Instituto Politécnico Nacional

ESCOM

Practica 3

**Administrador de procesos en Linux y Windows**

**Sistemas Operativos – 2CM17**

**Integrantes:**

Mora Ayala José Antonio

Ramírez Cotonieto Luis Fernando

Torres Carrillo Josehf Miguel Angel

Tovar Jacuinde Rodrigo

**Profesor:**

Cortes Galicia Jorge

# Introducción

A continuación, se presenta una introducción a los temas que serán profundizados dentro de esta práctica mediante la implementación de código funcional relacionado a cada uno de los temas que competen este documento, así como a cada uno de los integrantes del equipo

Como todos sabemos, conforme ha pasado el tiempo las generaciones van cambiando la manera en que experimentan la evolución tecnológica que se ha dado en el mundo desde hace varios años y la cual, hoy continua, pues la tecnología cada día avanza y se van realizando nuevos descubrimientos a diario. Situación que resulta benéfica la mayoría del tiempo, pero también puede llegar a ser contraproducente, pues conforme más acceso y facilidad se tiene a estas tecnologías y lo que conlleva interactuar con las mismas las personas en general han perdido el interés en conocer cómo es que funcionan o que partes son las que componen los dispositivos con los cuales interactúan día a día.  
Es por eso que comenzaremos dando un antecedente y conceptos generales y básicos para comprender de una forma más completa esta practica

## ¿Qué es un proceso?

Cada programa que se ejecuta es un proceso con recursos asignados y gestionado por el kernel. La ™ gestión de procesos comprende la monitorización, detención y cambio de prioridad de los procesos. Generalmente los procesos son gestionados automáticamente por el kernel del S.O. (son creados, ejecutados y detenidos sin la intervención del usuario). Algunas veces los procesos se detendrán por razones desconocidas y será necesario reiniciar el proceso. Otras veces algún proceso se ejecutará descontroladamente malgastando los recursos del sistema, entonces será necesaria una intervención manual del administrador para detener el proceso.

## Párametros de un proceso:

PROCESS ID (PID): Cada proceso tiene un número asociado que se le asigna cuando es creado. Los PIDs son números enteros únicos para todos los procesos sistema. USER ID & GROUP ID: Cada proceso tiene que tener asociado unos privilegios que limiten el acceso al sistema de ficheros. Estos privilegios quedan determinados por el user ID y group ID del usuario que creo el proceso. PARENT PROCESS: Todo proceso es creado por otro proceso, el proceso padre (parent process). El primer proceso iniciado por el kernel cuando el sistema arranca es el programa init. Este proceso tiene el PID 1 y es el padre de todos los procesos del sistema. PARENT PROCESS ID: El PID del proceso que inicio el proceso hijo. ENVIROMENT: Cada proceso mantiene una lista de variables y sus correspondientes valores. El conjunto de estas variables recibe el nombre de process enviroment. Normalmente el entorno de un proceso hijo se hereda del proceso padre a menos de que se indique de otra forma. CURRENT WORKING DIRECTORY: Cada proceso tiene asociado un directorio por defecto, donde el proceso leerá/escribirá archivos, a menos que se le especifique explícitamente lo contrario. NICE NUMBER: Permite al usuario modificar la prioridad de ejecución de un proceso.

