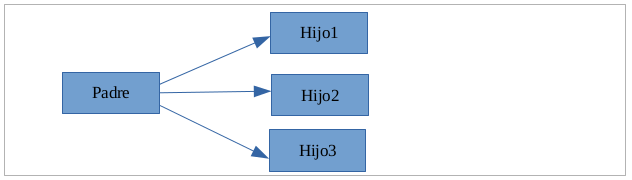
**SPECIFICATION**

**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 <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#include <sys/wait.h>

void main ()

{

pid\_t pid;

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

pid = fork();

if (pid == -1)

{

printf("No se ha podido crear el proceso hijo...");

exit (-1);

}

if (pid == 0) ////////// Nos encontramos en el proceso hijo

{

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

exit(0);

}

}

//Por cada hijo que se crea el padre tiene que esperar para que no salga el primero

wait(NULL);

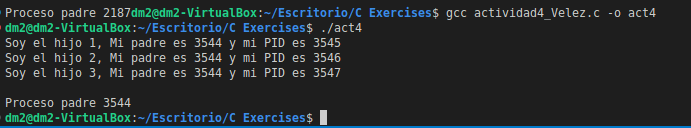
wait(NULL);

wait(NULL);

// No tienes que usar el getppid porque los que hablan antes son los hijos entonces el pid es el padre

printf ("\nProceso padre %d\n", getpid());

}



**Problem solving:**

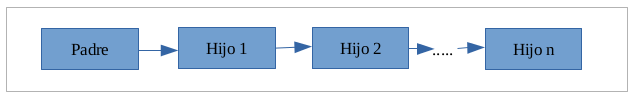
It didn’t do the loop properly because the parent never exit.

The loop never ends because the parent didn´t wait to the child and the child became a zombie process.

The parent PID didn’t go with the child PID parent because I was saving its value on a variable inside the loop and the PID was changing.

**Activity 5**

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



**Solution:**

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#include <sys/wait.h>

void main ()

{

pid\_t pid;

// El número de hijos que quieres que tenga el padre

int numChild = 4;

for(int i = 0; i<=numChild; i++){

pid = fork();

if (pid == -1)

{

printf("No se ha podido crear el proceso hijo...");

exit (-1);

}

if (pid != 0){

// Nos encontramos en el proceso padre

pid = wait(NULL);

//Avisar que ha terminado bien su ejecución

exit(0);

} else {

// Estamos en el proceso hijo

// En la primera vuelta dice que es el padre

if (i == 0){

printf("Soy el proceso padre mi PID es %d\n", getpid());

}else {

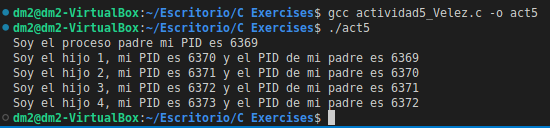
printf ("Soy el hijo %d, mi PID es %d y el PID de mi padre es %d\n", i, getpid(), getppid());

}

}

}

}



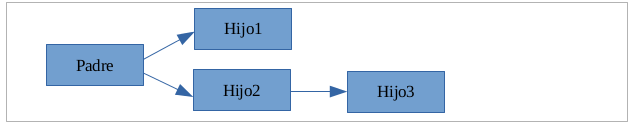
**Problem solving:**

The program execute itself twice, it didn’t loop because I haven’t said to the parent to exit and the if clause was incorrect defined.

It has a deadlock because the child wait for the parent and the parent for the child (only the parent has to wait and exit(0) in this program).

**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 <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#include <sys/wait.h>

void main ()

{

pid\_t pid, pid2;

for(int i = 1; i<3; i++){

pid = fork();

if (pid == -1)

{

printf("No se ha podido crear el proceso hijo...");

exit (-1);

}

if (pid == 0) // Nos encontramos en el proceso hijo

{

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

if (i == 2){

// Si es el hijo 2 crea un nuevo subproceso

pid2 = fork();

switch (pid2){

case -1:

printf("No se ha podido crear el proceso 3...");

exit (-1);

break;

case 0:

// Proceso hijo del proceso 2

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

break;

default: // Proceso hijo que a la vez es padre

wait (NULL);

}

}

exit(0);

}

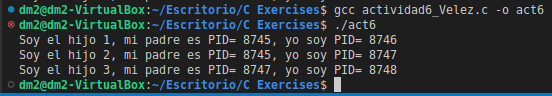
}

//Por cada hijo que se crea el padre tiene que esperar

wait(NULL);

wait(NULL);

}



**Problem solving:**

The PID of both children are the same because I have store their PID on the same variable without a loop.

The second child speak twice, in this one I only had to remove the print on the default clause.

**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 <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#include <sys/wait.h>

void main ()

{

pid\_t pid;

int num = 6;

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

pid = fork();

if (pid == -1)

{

printf("No se ha podido crear el proceso hijo...");

exit (-1);

}

if (pid == 0) // Nos encontramos en el proceso hijo

{

printf ("Variable en proceso hijo: %d\n",num-5);

}else{

// Nos encontramos en el proceso padre

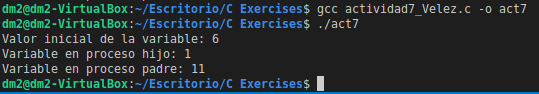
wait(NULL);

printf ("Variable en proceso padre: %d\n",num+5);

exit(0);

}

}



**Problem solving:**

Surprisingly in this exercise i didn’t have any problem.

**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%).