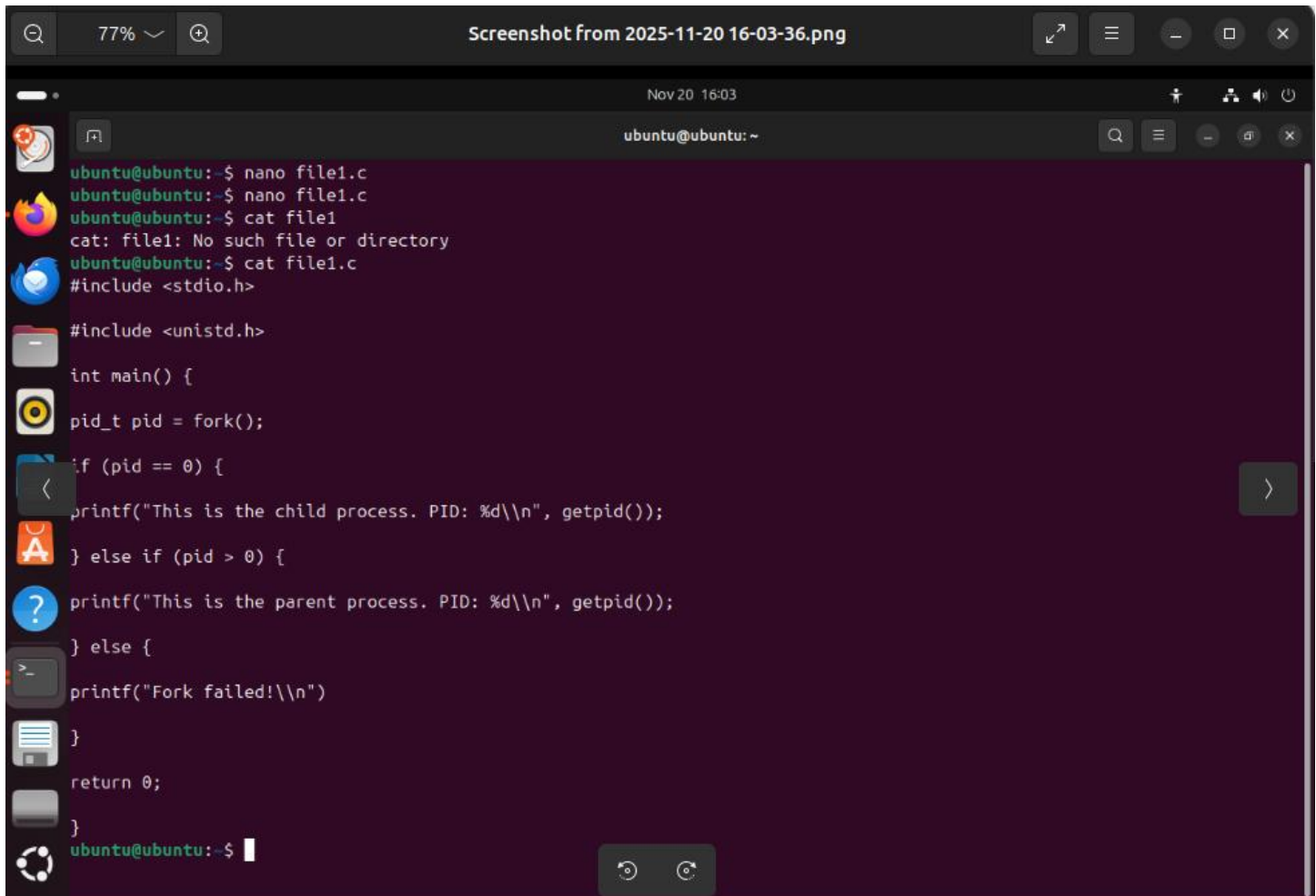
## Assignment\_2

### Task-1 :

First, I installed the GCC to run the C programs using this command :

```
sudo apt install gcc
```