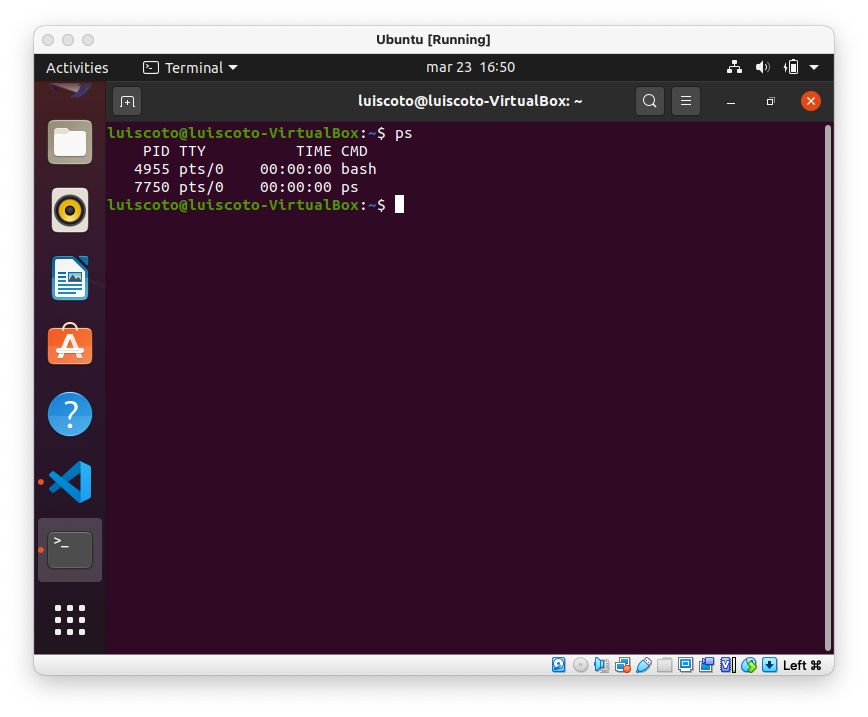
## Comandos fundamentales

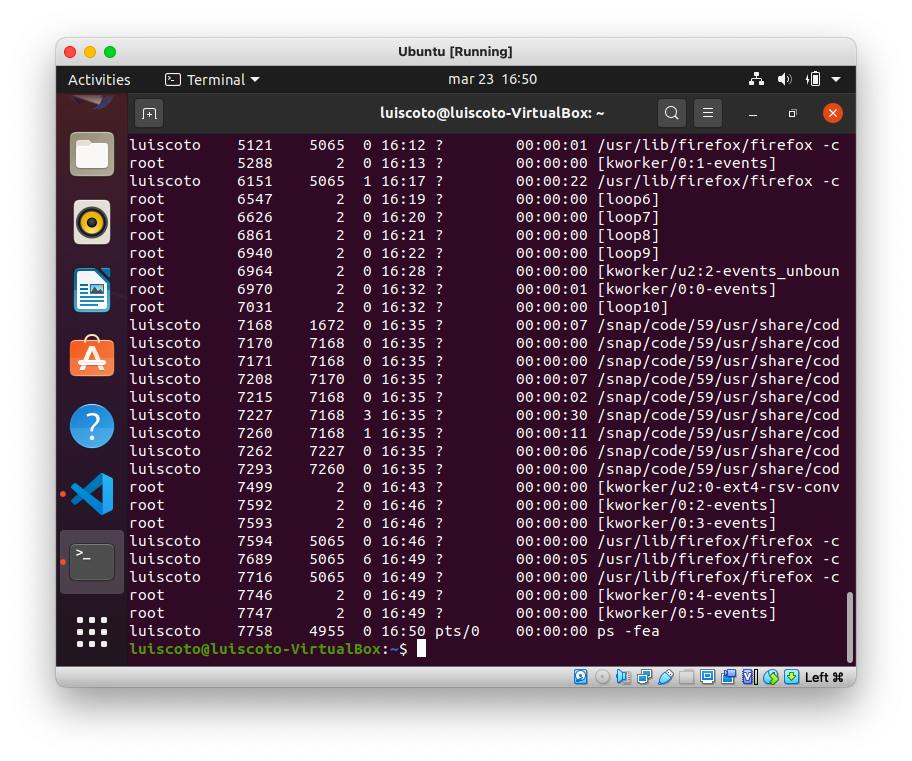
* ps: El comando PS es el que permite informar sobre el estado de los procesos. PS esta basado en el sistema de archivos /proc, es decir, lee directamente la información de los archivos que se encuentran en este directorio. Tiene una gran cantidad de opciones, incluso estas opciones varían dependiendo del estilo en que se use el comando
* pstree: El comando pstree muestra los procesos en forma de árbol.
* kill: El comando kill, que literalmente quiere decir matar, sirve no solo para matar o terminar procesos sino principalmente para enviar señales (signals) a los procesos. La señal por default (cuando no se indica ninguna es terminar o matar el proceso), y la sintaxis es kill PID, siendo PID el número de ID del proceso. Asi por ejemplo, es posible enviar una señal de STOP al proceso y se detendrá su ejecución, después cuando se quiera mandar una señal de Continuar y el proceso continuara desde donde se quedo.
* top: Muestra en tiempo real el estado de los procesos del sistema
* htop: Programa que administra todos los procesos activos en tiempo real

# Desarrollo Experimental

Sección Linux:

1. Introduzca los siguientes comandos a través de la consola del sistema operativo  
   Linux:

Ps

ps –fea

**¿Qué información le proporcionan los comandos anteriores?**

Estos dos comandos nos muestran la información de procesos en ejecución del sistema operativo, se nos muestran datos como el id, el id del proceso padre y cuando colocamos el ps-fea se nos muestra la información de esto de una manera mucho mas detallada

1. A través de la ayuda en línea que proporciona Linux, investigue para que se utiliza el comando ps y mencione las opciones que se pueden utilizar con dicho comando.  
   Además investigue el uso de las llamadas al sistema fork(), execv(), getpid(), getppid() y wait() en la ayuda en línea, mencione que otras funciones similares a execv() existen, reporte sus observaciones.
2. **fork():** jd

Fork crea un nuevo proceso duplicando la llamada del proceso, el nuevo proceso es llamado proceso hijo, el proceso que lo llamo es nombrado el proceso padre

1. **execv():** jd

es un comando que funciona para ejecutar un programa externo mediante los parámetros de un char denominado argv[] el cual es un puntero

1. **getpid():** jd

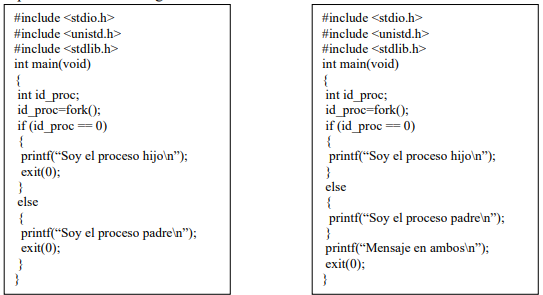
Nos devuelve el identificador de proceso actual, eso es usado normalmente por rutinas que generan nombres únicos de ficheros temporales

