Lab N°02 : Basic Mechanisms of Operating System

Objective: Compilation and Linker / Process States

- View the compilation process step by step
- Create simple processes.
- Observe Linux processes in all their states

Exercise II.1: (Compiler environment : gcc)

Q1) Create a file named **hello.c** and type in the program below.

```
#include <stdio.h>
int main() {
      printf("Hello World!\n");
      return 0;
}
```

Q2) Compile the program using the command : **make hello**

What do you notice?

Q3) Compile it with command **gcc hello.c -o hello**

What do you notice?

Q4) Run it with **./hello** .

Exercise II.2: (Compilation to Linker Step by Step)

The compilation process (in the broadest sense) breaks down into 4 steps:

Pre-processing ; Compilation ; Assembly ; Linker (link edition).

By default, **GCC** deletes all intermediate files (i.e. **.i**, **.s**, **.o**, etc) when they are not needed anymore, so you won't be able to see them in your working directly.

But we will ask it to stop after a particular step: (Use the same program as in **Exercise II.1)**

Q1) Run the **pre-processing**, but do not compile (the file named **hello.i**).

Q2) Open the **hello.i** file and observe the resulting code. What do you think?

Q3) Run the **pre-processing** and then **compile** the result, but does not assemble (the file named **hello.s**).

Q4) Open the **hello.s** file and observe the resulting code. What do you think?

Q5) Run the **pre-processing**, **compiles**, and **assembles**, but does not run the linker (the file named **hello.o**).

Q6) Open the **hello.o** file and observe the resulting code. What do you think?

Q7) Run the **pre-processing**, **compiles**, **assembles**, and **linker** (the file named **hello**).

Q8) Run by the command **./hello** and observe the resulting code. What do you think?

Q9) What do you observe about the file sizes? Explain why it's different.

Exercise II.3: (Compilation, Linker and Library)

Let's assume that the "**cal_avg.c**" file contains a main program written in "**C**" language. It consists in calculating and displaying the average of real numbers read from the keyboard.

Average calculations are performed using the following functions:

- The "**avg2(arg1,arg2)**" function calculates and returns the average of the real numbers "**arg1**" and "**arg2**".
- The "**avg3(arg1, arg2, arg3)**" function calculates and returns the average of the real numbers "**arg1**", "**arg2**" and "**arg3**".

Q1) Assume that the "**avg2()**" and "**avg3()**" functions are defined in the "**cal_avg.c**" file and written after the "**main()**" function.

a) Write the "**cal_avg.c**" program, then compile and run it.

b) Display executable file size.

c) Recompile with "**-static**" option and display executable file size.

d) What can we conclude?

Q2) Now, let's assume that the "**avg2()**" and "**avg3()**" functions are defined in separate files: the "**avg2()**" function is defined in a file called "**average2.c**" and the "**avg3()**" function is defined in another file called "**average3.c**".

a) Modify the "**cal_avg.c**" program, then compile and run it.

Q3) Write a file named "**helpful.h**" containing the prototypes of the "**avg2()**" and "**avg3()**" functions, then save it in the current directory. Include this file in the main program and recompile.

Q4) Recompile the program using the "**make**" command.

Q5) Create the "**~/include**" directory and move the "**helpful.h**" file into it. Then recompile and run the main program.

Q6) Create a static library named "**liboutils.a**" and a dynamic library named "**liboutils.so**" containing the files "**average2.o**" and "**average3.o**". Save these libraries in the "**~/lib**" directory.

Compile and run the "**cal_avg.c**" program using these libraries.