Now I'm ready to make a .c file and write the code :



```
ubuntu@ubuntu:~$ nano file1.c
ubuntu@ubuntu:~$ nano file1.c
ubuntu@ubuntu:~$ cat file1
cat: file1: No such file or directory
ubuntu@ubuntu:~$ cat file1.c
#include <stdio.h>

#include <unistd.h>

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

    if (pid == 0) {
        printf("This is the child process. PID: %d\\n", getpid());
    } else if (pid > 0) {
        printf("This is the parent process. PID: %d\\n", getpid());
    } else {
        printf("Fork failed!\\n")
    }

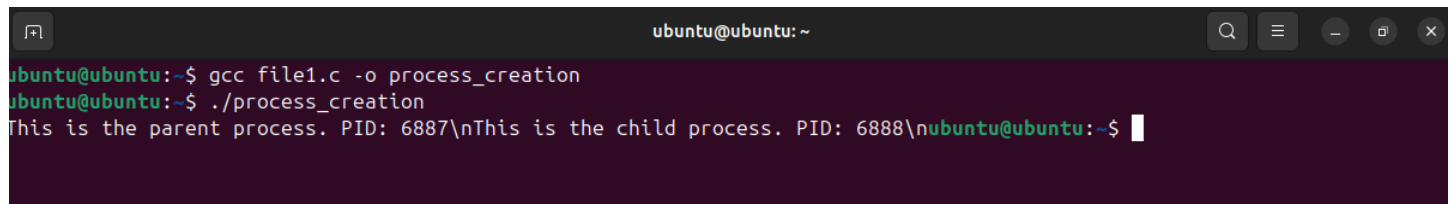
    return 0;
}
ubuntu@ubuntu:~$
```

**But before running, what this code actually do !!?**

First, I import the libraries that I need to IO and to use fork()

Fork() is a system call that create a process on linux , every process have a unique PID ,so we assign PID variable of type pid\_t to the fork() to make operations in the processes PIDs, fork returns 0 if there is a child process so I check using if condition and returns a statement and the PID of this process, if the PID>0 then it was a parent process the I make the same thing as before, if fork returns -1 that means it's failed so I handled it.

**After understanding the code Now, we can run : )**

A terminal window with a dark background and light green text. The window title is 'ubuntu@ubuntu: ~'. The terminal shows the following commands and output:

```
ubuntu@ubuntu:~$ gcc file1.c -o process_creation
ubuntu@ubuntu:~$ ./process_creation
This is the parent process. PID: 6887\nThis is the child process. PID: 6888\nubuntu@ubuntu:~$
```

**As expected it displays the child and parent with there PIDs.**

## Task-2:

We should create two files and link them, the files names : file1.c , file2.c

Every file has a simple code to test that the linker works.

### File2.c:

```
ubuntu@ubuntu:~$ nano file2.c
ubuntu@ubuntu:~$ cat file2.c
#include <stdio.h>

void hello() {

printf("Hello from file1!\n");

}
ubuntu@ubuntu:~$
```

It includes function called hello() that prints the statement when it called.

### File3.c:

```
ubuntu@ubuntu:~$ cat file3.c
void hello();

int main() {

hello();

return 0;

}
ubuntu@ubuntu:~$
```

Make a function declaration then it called into the main.

Our Goal to link this two files, so we use the commands:

`gcc file2.c file3.c -o output`

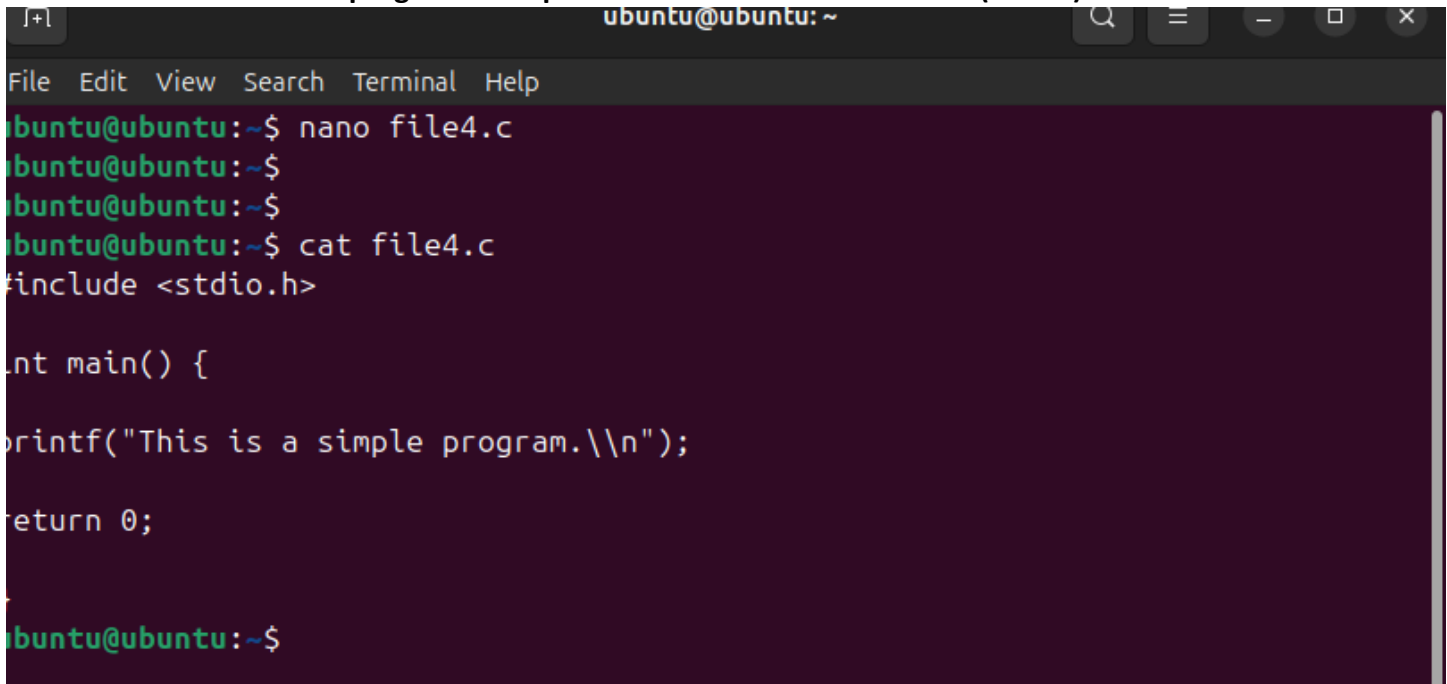
`./output`

```
ubuntu@ubuntu:~$ gcc file2.c file3.c -o output
ubuntu@ubuntu:~$ ./ output
bash: ./: Is a directory
ubuntu@ubuntu:~$ ./output
Hello from file1!\nubuntu@ubuntu:~$ -
```

**It runs successfully and prints the statement.**

### Task-3:

We should write a program to inspect the libraries it uses with ldd (loader):

A terminal window titled 'ubuntu@ubuntu: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
ubuntu@ubuntu:~$ nano file4.c
ubuntu@ubuntu:~$
ubuntu@ubuntu:~$
ubuntu@ubuntu:~$ cat file4.c
#include <stdio.h>

int main() {
    printf("This is a simple program.\n");
    return 0;
}
```

Very simple program that prints a statement.

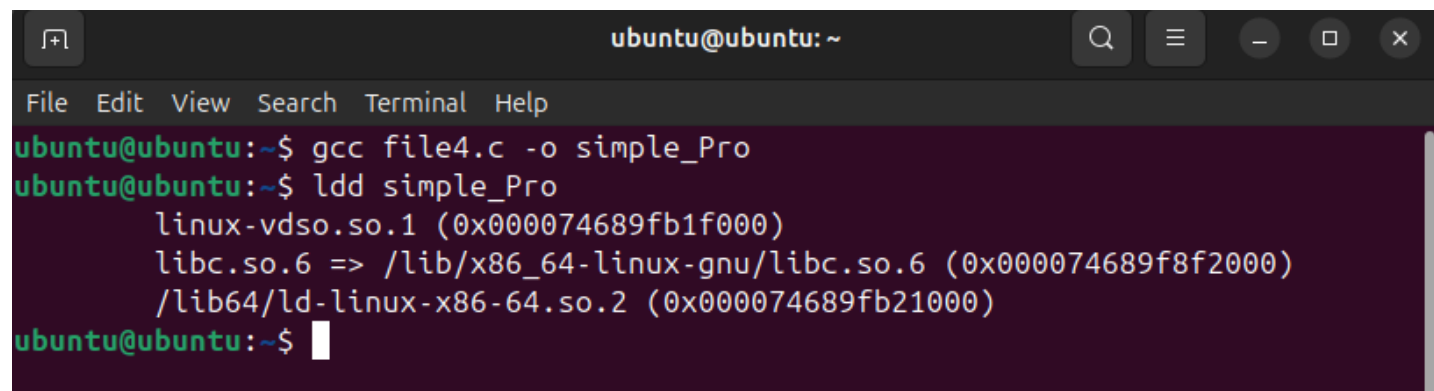
Now we should run this program:

