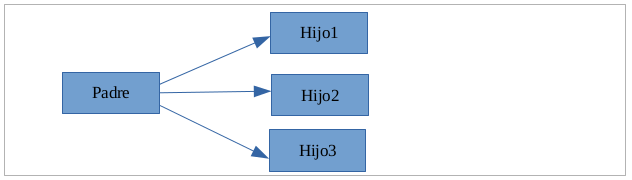
**ENUNCIADO**

**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.c

**Resolución de problemas:**

**#include <stdio.h>**

#include <stdlib.h>

#include <unistd.h>

#include <sched.h>

#include <time.h>

int main() {

pid\_t pid;

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

pid = fork();

if (pid < 0) {

// Error al crear el hijo

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

exit(1);

} else if (pid == 0) {

// Estamos en el proceso hijo

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

exit(0); // Terminamos el proceso hijo

}

// El proceso padre sigue ejecutando el ciclo para crear más hijos

}

// Estamos en el proceso padre

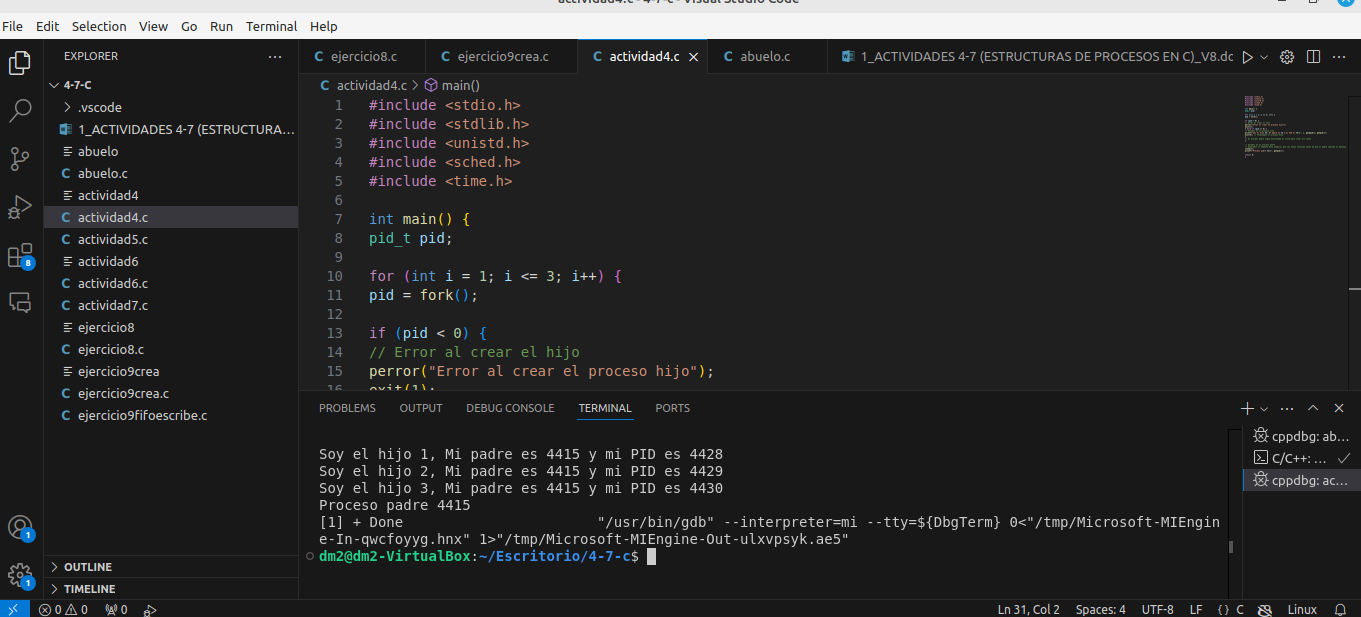
// Esperamos un momento para asegurar que los hijos terminen antes de que el padre imprima su mensaje

sleep(1);

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

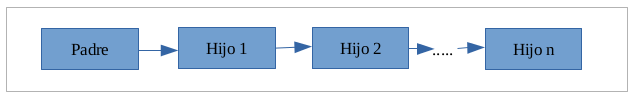
return 0;

}

****

**Actividad 5**

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



**Solución:**

Incluir aquí el código del programa Actividad5.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sched.h>

#include <time.h>

int main() {

int n;

pid\_t pid;

printf("Ingrese el número de hijos para crear: ");

scanf("%d", &n);

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

pid = fork ();

if (pid < 0) {

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

exit (-1);

