Benjamin DAVID

Robin VAN DESSEL

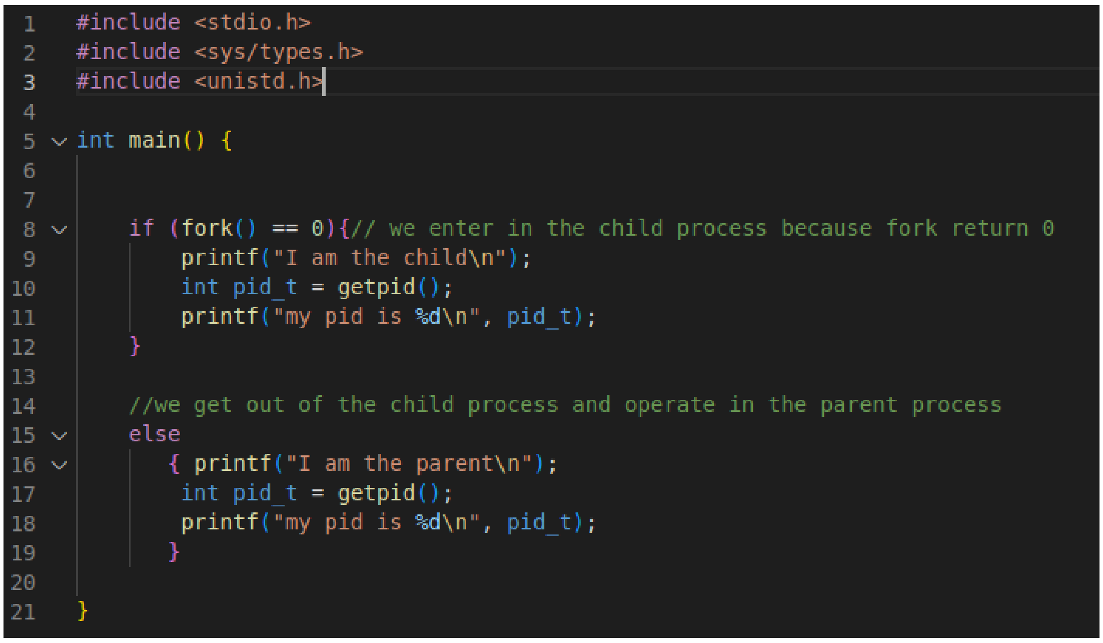
Group 5

Lab 1 Report

1. Creating and Running a Process (1) - fork

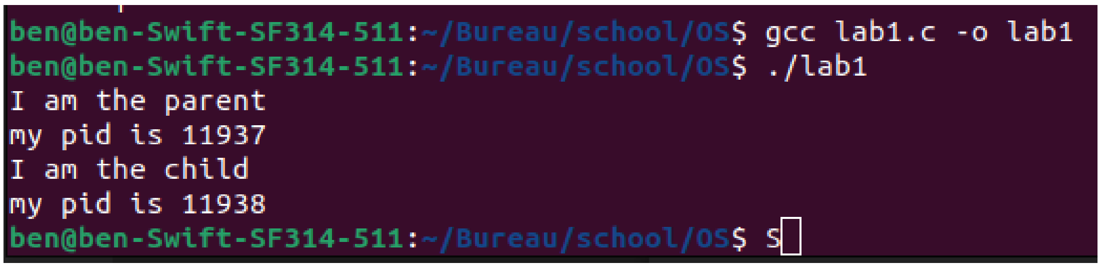
2. After a fork call, a new child process is created. It is an exact copy of parent process. Every process is defined by a unique PID (Process ID). The child has a different PID from his parent, but they share the same PPID (Parent Process ID).

3.



