

Instituto Politécnico Nacional

Escuela Superior de Cómputo

*Sistemas Operativos*

*“Práctica 4. Administrador de procesos en Linux y Windows (2)”*

**Grupo:** 2CM8

**Integrantes:**

* Martínez Coronel Brayan Yosafat.
* Monteros Cervantes Miguel Angel.
* Ramírez Olvera Guillermo.
* Sánchez Méndez Edmundo Josue.

**Profesor:** Cortés Galicia Jorge



**Práctica 4. Administrador de procesos en Linux y Windows (2) Introducción**

En la práctica anterior utilizamos procesos para realizar los programas y ahora en esta práctica ocupamos hilos, ¿Cuál es la diferencia?, ¿Cuál usar?, ¿Hay semejanzas? Todo lo anterior y un poco mas lo veremos a continuación.

Para empezar, ¿cómo funcionan los hilos?

Un hilo es la unidad de ejecución dentro de un proceso. Un proceso puede tener desde un solo hilo hasta muchos hilos. Cuando se inicia un proceso, se le asigna memoria y recursos. Cada hilo en el proceso comparte esa memoria y recursos. En los procesos de single-thread (solo un hilo), el proceso contiene un hilo. El proceso y el hilo son uno y el mismo, y solo está sucediendo una cosa. En los procesos de multithread (múltiples hilos=, el proceso contiene más de un hilo, y el proceso está logrando una serie de cosas al mismo tiempo (técnicamente, a veces es *casi* al mismo tiempo)

Hablamos de los dos tipos de memoria disponibles para un proceso o hilo, la pila y el heap (montículo). Es importante distinguir entre estos dos tipos de memoria de proceso porque cada subproceso tendrá su propia pila, pero todos los hilos de un proceso compartirán el montículo.

Los hilos a veces se denominan procesos ligeros porque tienen su propia pila, pero pueden acceder a datos compartidos. Debido a que los hilos comparten el mismo espacio de direcciones que el proceso y otros hilos dentro del proceso, el costo operativo de la comunicación entre los hilos es bajo, lo cual es una ventaja. La desventaja es que un problema con un hilo en un proceso afectará ciertamente a otros hilos y la viabilidad del proceso en sí.

**Procesos vs Hilos**

|  |  |
| --- | --- |
| Procesos | HILOs |
| Los procesos son operaciones de peso pesado. | Los hilos son operaciones de menor peso. |
| Cada proceso tiene su propio espacio de memoria. | Los hilos utilizan la memoria del proceso al que pertenecen. |
| La comunicación entre procesos es lenta ya que los procesos tienen diferentes direcciones de memoria. | La comunicación entre hilos puede ser más rápida que la comunicación entre procesos porque los hilos del mismo proceso comparten memoria con el proceso al que pertenecen. |
| El cambio de contexto entre procesos es más caro. | El cambio de contexto entre hilos del mismo proceso es menos costoso. |
| Los procesos no comparten memoria con otros procesos. | Los hilos comparten memoria con otros hilos del mismo proceso. |

**¿Qué pasa con la concurrencia y el paralelismo?**

Una pregunta que podría hacerse es si los procesos o hilos pueden ejecutarse al mismo tiempo. La respuesta es, depende. En un sistema con múltiples procesadores o núcleos de CPU (como es común con los procesadores modernos), se pueden ejecutar múltiples procesos o hilos en paralelo. Sin embargo, en un solo procesador, no es posible tener procesos o hilos realmente ejecutándose al mismo tiempo. En este caso, la CPU se comparte entre procesos o hilos en ejecución utilizando un algoritmo de programación de procesos que divide el tiempo de la CPU y produce la ilusión de ejecución en paralelo. El tiempo asignado a cada tarea se denomina "intervalo de tiempo". El cambio entre tareas ocurre tan rápido que generalmente no es perceptible. Los términos *paralelismo* (ejecución simultánea genuina) y *concurrencia* (intercalado de procesos en el tiempo para dar la apariencia de ejecución simultánea), distinguir entre los dos tipos de operación simultánea real o aproximada.

**¿Por qué debería usar un hilo en lugar de usar un proceso?**

Hay dos razones de gran peso para preferir multihilo sobre multiprocesos:

* La comunicación entre hilos (compartir datos, etc.) es significativamente más sencilla de programar que la comunicación entre procesos.
* Los cambios de contexto entre hilos son más rápidos que entre procesos. Es decir, es más rápido para el sistema operativo detener un hilo y comenzar a ejecutar otro que hacer lo mismo con dos procesos.

Ahora un ejemplo práctico:

Las aplicaciones con GUI suelen utilizar un hilo para la GUI y otros para el cálculo en segundo plano. El corrector ortográfico de MS Office, por ejemplo, es un hilo independiente del que ejecuta la interfaz de usuario de Office. En tales aplicaciones, el uso de múltiples procesos daría como resultado un rendimiento más lento y un código difícil de escribir y mantener.