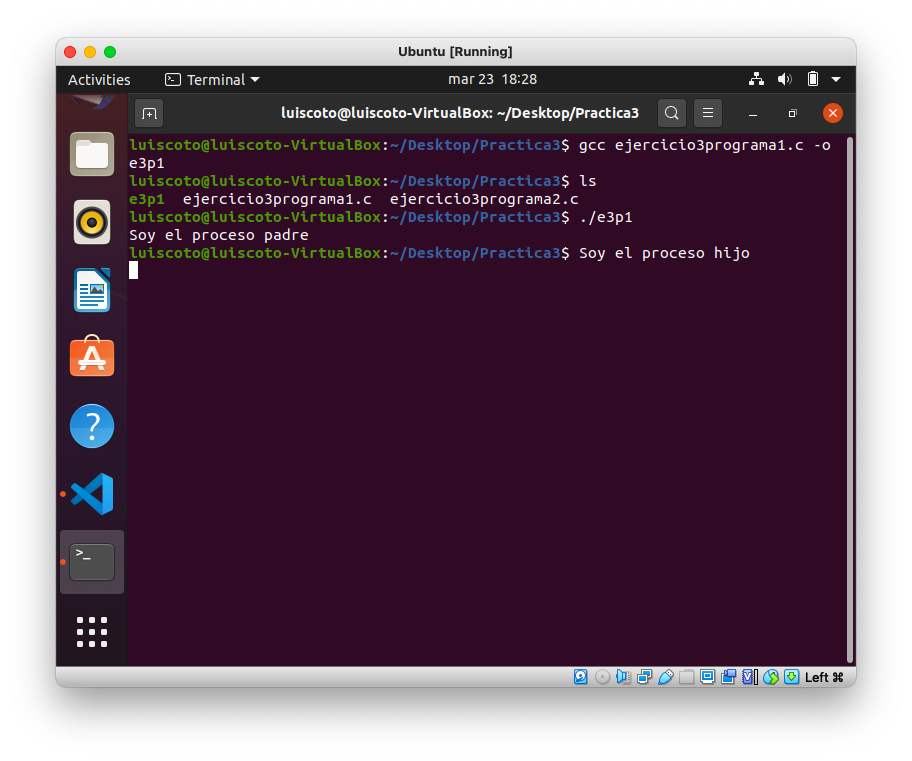
1. **getppid():** jd

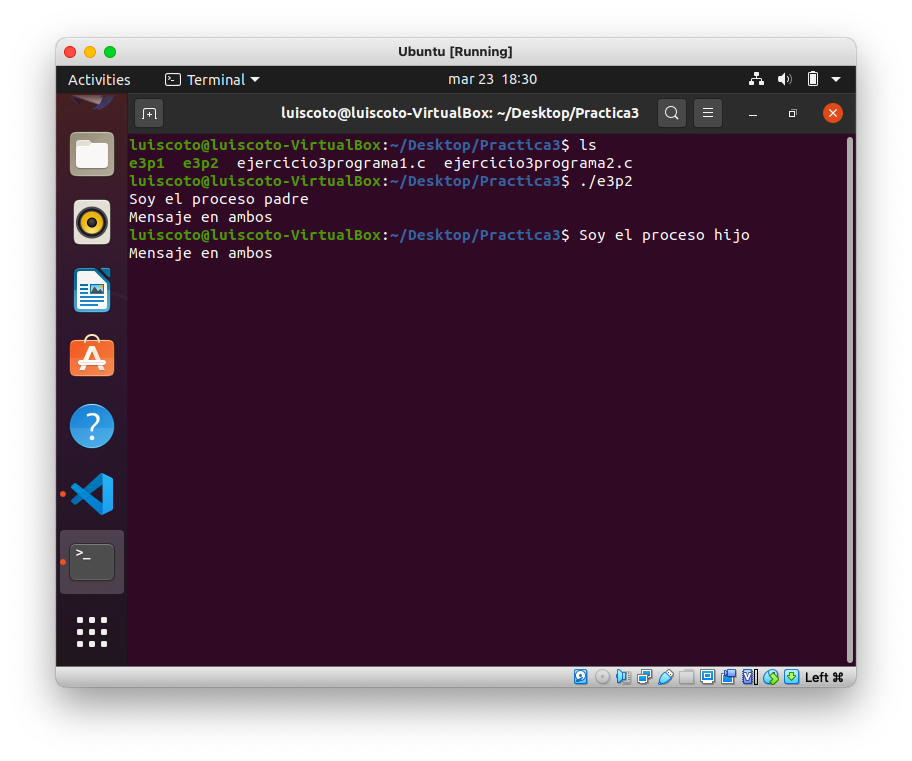
Nos devuelve el identificador del proceso del padre del proceso actual

1. **wait():** jd

la función hace que el proceso espere por la finalización de cualquier hijo y retorna el PID del proceso hijo

1. Capture, compile y ejecute los dos programas de creación de un nuevo proceso por copia exacta de código que a continuación se muestra. Observe su funcionamiento y experimente con el código.