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

int main() {


    if (fork() == 0){// we enter in the child process because fork return 0
        printf("I am the child\n");
        int pid_t = getpid();
        printf("my pid is %d\n", pid_t);
    }

    //we get out of the child process and operate in the parent process
    else
       { printf("I am the parent\n");
        int pid_t = getpid();
        printf("my pid is %d\n", pid_t);
       }

}
```

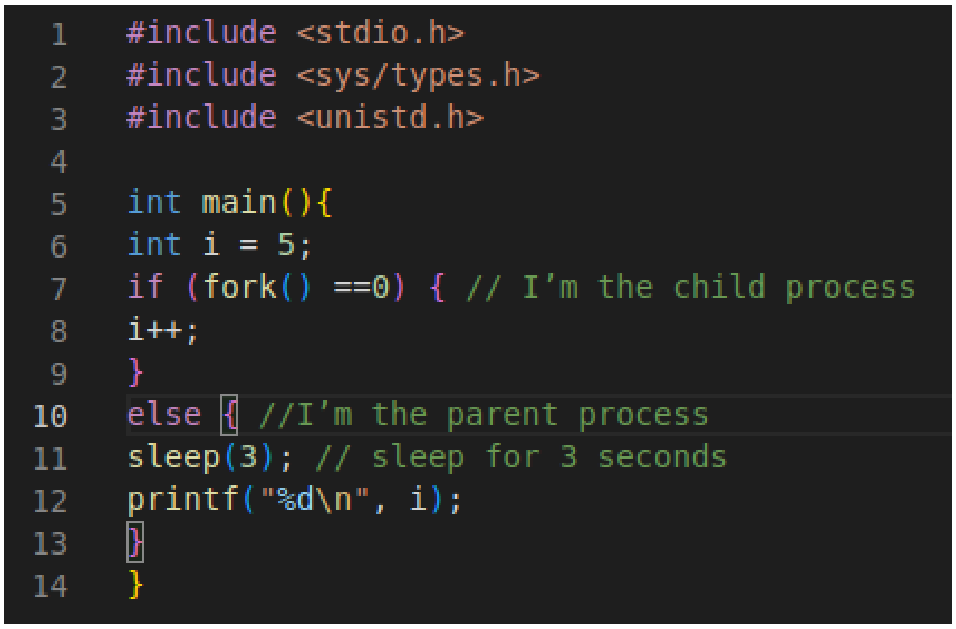
The *fork()* creates a child process. The *fork()* function returns 0 in the child process, so we are in the first condition. In the parent process we enter in the else loop because the fork returns a different PID, so we are in the parent process. In each process, we return his PID and prints it.



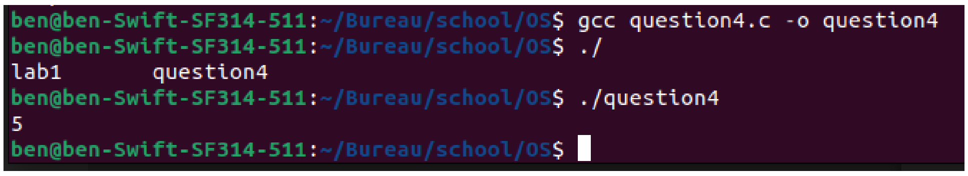
```
ben@ben-Swift-SF314-511:~/Bureau/school/OS$ gcc lab1.c -o lab1
ben@ben-Swift-SF314-511:~/Bureau/school/OS$ ./lab1
I am the parent
my pid is 11937
I am the child
my pid is 11938
ben@ben-Swift-SF314-511:~/Bureau/school/OS$ $
```

We compile and execute the C code, and this is the result. We can see that PID are different between child and parent process.

4.



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

int main(){
int i = 5;
if (fork() ==0) { // I'm the child process
i++;
}
else { //I'm the parent process
sleep(3); // sleep for 3 seconds
printf("%d\n", i);
}
}
```