```
gcc file4.c -o simple_pro
```

we should find an output named simple\_pro. Use

ldd to list the dynamic libraries:

```
ldd simple_pro
```

A terminal window titled 'ubuntu@ubuntu: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

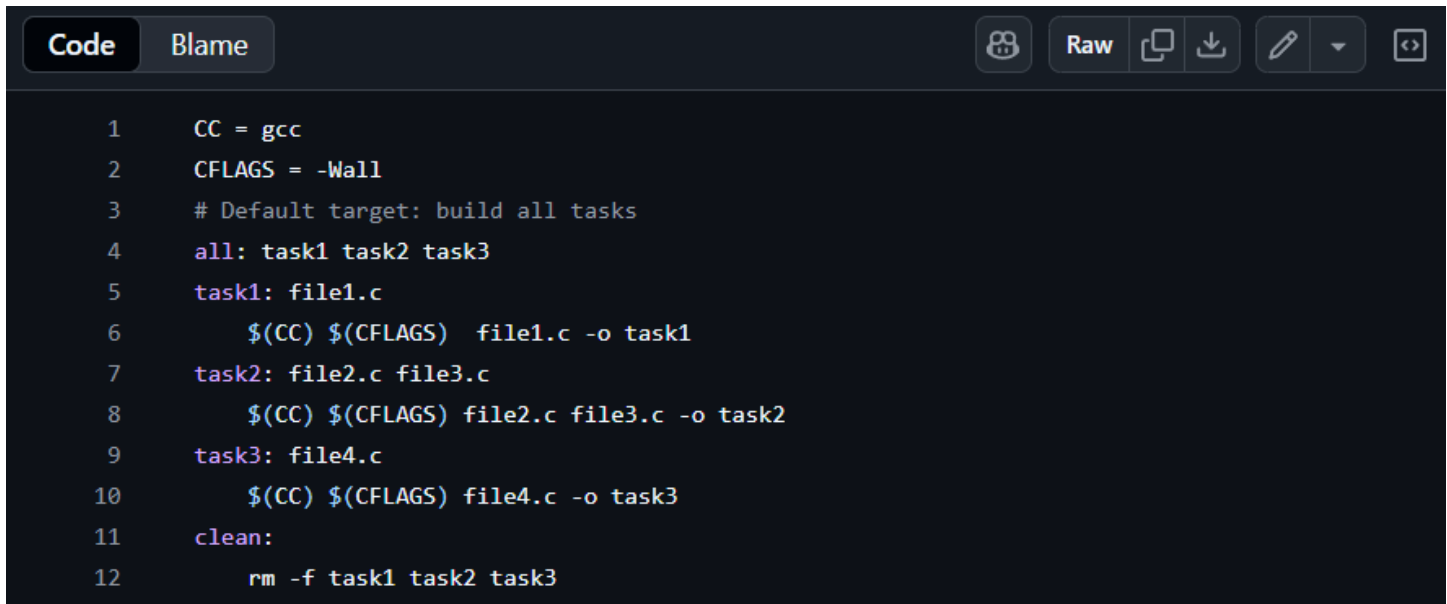
```
ubuntu@ubuntu:~$ gcc file4.c -o simple_Pro
ubuntu@ubuntu:~$ ldd simple_Pro
        linux-vdso.so.1 (0x000074689fb1f000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x000074689f8f2000)
        /lib64/ld-linux-x86-64.so.2 (0x000074689fb21000)
```

the libraries that added dynamically was displayed !

## Makefile:

I want to make a tool that compiles the programs I write in several files, i make a search to learn how to do it , and I find that source [Makefile Tutorial By Example](#) .

Then I write the code into makefile file like that:



```
1  CC = gcc
2  CFLAGS = -Wall
3  # Default target: build all tasks
4  all: task1 task2 task3
5  task1: file1.c
6      $(CC) $(CFLAGS) file1.c -o task1
7  task2: file2.c file3.c
8      $(CC) $(CFLAGS) file2.c file3.c -o task2
9  task3: file4.c
10     $(CC) $(CFLAGS) file4.c -o task3
11  clean:
12     rm -f task1 task2 task3
```

Let's break down the difficult parts, I used -wall to make sure the file executed as I want and return to me a statement for that, all files putted in something like switch in java, and I make the linker between file1 and file2 as required, at the end I clean the results files if you tell the program.

**That's All, we Write the code examples, Explain each one and what it does, and take Screenshots of compiling and running the code.**