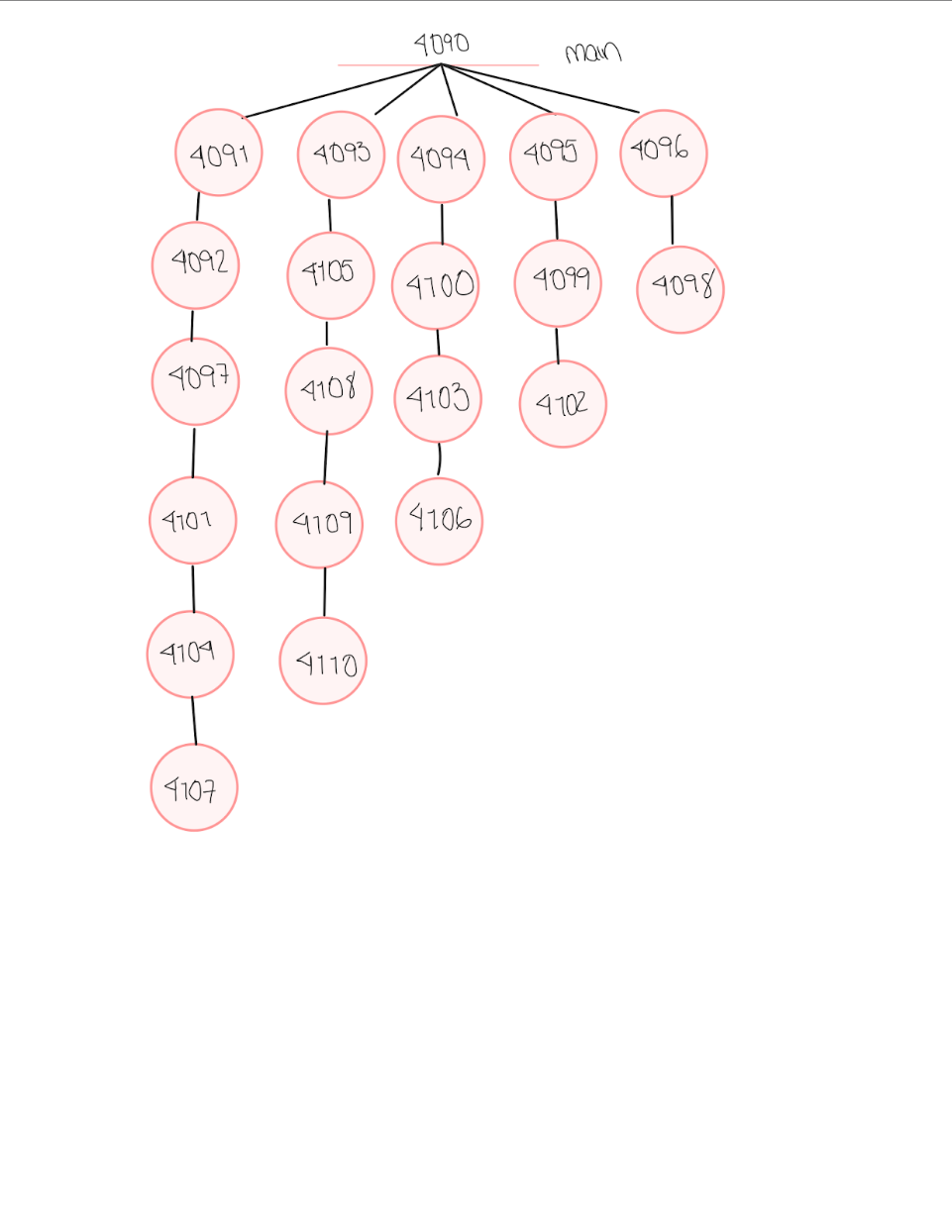
**Programa 1**

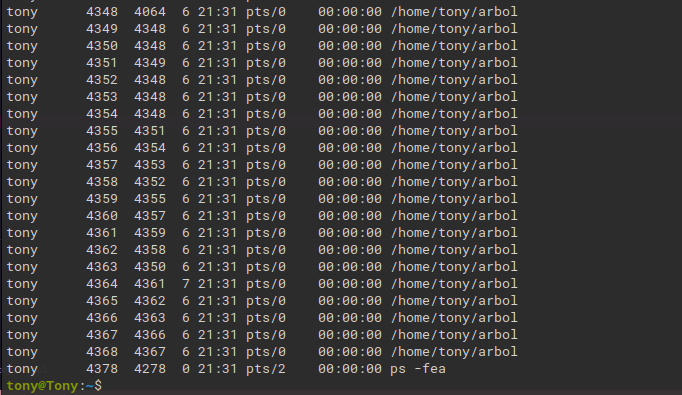
**Programa 2**

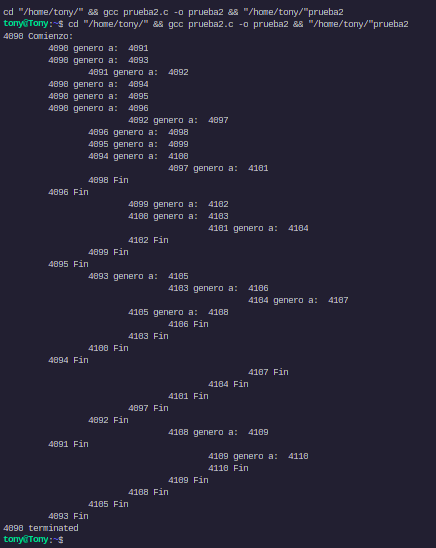
1. Programe una aplicación que cree el árbol de procesos mostrado al final de este