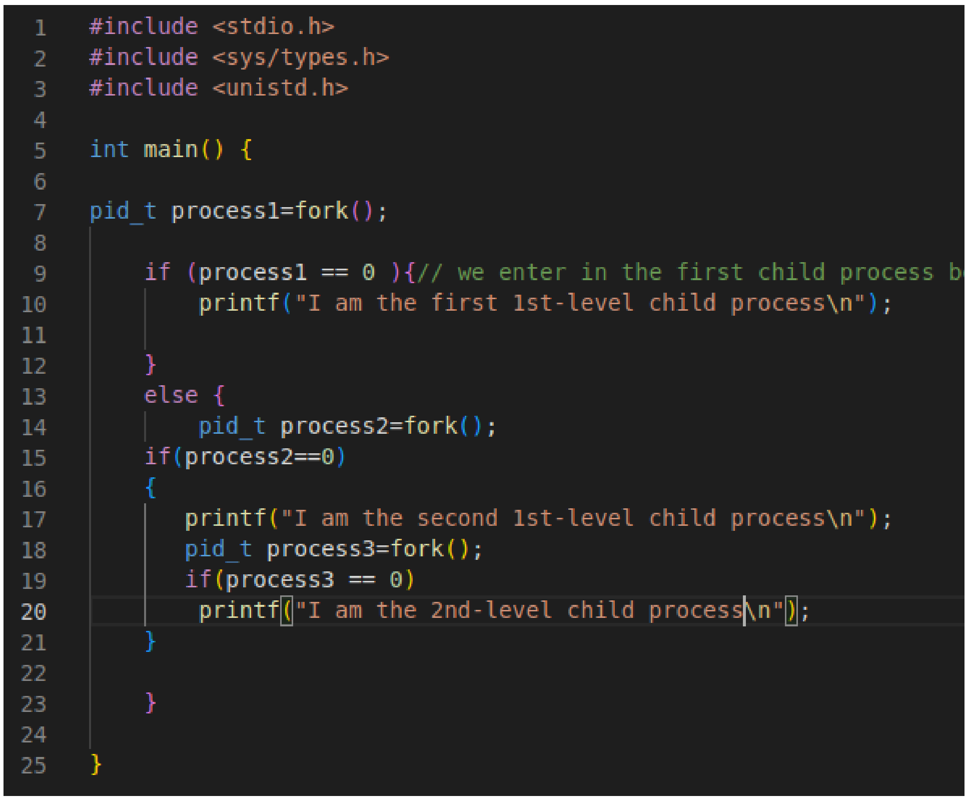


```
ben@ben-Swift-SF314-511:~/Bureau/school/OS$ gcc question4.c -o question4
ben@ben-Swift-SF314-511:~/Bureau/school/OS$ ./
lab1        question4
ben@ben-Swift-SF314-511:~/Bureau/school/OS$ ./question4
5
ben@ben-Swift-SF314-511:~/Bureau/school/OS$ 
```

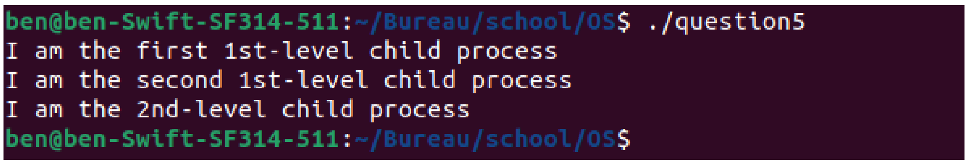
During the child process we incremented the "i" value by one, but in the parent process as a result we still have a 5 value.

We logically need the *sleep()* because when we use *fork()*, it duplicates the code and runs both process at the same time. We must wait for the child process to increment "i" before printing it in the parent process. Otherwise, it can create Race condition.

5. https://stackoverflow.com/questions/6542491/how-to-create-two-processes-from-a-single-parent



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

int main() {

pid_t process1=fork();

    if (process1 == 0 ){// we enter in the first child process b
        printf("I am the first 1st-level child process\n");

    }
    else {
        pid_t process2=fork();
    if(process2==0)
    {
       printf("I am the second 1st-level child process\n");
       pid_t process3=fork();
       if(process3 == 0)
        printf("I am the 2nd-level child process\n");
    }

