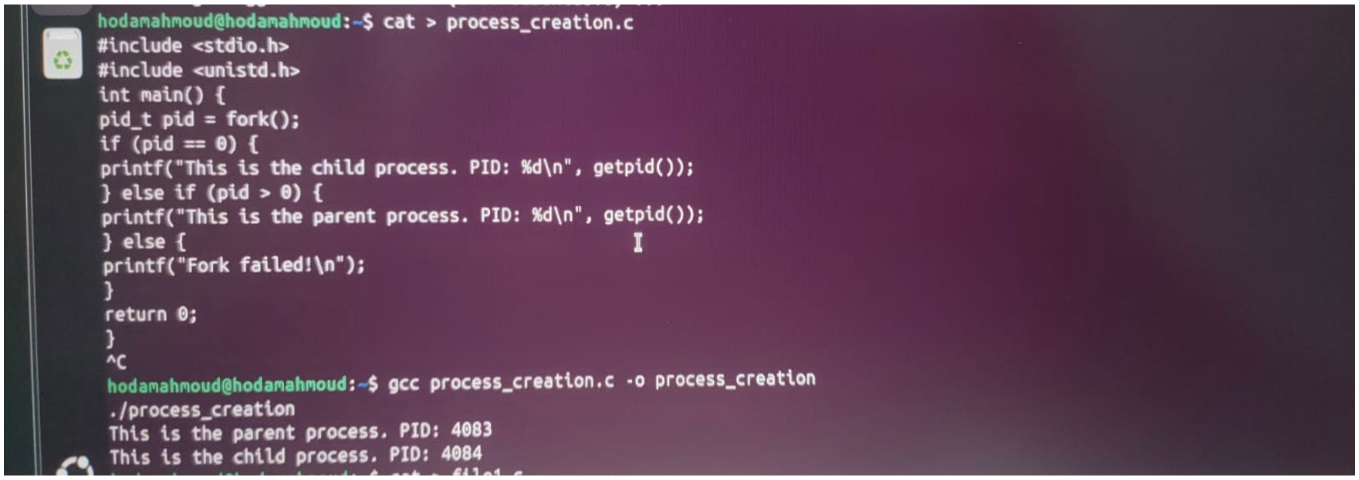
- Write the code examples you have seen in Lab-5. Explain each one and what it does

process_creation.c



```
hodamahmoud@hodamahmoud:~$ cat > process_creation.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;
}
^C
hodamahmoud@hodamahmoud:~$ gcc process_creation.c -o process_creation
./process_creation
This is the parent process. PID: 4083
This is the child process. PID: 4084
```

fork() : Creates a new process called a child process.

getpid() :returns the PID of the child process.

Returns a value to both parent and child:

0 → returned to the child process

Positive number (child PID) → returned to the parent process

-1 → if fork fails (no process created)

(pid == 0)

printf("This is the child process. PID: %d\n", getpid());

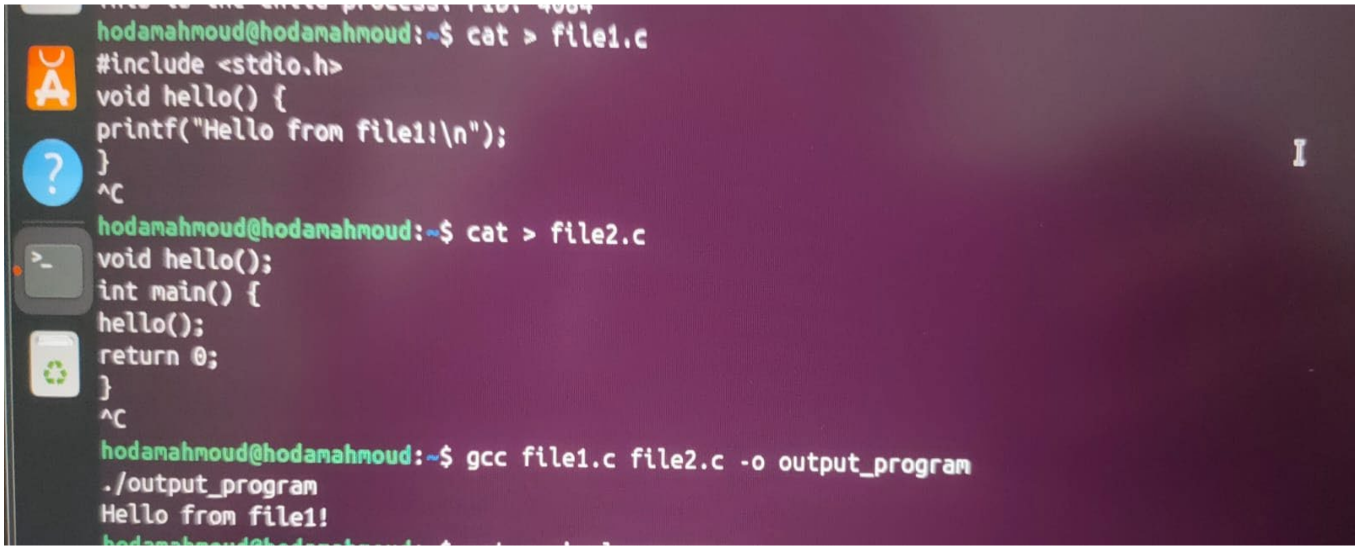
Parent process (pid > 0)