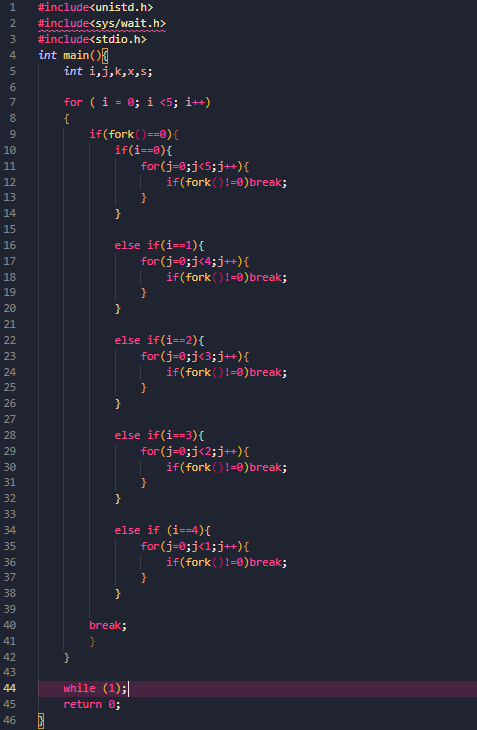
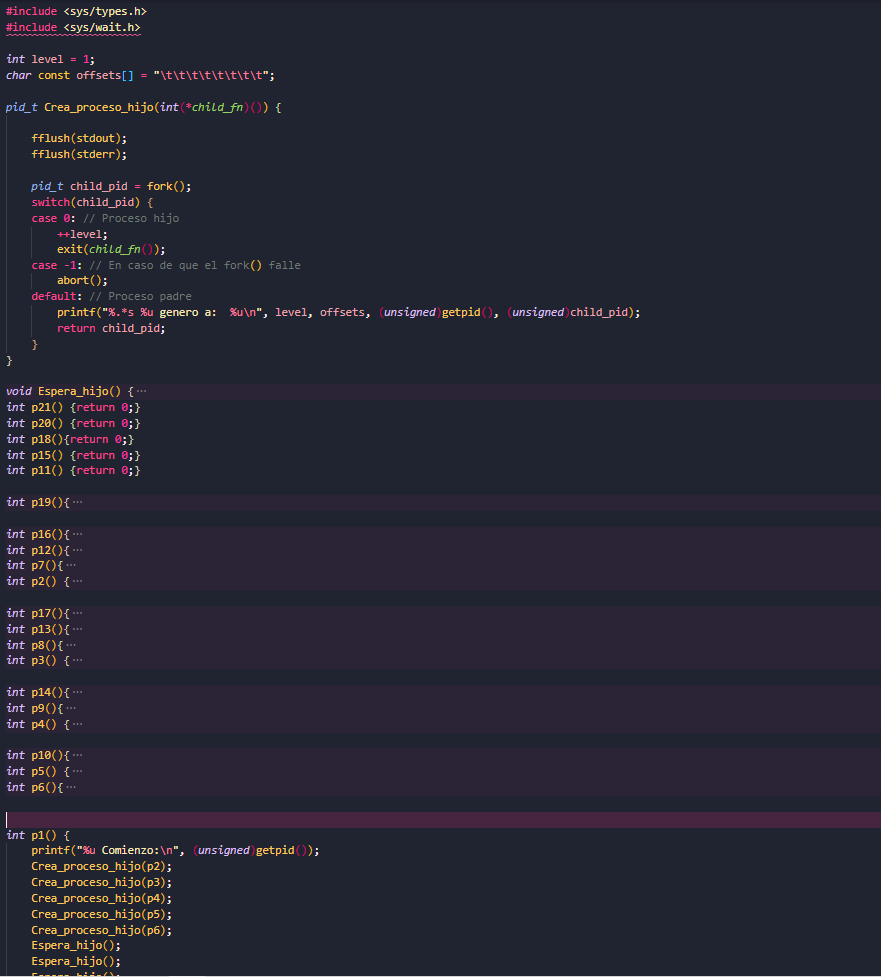
documento. Para cada uno de los procesos hijos creados (por copia exacta de código) se imprimirá en pantalla el pid de su padre, además se imprimirán en

pantalla los pids de los hijos creados de cada proceso padre. Dibuje en papel el árbol creado usando los pid’s que imprima en pantalla.









1. Programe una aplicación que cree cinco procesos (por copia exacta de código). El

primer proceso se encargará de realizar la suma de dos matrices de 7x7 elementos

tipo entero, el segundo proceso realizará la resta sobre esas mismas matrices, el

tercer proceso realizará la multiplicación de las matrices, el cuarto proceso obtendrá las transpuestas de cada matriz. Cada uno de estos procesos escribirá un archivo con los resultados de la operación que realizó. El quinto proceso leerá los archivos de resultados y los mostrará en pantalla cada uno de ellos.