    }

}
```

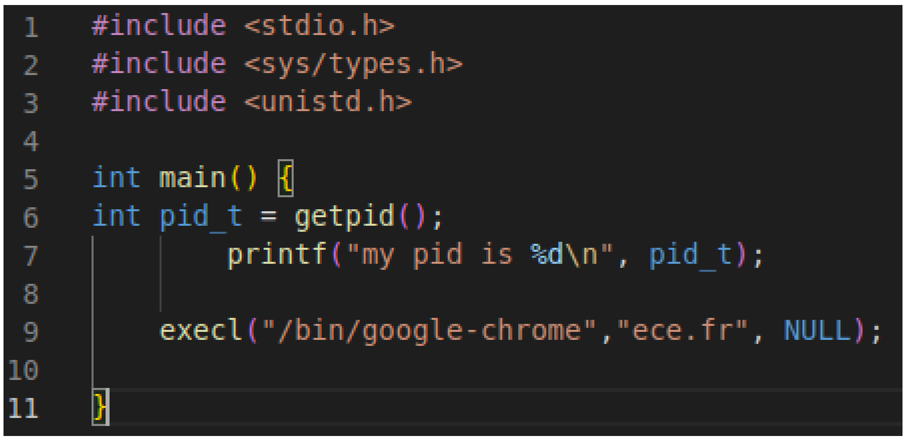


```
ben@ben-Swift-SF314-511:~/Bureau/school/OS$ ./question5
I am the first 1st-level child process
I am the second 1st-level child process
I am the 2nd-level child process
ben@ben-Swift-SF314-511:~/Bureau/school/OS$
```

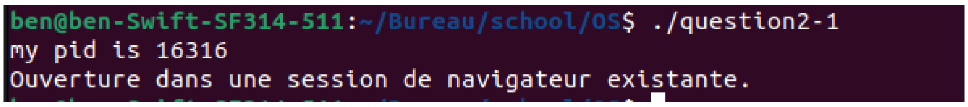
The first *fork()* creates the first 1st-level child process. The second *fork()* is called in the parent process and creates the second 1st-level child process. Then, the last *fork()* is used in the second 1st-level child process and creates a 2nd-level child process

2. Creating and Running a Process (2) - exec

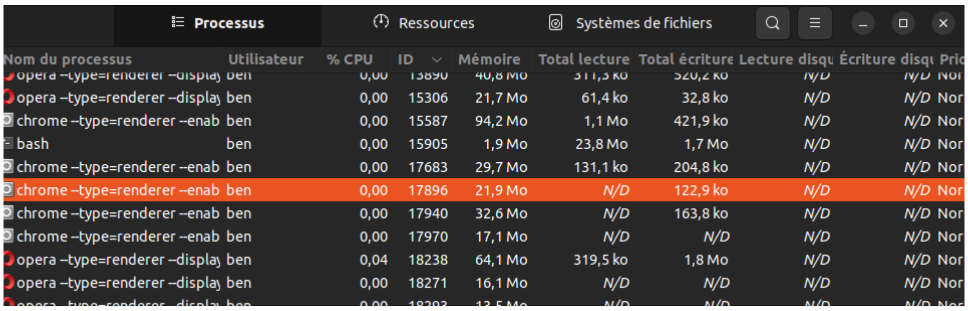
2.



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

int main() {
int pid_t = getpid();
        printf("my pid is %d\n", pid_t);

    execl("/bin/google-chrome","ece.fr", NULL);

}
```



```
ben@ben-Swift-SF314-511:~/Bureau/school/OS$ ./question2-1
my pid is 16316
Ouverture dans une session de navigateur existante.
```