printf("This is the parent process. PID: %d\n", getpid());

(pid < 0)

printf("Fork failed!\n");

output_program



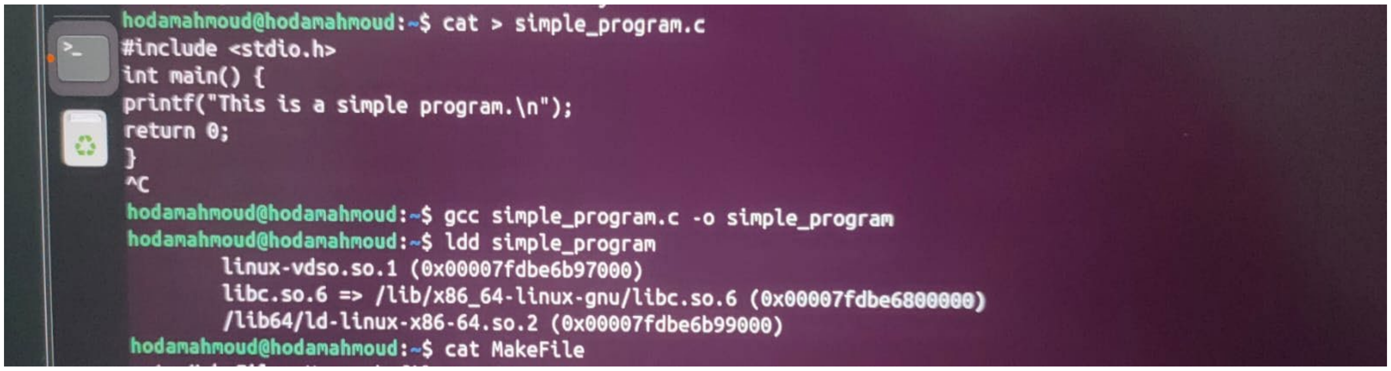
```
hodamahmoud@hodamahmoud:~$ cat > file1.c
#include <stdio.h>
void hello() {
printf("Hello from file1!\n");
}
^C
hodamahmoud@hodamahmoud:~$ cat > file2.c
void hello();
int main() {
hello();
return 0;
}
^C
hodamahmoud@hodamahmoud:~$ gcc file1.c file2.c -o output_program
./output_program
Hello from file1!
```

file1.c-:

This file defines the function hello(). this is where the full code (the body of the function) actually exists. When any other file calls hello(), this is the function that will run.

file2.c-:
This file contains the main() function — the starting point of the program. hello()is a function declaration. This program will execute the hello function
gcc file1.c file2.c -o output_program:-
this command will link this two files into one called output_program and execute it .