Programe la misma aplicación sin la creación de procesos, es decir de forma

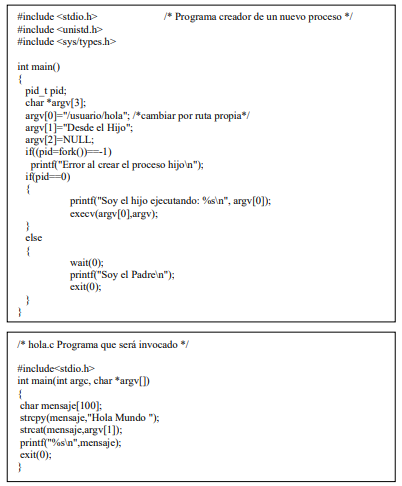
secuencial. Obtenga los tiempos de ejecución de las aplicaciones, compare estos

tiempos y dé sus observaciones.

1. Capture, compile y ejecute el siguiente programa de creación de un nuevo proceso

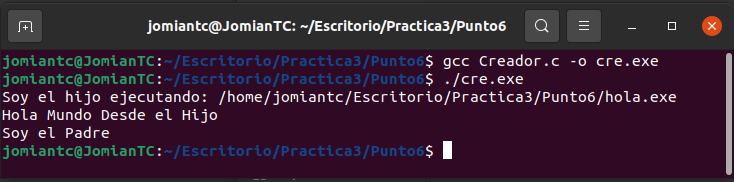
con sustitución de un nuevo código, así como el programa que será el nuevo código

a ejecutar. Observe su funcionamiento y experimente con el código.



Obtenga el ejecutable hola del programa hola.c antes de ejecutar el programa

creador de un nuevo proceso.

Ejecución de programa

1. Programe una aplicación que cree un proceso hijo a partir de un proceso padre, el

hijo creado a su vez creará tres procesos hijos más. Cada uno de los tres procesos generados ejecutará tres programas diferentes mediante sustitución de código, el

primer programa resolverá una ecuación algebraica de segundo grado mediante la

fórmula general, el segundo programa mostrará la serie de Fibonacci para un

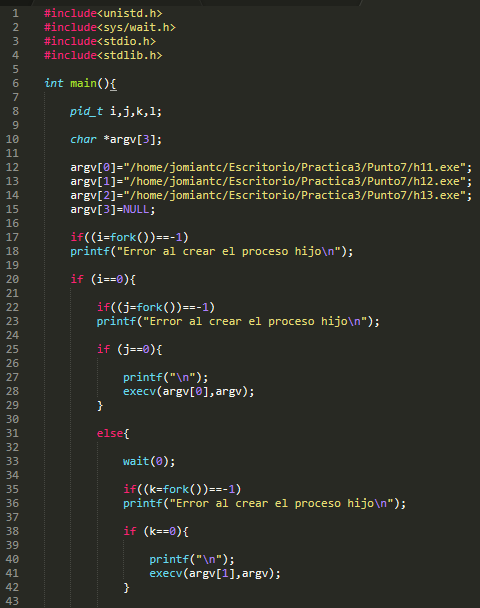
número N, y el tercer programa obtendrá la multiplicación de dos matrices de 7x7

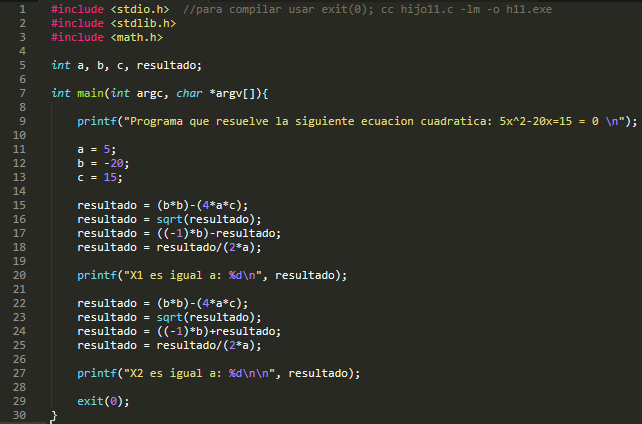
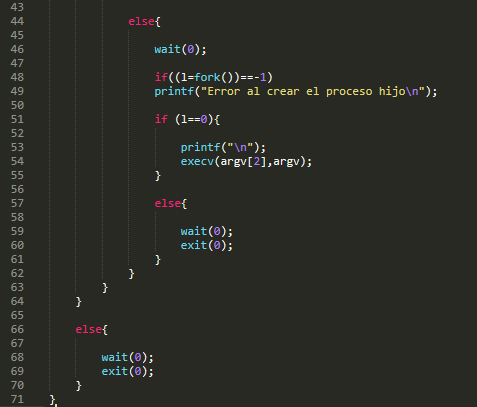
elementos de tipo entero. Observe el funcionamiento de su programa