**1. Competencias.**

El alumno aprende a familiarizarse con el administrador de procesos del sistema operativo Linux y Windows a través de la creación de procesos ligeros (hilos) para el desarrollo de aplicaciones concurrentes sencillas.

**2. Desarrollo**

**2.1. Sección Linux**

**2.1.1. Información de las funciones**

**2.1.1.1. pthread\_create()**



Funcion que inicia un nuevo hilo en el proceso de invocación. Bibliotecas necesarias:

* pthread.h

Valor de retorno:

* 0 = En caso de éxito.
* ‘-1’ = En caso de error.

**2.1.1.2. pthread\_join()**



La funcion *pthread\_join()* espera el hilo especificado por hilo para terminar. Si varios subprocesos simultáneamente intentan unirse con el mismo subproceso, los resultados pueden variar. Si *retval* no es NULL, *pthread\_join()* copia el estado de salida del hilo objetivo (es decir, el valor que el hilo objetivo suministró a *pthread\_exit*) en la ubicación apuntada por *retval* .

Bibliotecas necesarias:

* pthread.h

**2.1.1.3. pthread\_self()**



Devuelve el ID del hilo de llamada. Este es el mismo valor que se devuelve en \* thread en el pthread\_create llamada que creó este hilo.

Bibliotecas necesarias:

* pthread.h

**2.1.1.4. pthread\_exit()**



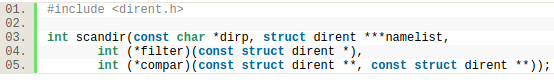
Finaliza el hilo de llamada y regresa un valor vía *retval* que (si el hilo se puede unir) está disponible para otro hilo en el mismo proceso que llama a pthread\_join .

Bibliotecas necesarias:

* pthread.h

Nota: Para las funciones anteriores es necesario compilar agregando *-l pthread*

**2.1.1.5. scandir()**



Escanea el directorio *dirp* , llamando al *filter*() en cada entrada del directorio. Las entradas para las cuales *filter*() devuelve un valor distinto de cero se almacenan en cadenas asignadas mediante *malloc*, ordenadas mediante *qsort*() con la funcion de comparación *compar*() y recopiladas en la lista de nombres de la matriz asignada a través de *malloc*. Si el filtro es NULL, se seleccionan todas las entradas.

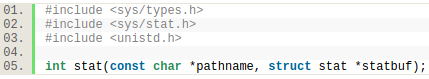
Valor de retorno:

* Devuelve la cantidad de entradas de directorio seleccionadas.
* En caso de error, se devuelve -1.

Bibliotecas necesarias:

* dirent.h

**2.1.1.6. stat()**



Devuelven información sobre un archivo. Muestra el archivo al que apunta *pathname* y rellena *statbuf* . Valor de retorno:

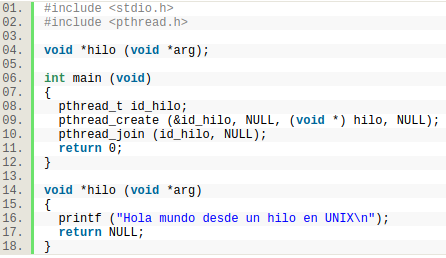
* En caso de éxito, se devuelve cero.
* En caso de error, se devuelve -1.

Bibliotecas necesarias:

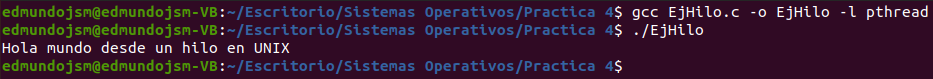
* sys/types.h
* sys/stat.h
* unistd.h

**2.1.2. Ejemplo de hilos en Linux**

Código(EjHilo.c)



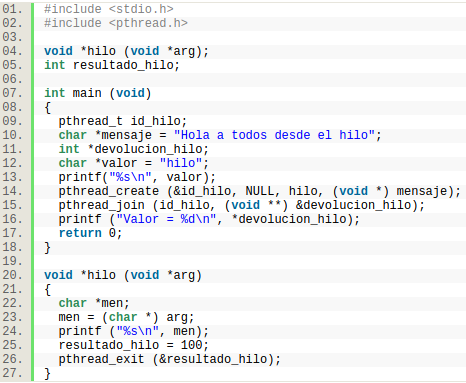
Compilación y ejecución del programa:



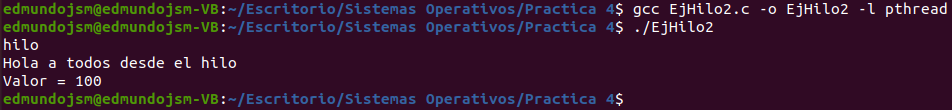
Creamos un hilo y a través de este mandamos el mensaje ’Hola mundo desde un hilo en UNIX’.