simple_program.c

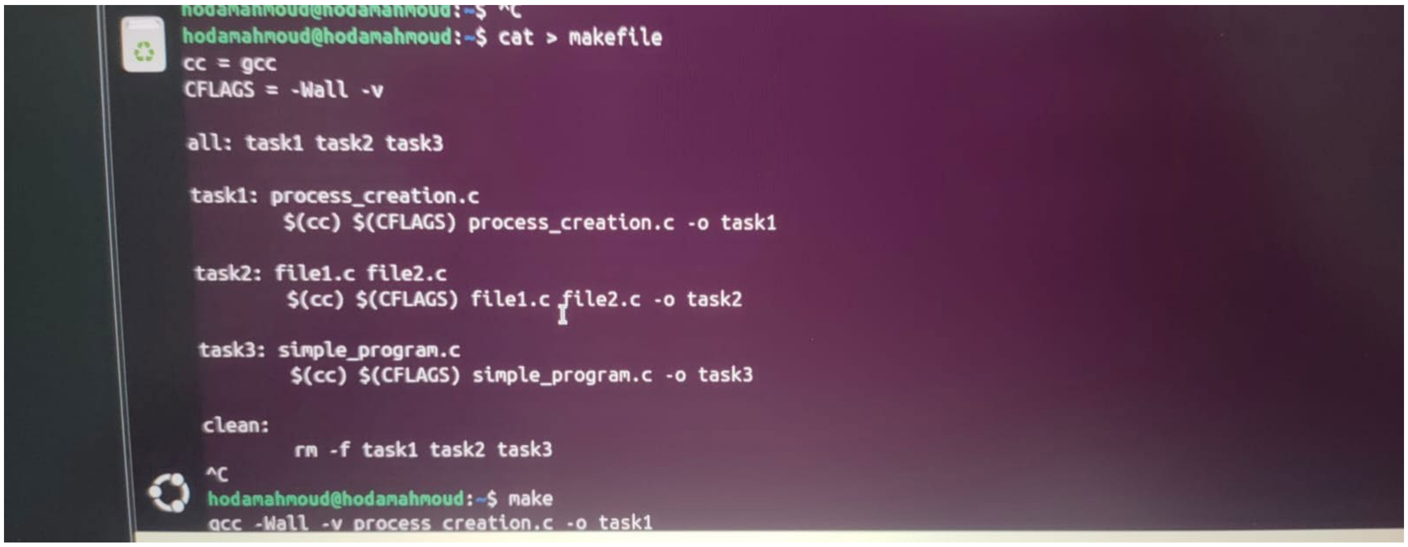


```
hodamahmoud@hodamahmoud:~$ cat > simple_program.c
#include <stdio.h>
int main() {
printf("This is a simple program.\n");
return 0;
}
^C
hodamahmoud@hodamahmoud:~$ gcc simple_program.c -o simple_program
hodamahmoud@hodamahmoud:~$ ldd simple_program
        linux-vdso.so.1 (0x00007fdbe6b97000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fdbe6800000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fdbe6b99000)
hodamahmoud@hodamahmoud:~$ cat MakeFile
```

This program just prints a message (This is a simple program).
gcc simple_program.c -o simple_program: This creates an executable file named simple_program.
ldd simple_program: This command lists all shared libraries the program uses.

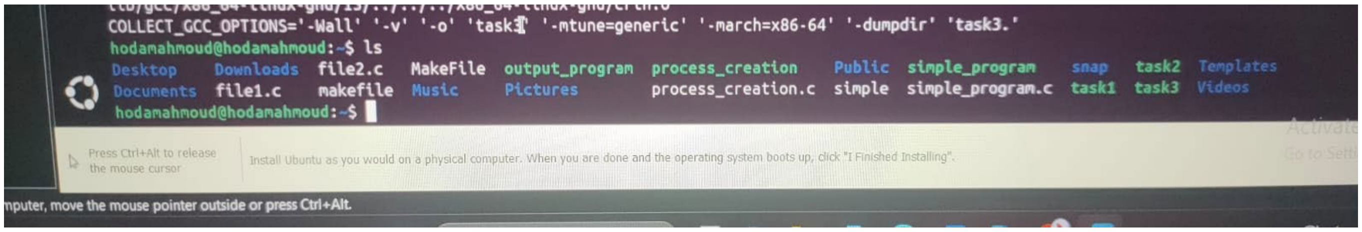
- **Write a makefile to compile each of the examples. Run each of the programs**

MakeFile



```
hodamahmoud@hodamahmoud:~$ ^C
hodamahmoud@hodamahmoud:~$ cat > makefile
cc = gcc
CFLAGS = -Wall -v

all: task1 task2 task3

task1: process_creation.c
        $(cc) $(CFLAGS) process_creation.c -o task1

task2: file1.c file2.c
        $(cc) $(CFLAGS) file1.c file2.c -o task2

task3: simple_program.c
        $(cc) $(CFLAGS) simple_program.c -o task3

clean:
        rm -f task1 task2 task3
^C
hodamahmoud@hodamahmoud:~$ make
gcc -Wall -v process_creation.c -o task1
```

I want to make a tool that compiles the programs I write in several files. I made a search to learn how to do it, and I found the source:
https://www.youtube.com/watch?v=a8mPKBxQ9No&pp=ygUbbWFrZWZpbGUgdHV0b3JpYWwgYyBmb3IgZ2Nj
Then I wrote the code into the makefile like that:
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 are putted in something like a switch in Java, and I made the linker between file1 and file2 as required. At the end, I clean the results files if you tell the program.



```
COLLECT_GCC_OPTIONS='-Wall' '-v' '-o' 'task3' '-mtune=generic' '-march=x86-64' '-dumpdir' 'task3.'
hodamahmoud@hodamahmoud:~$ ls
Desktop    Downloads  file2.c   MakeFile  output_program  process_creation    Public  simple_program    snap   task2  Templates
Documents  file1.c    makefile  Music     Pictures        process_creation.c  simple  simple_program.c  task1  task3  Videos
hodamahmoud@hodamahmoud:~$
```