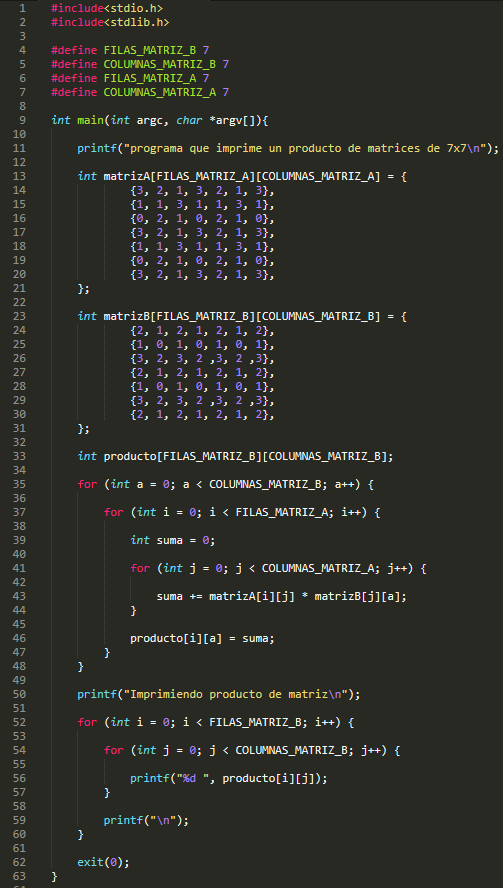
detalladamente y responda la siguiente pregunta ¿es posible un funcionamiento

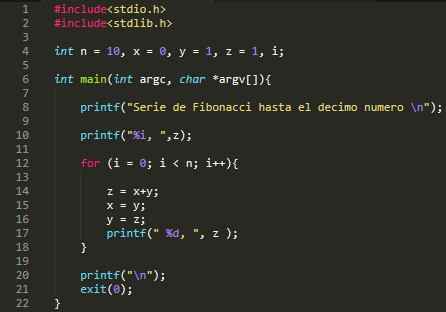
100% concurrente de su aplicación? Explique porque sí o no es 100% concurrente

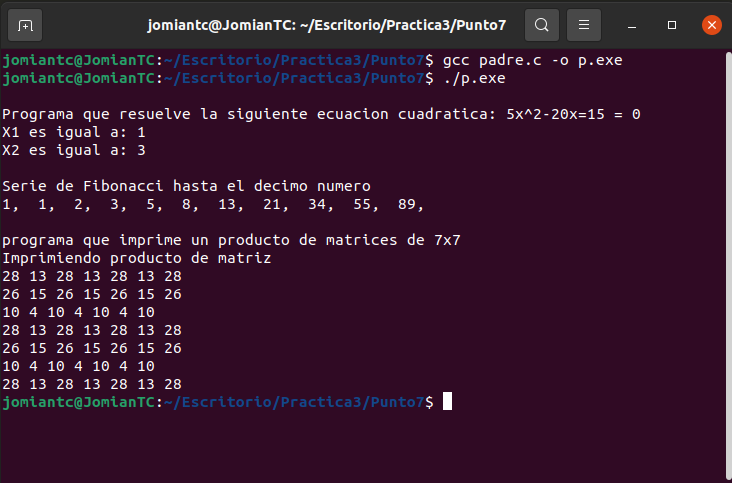
su aplicación.

**Programa Padre.c**

**Programa hijo11.c**

**Programa hijo13.c**

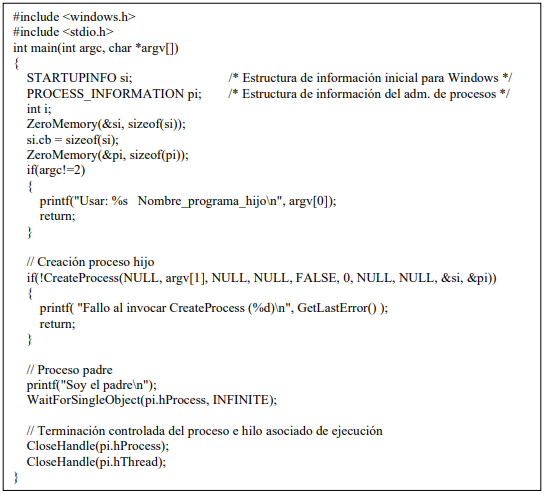
**Programa hijo12.c**

**Ejecución**

Es difícil decir si el programa es concurrente o no, de una manera si lo es ya que los procesos de los hijos 11, 12 y 13 se ejecutan 1 a 1 terminando primero uno y luego ejecutando el siguiente hasta terminar pero todo se lleva acabo mientras que el proceso hijo 1 aún se está ejecutando así que de cierta forma seria concurrente pero si ese proceso no existiera el programa no sería concurrente, aunque algunas opiniones afirman que si lo es ya que aún existe el proceso padre que ejecutaría a los procesos 11, 12 y 13 así que sería concurrente

1. Programe la aplicación desarrollada en el punto 5 de la sección de Linux utilizando esta vez la creación de procesos por sustitución de código.