}

if (pid == 0) {

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

}

else { // El proceso padre

wait(NULL); // El padre espera a que termine su hijo

break; // El padre sale del ciclo para no crear más hijos

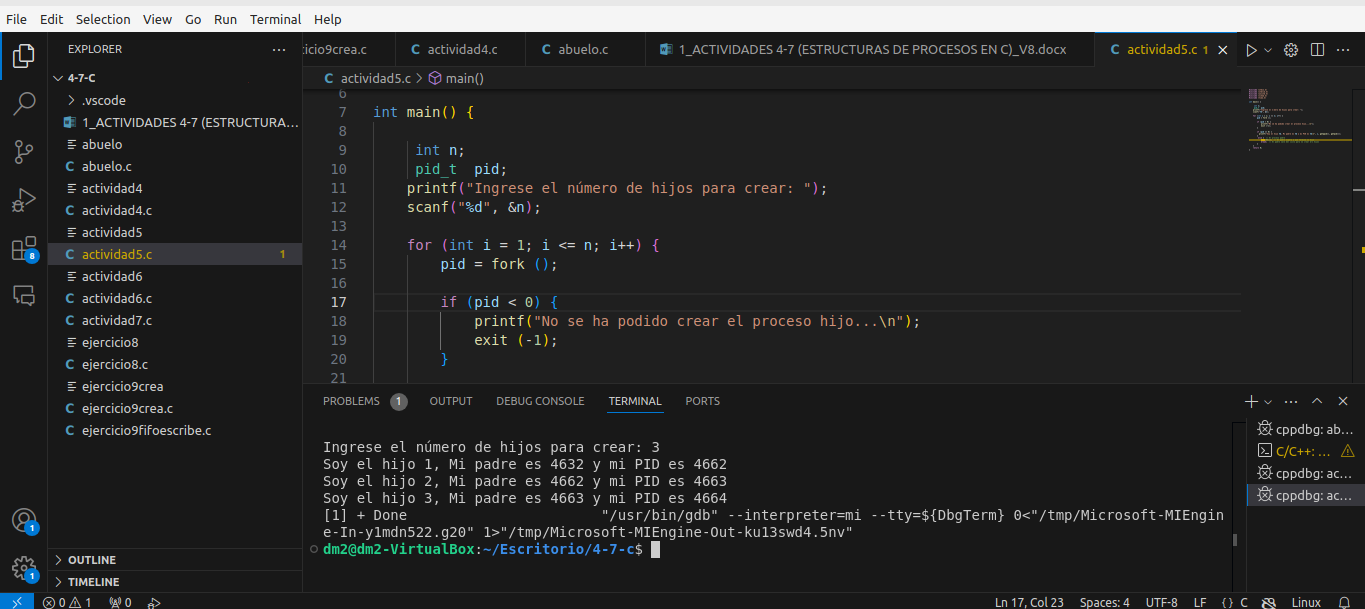
}

}

return 0;

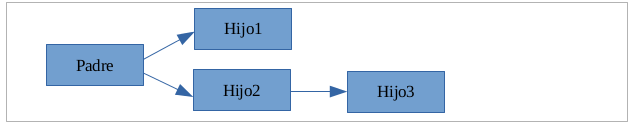
}

**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

**Resolución de problemas:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

int main() {

pid\_t pid;

// Crear el primer hijo

if ((pid = fork()) == 0) {

// Hijo 1

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

exit(0);

}

// Crear el segundo hijo

if ((pid = fork()) == 0) {

// Hijo 2

printf("Soy el hijo 2, Mi padre es %d y mi PID es %d\n", getppid(), getpid());

// Crear el hijo 3 dentro del hijo 2

if ((pid = fork()) == 0) {

// Hijo 3

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

exit(0);

}

// Hijo 2 espera al hijo 3

wait(NULL);

exit(0);