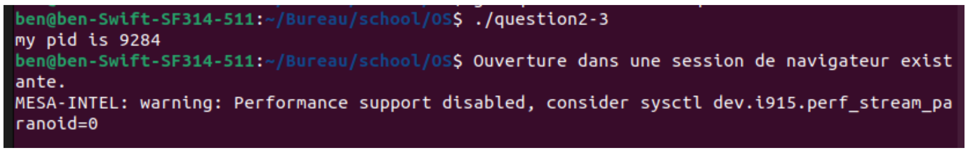


Nom du processus	Utilisateur	% CPU	ID	Mémoire	Total lecture	Total écriture	Lecture disqu	Écriture disqu	Pric
opera --type=renderer --display	ben	0,00	13890	40,8 Mo	311,3 ko	520,2 ko	N/D	N/D	Nor
opera --type=renderer --display	ben	0,00	15306	21,7 Mo	61,4 ko	32,8 ko	N/D	N/D	Nor
chrome --type=renderer --enab	ben	0,00	15587	94,2 Mo	1,1 Mo	421,9 ko	N/D	N/D	Nor
bash	ben	0,00	15905	1,9 Mo	23,8 Mo	1,7 Mo	N/D	N/D	Nor
chrome --type=renderer --enab	ben	0,00	17683	29,7 Mo	131,1 ko	204,8 ko	N/D	N/D	Nor
chrome --type=renderer --enab	ben	0,00	17896	21,9 Mo	N/D	122,9 ko	N/D	N/D	Nor
chrome --type=renderer --enab	ben	0,00	17940	32,6 Mo	N/D	163,8 ko	N/D	N/D	Nor
chrome --type=renderer --enab	ben	0,00	17970	17,1 Mo	N/D	N/D	N/D	N/D	Nor
opera --type=renderer --display	ben	0,04	18238	64,1 Mo	319,5 ko	1,8 Mo	N/D	N/D	Nor
opera --type=renderer --display	ben	0,00	18271	16,1 Mo	N/D	N/D	N/D	N/D	Nor
opera --type=renderer --display	ben	0,00	18293	13,5 Mo	N/D	N/D	N/D	N/D	Nor

The PID of the new running process is different from the original one because we can't find any process with the printed PID in the task manager

3. After fork() the parent and the child become different processes and they don't share the same memory space. But child process is a copy of the parent process so that's why they have the same resources.

4.

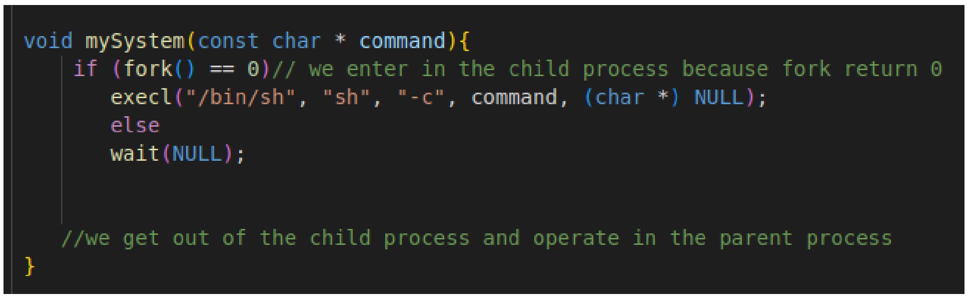


```
ben@ben-Swift-SF314-511:~/Bureau/school/OS$ ./question2-3
my pid is 9284
ben@ben-Swift-SF314-511:~/Bureau/school/OS$ Ouverture dans une session de navigateur exist
ante.
MESA-INTEL: warning: Performance support disabled, consider sysctl dev.i915.perf_stream_pa
ranoid=0
```

The program displays the process id and execute the exec call but doesn't display the printf() line. That's because exec family replaces the current process image with a new process image so nothing happens after execl.

