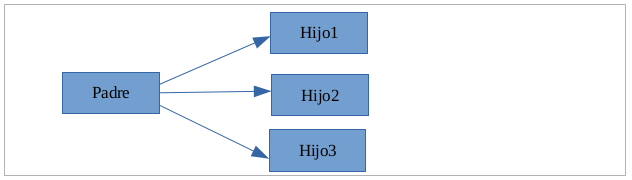
**SPECIFICATION**

**Activity 4**

Create a program in C that generates a process structure made up of a parent and 3 children. Display the PID for each child and for the parent. Also display the PID of the parent for all the children.



**Expected result:**

$./actividad4

Soy el hijo 3, Mi padre es 2186 y mi PID es 3077

Soy el hijo 2, Mi padre es 2186 y mi PID es 3076

Soy el hijo 1, Mi padre es 2186 y mi PID es 3075

Proceso padre 2186

Or:

$./actividad4

Soy el hijo 1, Mi padre es 2186 y mi PID es 3075

Soy el hijo 2, Mi padre es 2186 y mi PID es 3076

Soy el hijo 3, Mi padre es 2186 y mi PID es 3077

Proceso padre 2186

**Solution:**

Include here the code of the program actividad4.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

void main()

{

pid\_t pid,hijo\_pid,padre;

padre = getpid(); // I store the fathers pid to be able to print it later

for(int i=1;i<4;i++) //loop for making children

{

pid = fork();

if(pid == -1){

printf("Error al crear proceso");

}

if(pid == 0) //this is the children process

{

printf("Soy el hijo %d mi padre es %d y mi pid es %d\n",i,getppid(),getpid());

exit(0);

}

}

printf("Proceso padre %d",padre);

for(int i=1;i<4;i++) {

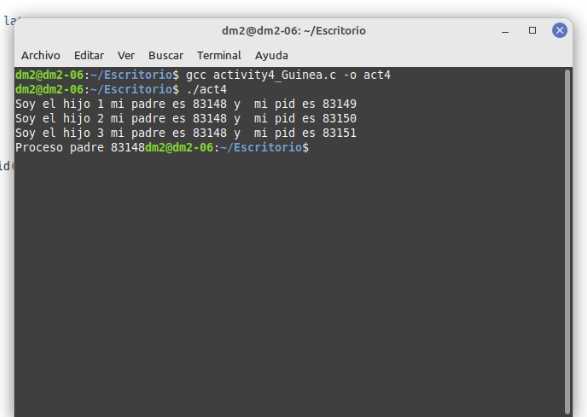
hijo\_pid = wait(NULL); //waiting for the children to finish

}

}

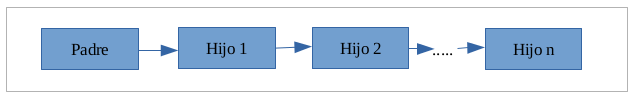
**Problem solving:**

following the structure of the examples, I struggled until I understood the need of the wait and exit functions.

****

**Activity 5**

Create a program in C that generates the following process structure:



**Solution:**

Include here the code of the program actividad5.c

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

#include <unistd.h>

#define N 4 //a constant defined to be changed as needed

void main()

{

printf("Proceso padre: PID %d \n",getpid()); //the parent

pid\_t pid;

for (int i = 1; i < N; i++)

{

int pid = fork();

if (pid == -1)

{

printf("Error al crear proceso hijo");

exit(-1);

}

if (pid == 0) //child process

{

printf("hijo %d PID = %d, padre PID = %d\n", i,getpid(), getppid());

}

else{

wait(NULL);

exit(0);

}

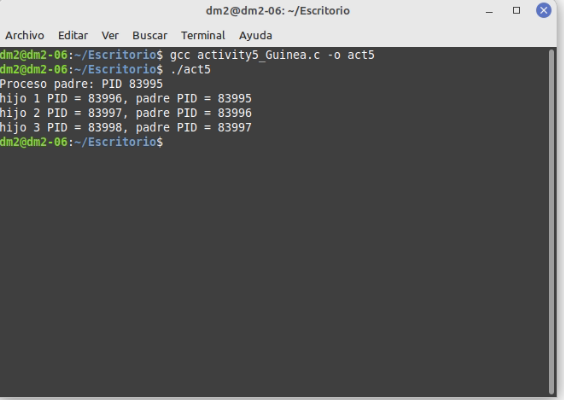
}

pid = wait(NULL);

}

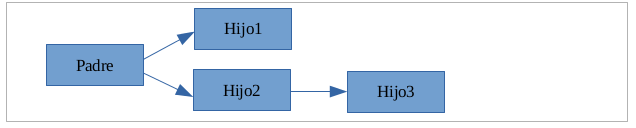
**Problem solving:**

For me the most complex one as at first I didn’t use the exit function and I was getting many prints