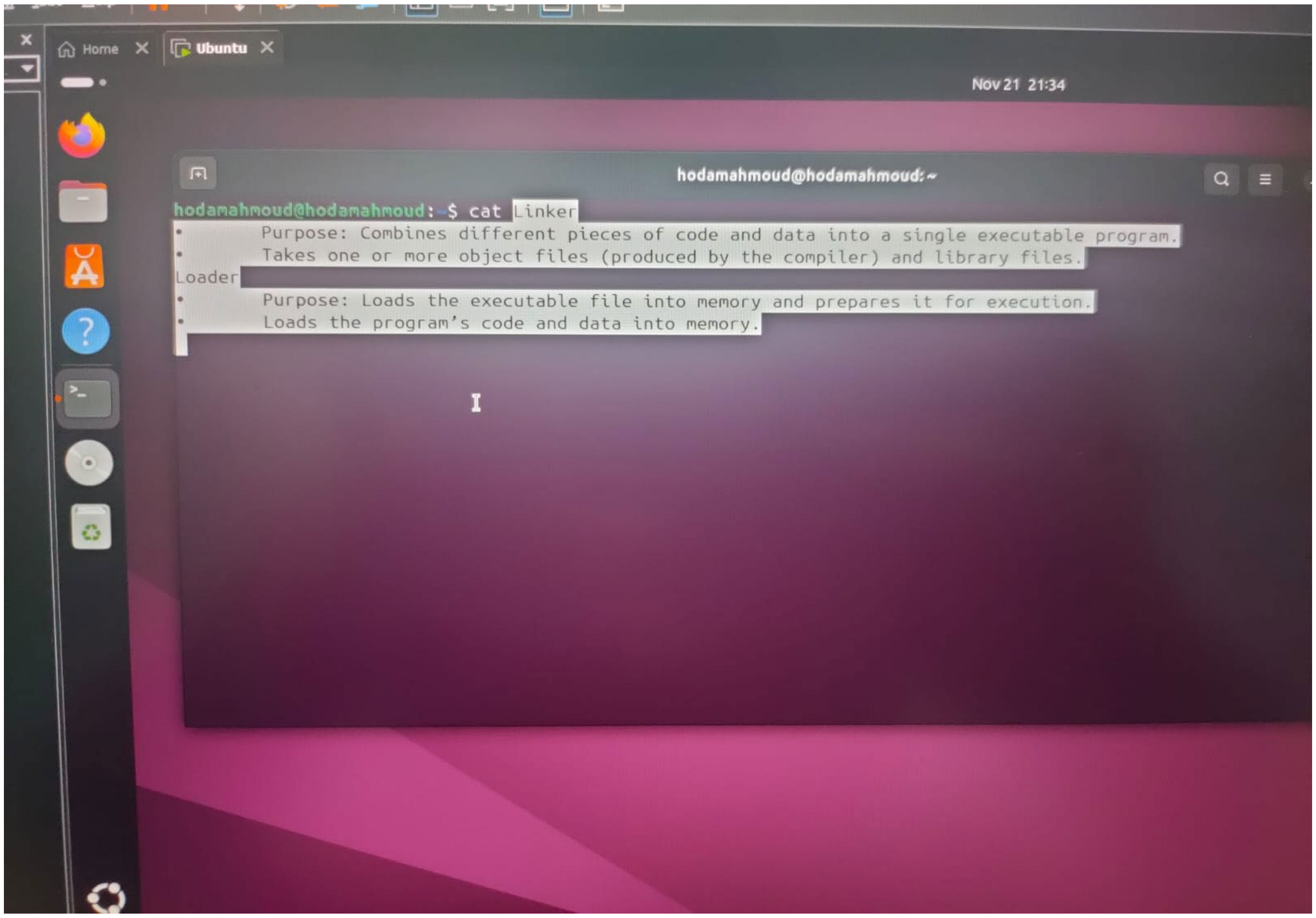
Press Ctrl+Alt to release the mouse cursor

Install Ubuntu as you would on a physical computer. When you are done and the operating system boots up, click "I Finished Installing".

mputer, move the mouse pointer outside or press Ctrl+Alt.

Finalllllllly , the output is boooom (task1 , task2 , task3) with the green color.

Linker and Loader .txt



```
hodamahmoud@hodamahmoud:~$ cat Linker
•           Purpose: Combines different pieces of code and data into a single executable program.
•           Takes one or more object files (produced by the compiler) and library files.
Loader
•           Purpose: Loads the executable file into memory and prepares it for execution.
•           Loads the program's code and data into memory.
```