**Sección Windows:**

1. **Inicie sesión en Windows.**
2. **Para esta práctica se utilizará el ambiente de programación Dev C/C++.**
3. **Capture y compile el programa de creación de un nuevo proceso que a continuación se muestra**.
4. **Capture y compile el programa que contendrá al proceso hijo que a continuación se muestra.**
5. **Ejecute el primer código pasando como argumento el nombre del archivo ejecutable del segundo código capturado. Observe el funcionamiento del programa, reporte sus observaciones y experimente con el código.**

Realizando la Ejecución de Código tal como se indica se puede observar que da diversos warnings, pues en el proceso padre cada que se hace una evaluación o una verificación se da un return; el cual no devuelve nada, de igual forma se observa que al momento de realizar la sustitución del nombre del programa hijo en donde se indica este no es aceptado y no se crea dicho proceso, pues argv[0]; nos proporciona la siguiente línea de comando:

Usar: C:\Users\TONY AYALA\Google Drive\Escolar\College\CuartoSemestre\Sistemas\_Operativos\Tarea3\Programas\Puntos3\_4\_5\padre.exe

La cual llega hasta la línea ejecutable del programa padre es por eso que al código padre se le ha realizado esta modificación:

#include <windows.h>

#include <stdio.h>

*int* main(*int* *argc*, *char* \**argv*[])

{

*STARTUPINFO* si; /\* Estructura de información inicial para Windows \*/

*PROCESS\_INFORMATION* pi; /\* Estructura de información del adm. de procesos \*/

*int* i;

 ZeroMemory(&si, sizeof(si));

 si.cb = sizeof(si);

 ZeroMemory(&pi, sizeof(pi));

 if(*argc*!=2)

 {

 printf("Usar: %s\n", *argv*[0]);

 strcat(*argv*[0],"\\Ejecutable.exe");

 printf("Usar: %s\n", *argv*[0]);

 }

 // Creación proceso hijo

 if(!CreateProcess("C:\\Users\\TONY AYALA\\Google Drive\\Escolar\\College\\CuartoSemestre\\Sistemas\_Operativos\\Tarea3\\Programas\\Puntos3\_4\_5\\Ejecutable.exe", NULL, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))

 {

 printf( "Fallo al invocar CreateProcess (%d)\n", GetLastError() );

 }

 // Proceso padre

 printf("Soy el padre\n");

 WaitForSingleObject(pi.hProcess, INFINITE);

 // Terminación controlada del proceso e hilo asociado de ejecución

if(CloseHandle(pi.hProcess) !=0){

printf("El manejador del proceso fue cerrado de forma exitosa\n");}

else{

    printf("Algo no sucedio como debia");

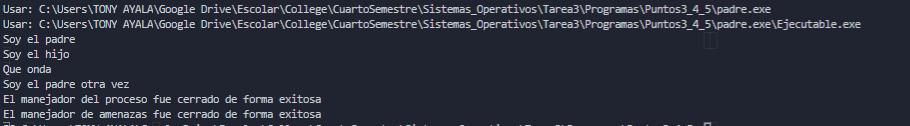
}

if(CloseHandle(pi.hThread) != 0)

printf("El manejador de amenazas fue cerrado de forma exitosa\n");

return 0;

}

Dando como parámetro a la función creadora del proceso hijo la ruta donde se encuentra dicho proceso y agregando condicionales a aquellos manejadores que cierran los procesos para verificar si fueron cerrados de forma exitosa.

1. Compare y reporte tanto las diferencias como similitudes que encuentra con respecto a la creación de procesos por sustitución de código en Linux.
2. Programe una aplicación que cree un proceso hijo a partir de un proceso padre, el hijo creado a su vez creará 5 procesos hijos más. A su vez cada uno de los cinco procesos creará 3 procesos más. Cada uno de los procesos creados imprimirá en pantalla su identificador. CONSEJO: INVESTIGUE LA FUNCIÓN GetCurrentProcessId() DEL API WIN32.
3. Programe las aplicaciones desarrolladas en el punto 5 de la sección de Linux (tanto la de procesos como la secuencial) utilizando esta vez la creación de procesos en Windows.

# Conclusiones

La realización de esta practica me gusto mucho, debido a que pude tener una interacción mas profunda con los sistemas operativos que día a día manejo, lo cual me permitió comprender mucho mas la forma en que estos funcionan, me pareció muy interesante todo el concepto de los procesos y la forma en que nosotros podemos manipularlos a nuestra conveniencia para cualquier situación que se pueda presentar, como todo una cosa lleva a otra y realmente es lo que sucede con esta práctica, está llena de tantos conceptos tan interesantes que realmente no es difícil o pesado estar leyendo y practicando con los mismos.

***-Mora Ayala José Antonio***

***-Ramírez Cotonieto Luis Fernando***

Esta práctica me gusto bastante ya que no sabía que podíamos ejecutar de manera simultánea o casi simultánea 2 programas o procesos a la vez, esto es una manera muy cercana y parecida de la multitarea aunque algo limitada donde podemos ejecutar más de 2 programas a la vez y podemos ejecutar nuevos procesos o cerrar los ya creados únicamente usando la función fork() y execv(), al menos en el sistema operativo de Linux es muy sencillo estas funciones, aunque en Windows es mas complejo debido a las incompatibilidades o complejidad de los comandos en los programas, si se quiere hacer uso de estos comandos de manera más sencilla es recomendable que se haga en el entorno de Linux

***-Torres Carrillo Josehf Miguel Angel***

***-Rodrigo Tovar Jacuinde***