3. Writing your own shell

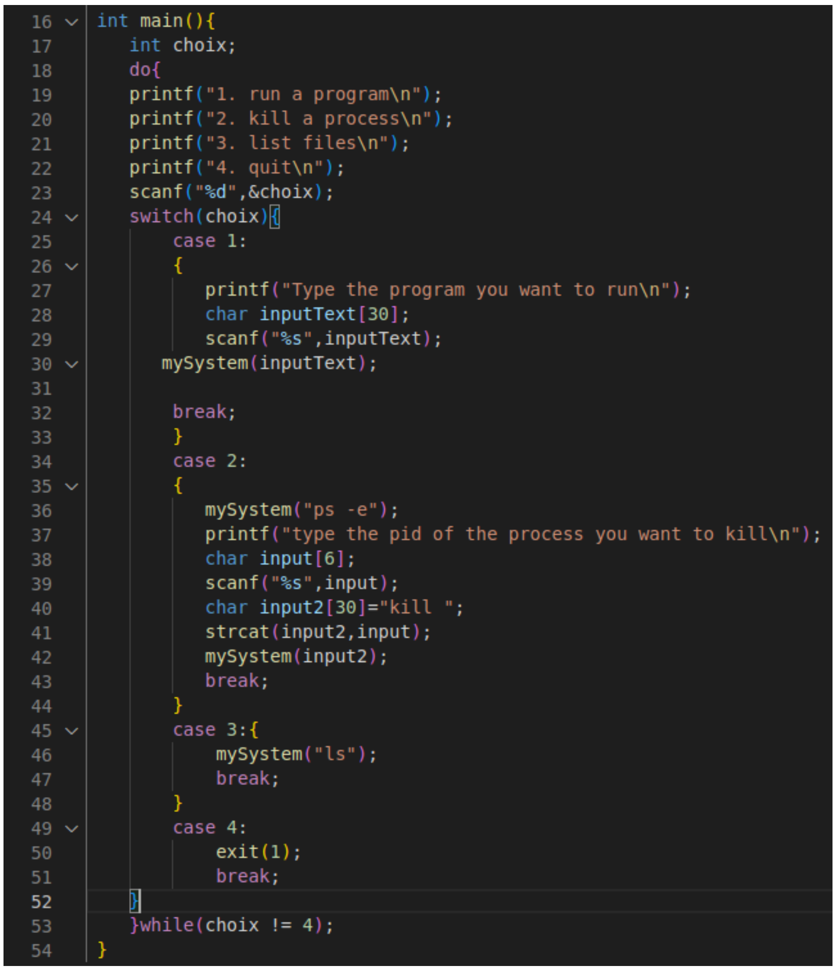
2. We implement our own system function which takes a command as a parameter.



```c
void mySystem(const char * command){
    if (fork() == 0)// we enter in the child process because fork return 0
       execl("/bin/sh", "sh", "-c", command, (char *) NULL);
       else
       wait(NULL);


   //we get out of the child process and operate in the parent process
}
```

We use the *fork()* function to create a child process. In the child process, we use the *execl()* function which takes our bash command in parameter and executes it. The parent process waits for the child process to end using the *wait()* function.

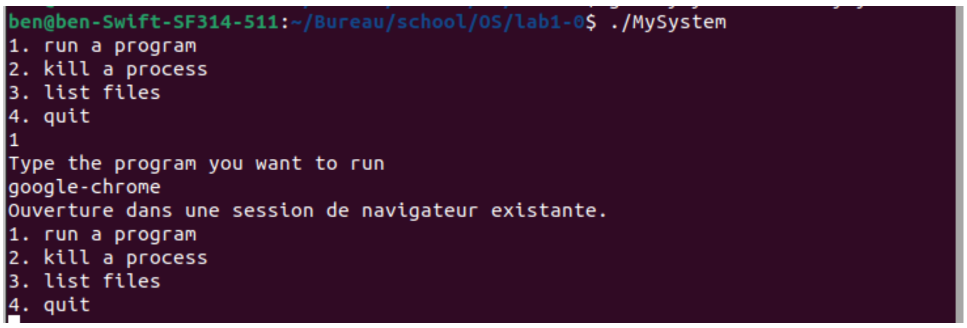
3.



```c
int main(){
   int choix;
   do{
   printf("1. run a program\n");
   printf("2. kill a process\n");
   printf("3. list files\n");
   printf("4. quit\n");
   scanf("%d",&choix);
   switch(choix){
       case 1:
       {
          printf("Type the program you want to run\n");
          char inputText[30];
          scanf("%s",inputText);
      mySystem(inputText);

       break;
       }
       case 2:
       {
          mySystem("ps -e");
          printf("type the pid of the process you want to kill\n");
          char input[6];
          scanf("%s",input);
          char input2[30]="kill ";
          strcat(input2,input);
          mySystem(input2);
          break;
       }
       case 3:{
           mySystem("ls");
           break;
       }
       case 4:
           exit(1);
           break;
   }
   }while(choix != 4);
}
```