****

**Activity 6**

Create a program in C that generates the following process structure:



**Expected result:**

$./actividad6

Yo soy el hijo 2, mi padre es PID= 3410, yo soy PID= 3412  
Yo soy el hijo 1, mi padre es PID= 3410, yo soy PID= 3411  
Yo soy el hijo 3, mi padre es PID= 3412, yo soy PID= 3413

**Solution:**

Include here the code of the program actividad6.c

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/wait.h>

void main() {

pid\_t hijo1 = fork(); // fork a child

if(hijo1 == 0) //first child

{

printf("yo soy el hijo 1, mi padre es PID=%d, yo soy PID=%d\n",getppid(),getpid());

exit(1);

}

else //child and grandchild

{

pid\_t hijo2 = fork(); // fork a child

if (hijo2 == 0)

{

printf("yo soy el hijo 2, mi padre es PID=%d, yo soy PID=%d\n",getppid(),getpid());;

pid\_t grandchild = fork(); // fork a grandchild

if (grandchild == 0)

{

printf("yo soy el hijo 3, mi padre es PID=%d, yo soy PID=%d\n",getppid(),getpid());

exit(3);

}

else

{

grandchild = wait(NULL);

}

exit(2);

}

else

{

//wait for the 2 children to finish

hijo1 = wait(NULL);

hijo2 = wait(NULL);

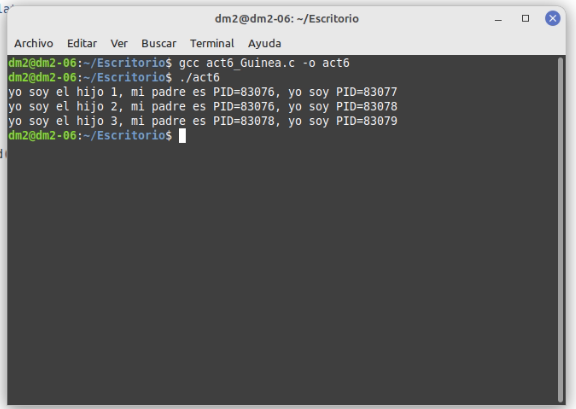
}

}

}

**Problem solving:**

after solving the previous one I could reuse code so it wasn’t as hard



**Activity 7**

Create a program in C to create a process (we will have a parent process and a child one). The program will define an integer variable and will assign it a value of 6. The parent process will increment this value by 5 and the child will subtract 5.

**Expected result:**

$./actividad7

Valor inicial de la variable: 6

Variable en Proceso Hijo: 1

Variable en Proceso Padre: 11

**Solution:**

Include here the code of the program actividad7.c

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

void main()

{

pid\_t pid;

int num = 6;

printf("Valor inicial de la variable %d\n",num);

pid = fork();

if(pid==-1){

printf("Error creating the child");

}

if (pid ==0){ //child

num = num -5;

printf("Variable en proceso hijo %d\n",num);

}

else{ //parent proccess

num = num +5;

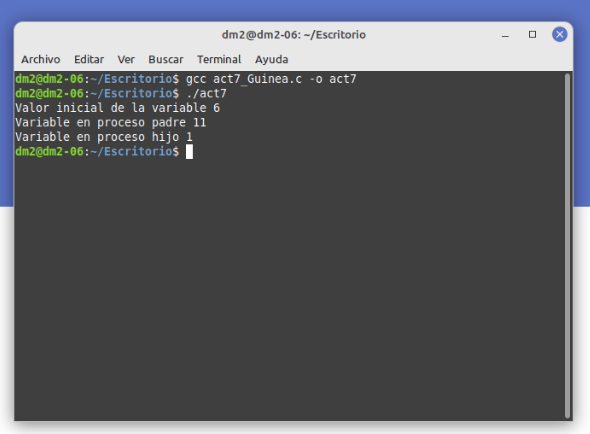
printf("Variable en proceso padre %d\n",num);

}

}

**Problem solving:**

for me the most intuitive one, helpful to understand how each process treats the variable in their own way



**REQUIREMENTS**

1. Deliver the solution of the activity in a single compressed file whose name is: the original name of the activity, followed by an underscore and the first surname of the student. The solution shall include the following:
   1. A file with the extension .odt or .docx that will contain
      1. A copy of the program code
      2. A screenshot of the result of running the program
   2. The source code file with extension .c

**ASSESSMENT CRITERIA**

1. Level of compliance with requirements (15%).
2. Correct functioning of programs developed in C (35%).
3. Alignment with expected results (35%).
4. Code clarity (15%).
5. Detailed explanation of how problems that may have arisen during the activity have been solved (up to an additional 15% on the mark obtained).
6. Warnings or compilation errors (-15%).
7. Level of comments in the source code (-15%).