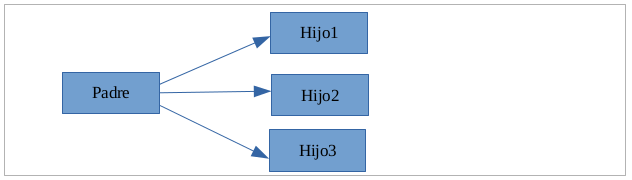
**ENUNCIADO**

**Actividad 4**

**✓**

Haz un programa C que genere una estructura de procesos con un padre y 3 hijos. Visualiza por cada hijo su PID y el del padre. Visualiza también el PID del padre de todos.



**Resultado esperado:**

$./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

O bien:

$./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

**Solución:**

Incluir aquí el código del programa Actividad4.

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

#include <sys/wait.h>

int main() {

// PID del proceso padre

pid\_t padre = getpid();

// Crear 3 procesos hijos

for (int i = 0; i < 3; i++) {

pid\_t hijo\_pid = fork();

if (hijo\_pid == 0) {

// Este código se ejecutará en cada proceso hijo

pid\_t pid = getpid();

printf("Soy el hijo %d, Mi padre es %d y mi PID es %d\n", i+1,padre,pid);

// Salir del proceso hijo

return 0;

}

}

// El proceso padre espera a que todos los hijos terminen

for (int i = 0; i < 3; i++) {

wait(NULL);

}

// Informacion sobre el proceso padre

printf("Soy el proceso padre. Mi PID es: %d\n", padre);

return 0;

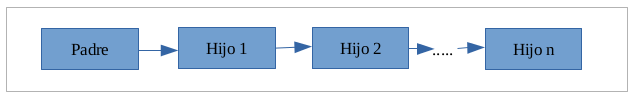
}

**Resolución de problemas:**

**Actividad 5**

**🗶**

Haz un programa C que genere la siguiente estructura de procesos:



**Solución:**

Incluir aquí el código del programa Actividad5.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main() {

// Obtener el PID del proceso padre

pid\_t parent\_pid = getpid();

printf("Soy el proceso padre %d.\n", parent\_pid);

// Crear el primer hijo

pid\_t child1\_pid = fork();

if (child1\_pid == 0) {

pid\_t child1\_pid\_actual = getpid();

printf("Soy el hijo %d y mi padre es %d.\n", child1\_pid\_actual, parent\_pid);

// Crear el segundo hijo

pid\_t child2\_pid = fork();

if (child2\_pid == 0) {

pid\_t child2\_pid\_actual = getpid();

printf("Soy el hijo %d y mi padre es %d.\n", child2\_pid\_actual, child1\_pid\_actual);

exit(EXIT\_SUCCESS);

}

exit(EXIT\_SUCCESS);

}

return 0;

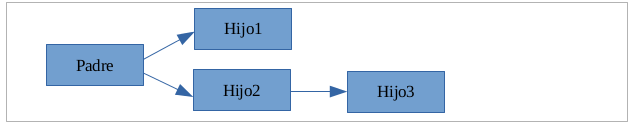
}

**Resolución de problemas:**

**Actividad 6**

**✓**

Haz un programa C que genere la siguiente estructura de procesos:



Resultado esperado:

$./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

**Solución:**

Incluir aquí el código del programa Actividad6.c

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

int main() {

pid\_t child1, child2, child3;

// PID del padre

pid\_t parent\_pid = getpid();

// Crear el primer hijo

child1 = fork();

if (child1 == 0) {

// Código del primer hijo

printf("Soy el primer hijo, mi PID es %d, mi padre es %d\n", getpid(), parent\_pid);

} else if (child1 > 0) {

// Crear el hijo2

child2 = fork();

if (child2 == 0) {

pid\_t hijo2 = getpid();

printf("Soy el segundo hijo, mi PID es %d, mi padre es %d\n", hijo2, parent\_pid);

// Crear el hijo3 y que tenga como padre el hijo2

child3 = fork();

if (child3 == 0) {

printf("Soy el tercer hijo, mi PID es %d, mi padre es %d\n", getpid(), hijo2);

}

}

}

return 0;

}

**Resolución de problemas:**

**Actividad 7**

**✓**

Realiza un programa en C que cree un proceso (tendremos un proceso padre y otro hijo). El programa definirá una variable entera y le dará un valor 6. El proceso padre incrementará dicho valor en 5 y el hijo restará 5.

Resultado esperado:

$./actividad7

Valor inicial de la variable: 6

Variable en Proceso Hijo: 1

Variable en Proceso Padre: 11

**Solución:**

Incluir aquí el código del programa Actividad7.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main() {

// Valor inicial

int valor = 6;

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

// Crear un nuevo proceso

pid\_t child\_pid = fork();

if (child\_pid == -1) {

perror("Error al crear el proceso hijo");

exit(EXIT\_FAILURE);

} else if (child\_pid == 0) {

// Restar 5 al valor

valor -= 5;

printf("Variable en proceso Hijo: %d\n", valor);

// Salir del proceso hijo

exit(EXIT\_SUCCESS);

} else {

// Incrementar 5 al valor

valor += 5;

printf("Variable en proceso Padre: %d\n", valor);

// Esperar a que el proceso hijo termine

wait(NULL);

}

return 0;

}

**Resolución de problemas:**

**CRITERIOS DE CORRECCIÓN**

1. Nivel de ajuste a la nomenclatura de objetos definida en el enunciado de la actividad y nivel de ajuste al almacenamiento de la información requerida en las subcarpetas definidas para la actividad y unidad didáctica correspondientes (15%).
2. Correcto funcionamiento de los programas desarrollados en C (35%).
3. Ajuste a los resultados esperados (35%).
4. Claridad del código C (15%).
5. Explicación detallada de cómo se han resuelto problemas que hayan podido surgir en la realización de la actividad (hasta un 15% adicional sobre la nota obtenida).
6. Avisos o errores de compilación (-15%).
7. Nivel de comentarios en el código fuente (-15%).