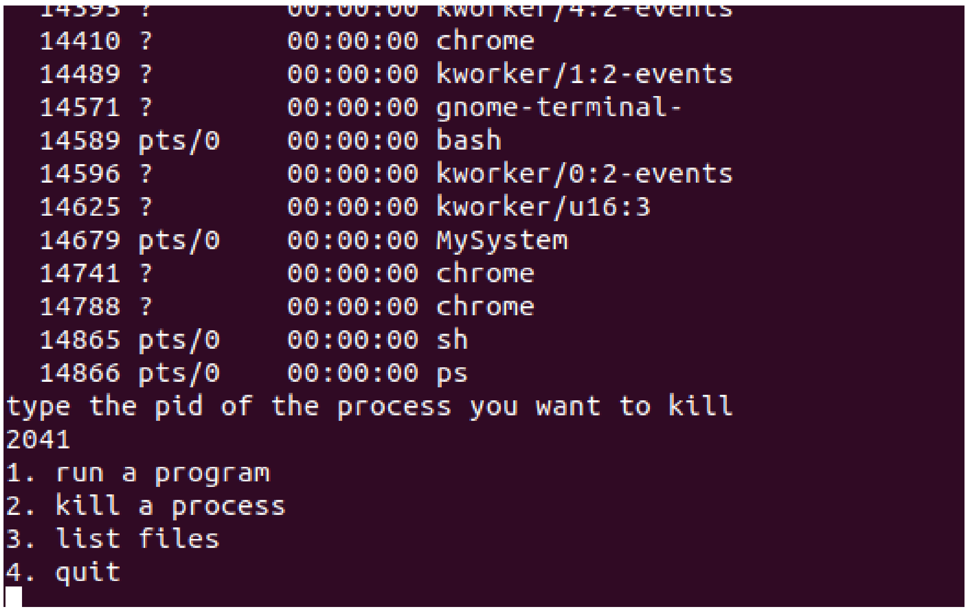
We use a do{}while(); loop to display the menu and the switch function to select wtah we want to perform.

In the first case, we type the name of a program we want to run and put it throught the *Mysystem()* function.



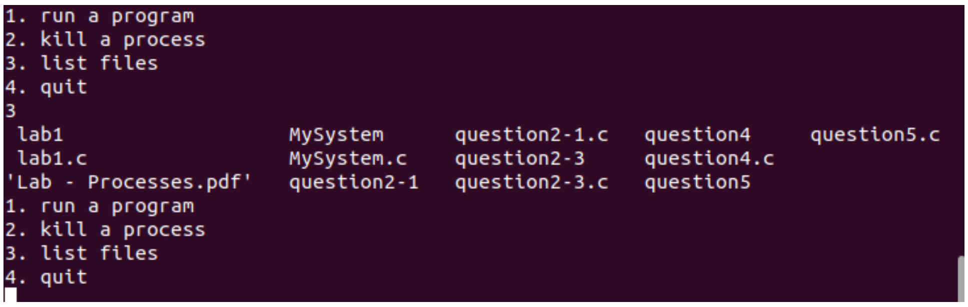
```
ben@ben-Swift-SF314-511:~/Bureau/school/OS/lab1-0$ ./MySystem
1. run a program
2. kill a process
3. list files
4. quit
1
Type the program you want to run
google-chrome
Ouverture dans une session de navigateur existante.
1. run a program
2. kill a process
3. list files
4. quit
```

In the seconde case, we first execute the bash command “ps –e” to print every process running. Then we ask the user to type the PID of the process he wants to kill. In order to kill a process, we put in parameters of *Mysystem()* function the command kill and the PID of the process.



```
  14393 ?        00:00:00 kworker/4:2-events
  14410 ?        00:00:00 chrome
  14489 ?        00:00:00 kworker/1:2-events
  14571 ?        00:00:00 gnome-terminal-
  14589 pts/0    00:00:00 bash
  14596 ?        00:00:00 kworker/0:2-events
  14625 ?        00:00:00 kworker/u16:3
  14679 pts/0    00:00:00 MySystem
  14741 ?        00:00:00 chrome
  14788 ?        00:00:00 chrome
  14865 pts/0    00:00:00 sh
  14866 pts/0    00:00:00 ps
type the pid of the process you want to kill
2041
1. run a program
2. kill a process
3. list files
4. quit
```

In the third case, we put the “ls” bash command throught Mysystem to show the files in the current folder.



```
1. run a program
2. kill a process
3. list files
4. quit
3
 lab1                    MySystem      question2-1.c   question4     question5.c
 lab1.c                  MySystem.c    question2-3     question4.c
'Lab - Processes.pdf'    question2-1   question2-3.c   question5
1. run a program
2. kill a process
3. list files
4. quit
```