}

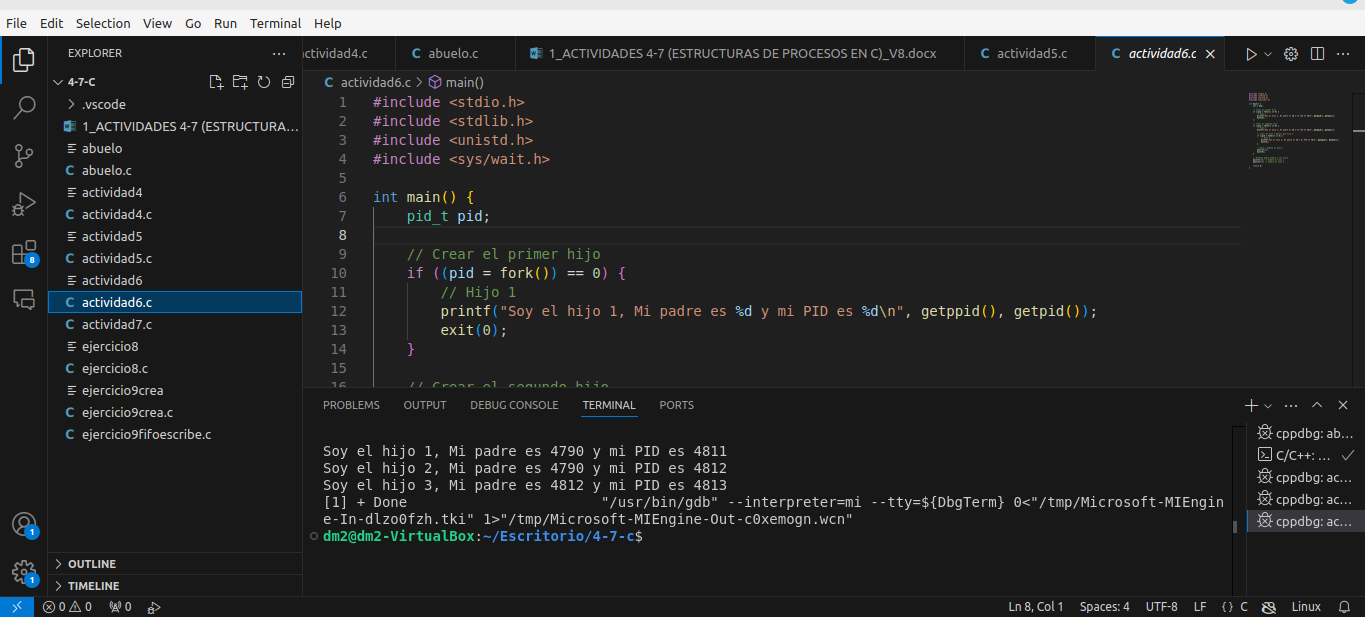
// Proceso padre espera a los hijos

wait(NULL); // Espera al hijo 1

wait(NULL); // Espera al hijo 2

return 0;

}

****

**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

**Resolución de problemas:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

int main() {

pid\_t pid;

int variable = 6; // Variable inicial

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

// Crear un proceso hijo

pid = fork();

if (pid < 0) {

// Manejo de errores

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

exit(1);

}

if (pid == 0) {

// Estamos en el proceso hijo

variable -= 5; // Restar 5 en el proceso hijo

printf("Variable en Proceso Hijo: %d\n", variable);

exit(0); // Terminar el proceso hijo

} else {

// Estamos en el proceso padre

wait(NULL); // Esperar a que el hijo termine

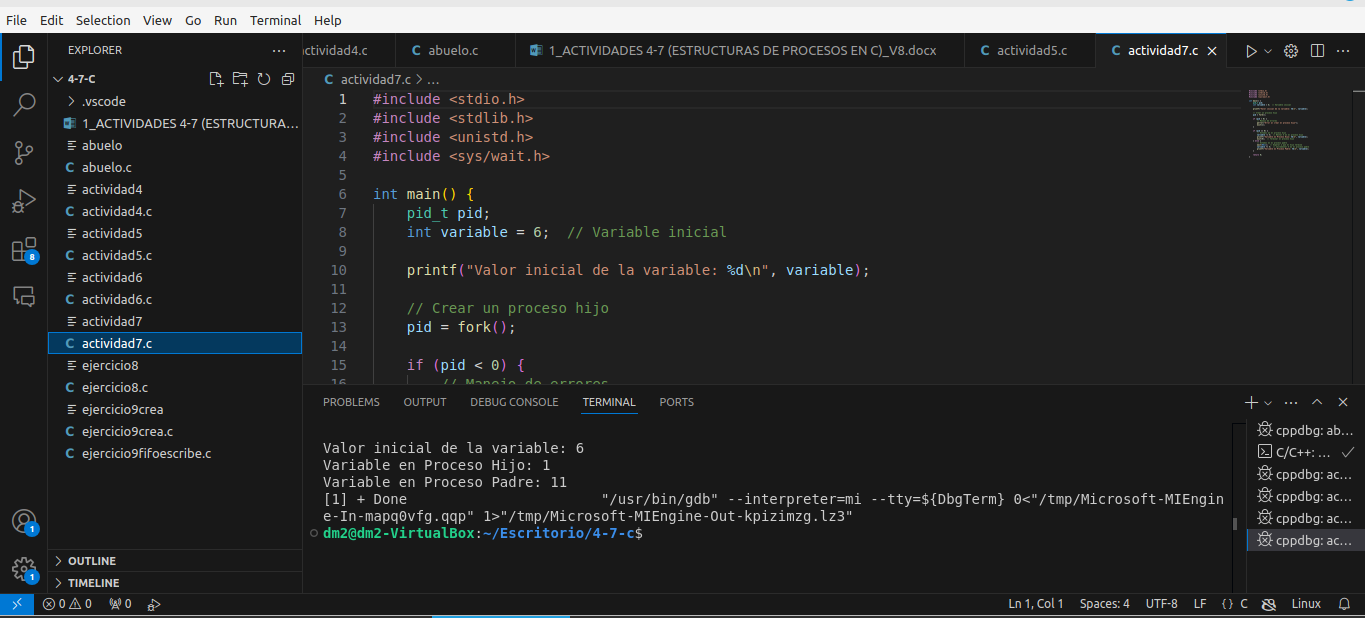
variable += 5; // Incrementar 5 en el proceso padre

printf("Variable en Proceso Padre: %d\n", variable);

}

return 0;

}

****

**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%).