**2.1.3. Otro ejemplo de hilos en Linux**

Código(EjHilo2.c)



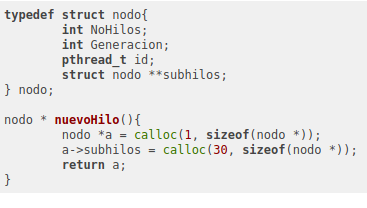
Compilación y ejecución del programa:



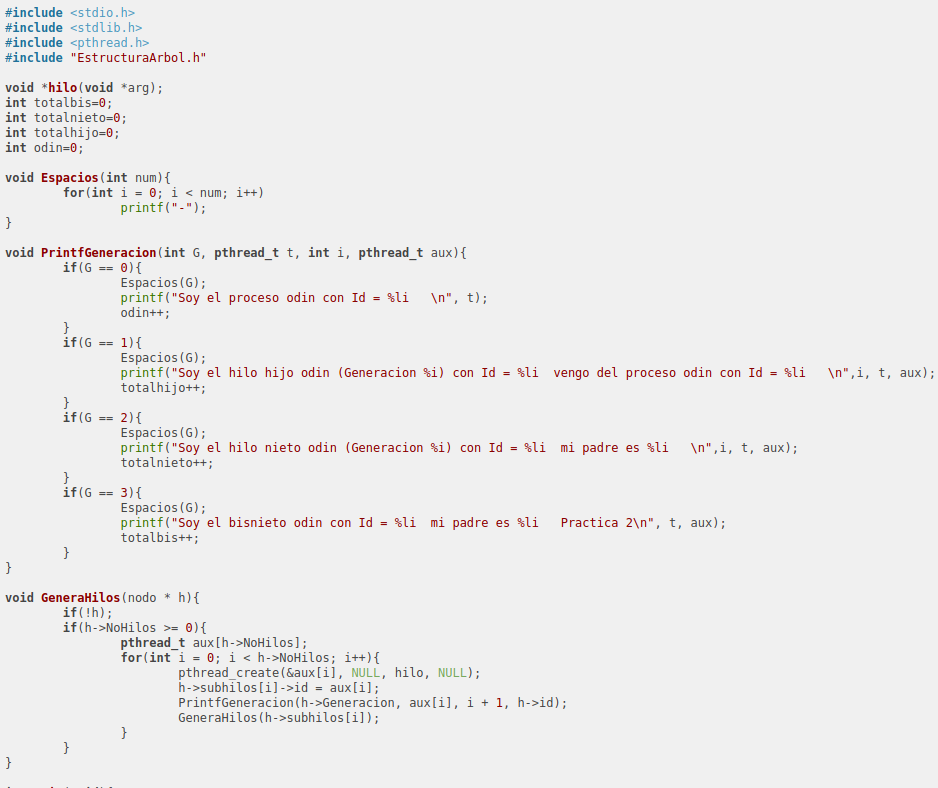
Creamos un hilo y a través de este mandamos el mensaje ’Hola a todos desde el hilo’ y también modificamos un valor que tenemos en el main.

**2.1.4. Árbol de hilos**

Código(EstructuraArbol.h)

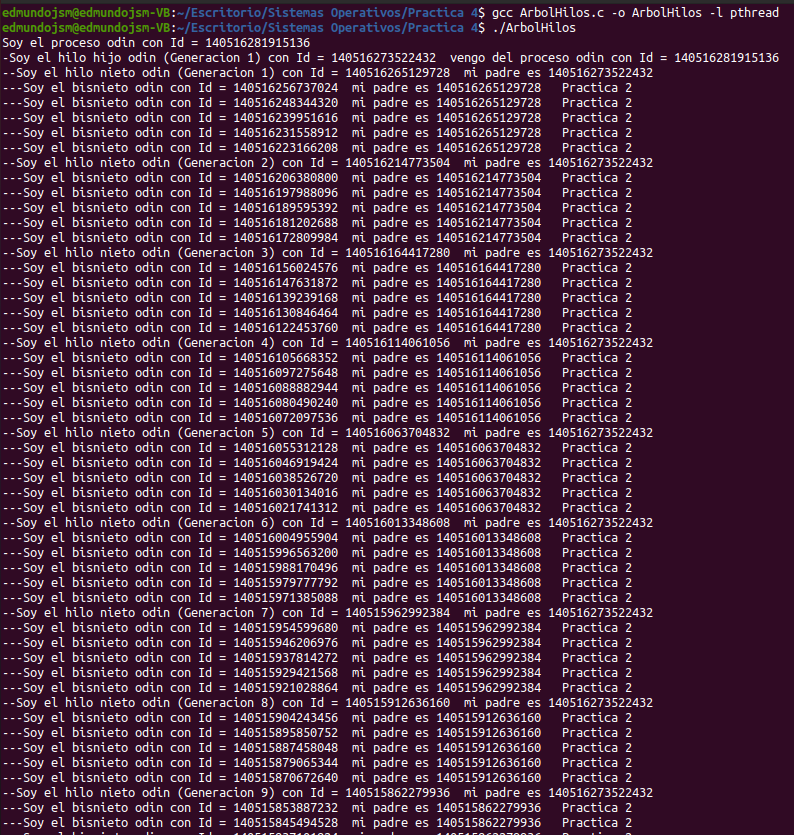


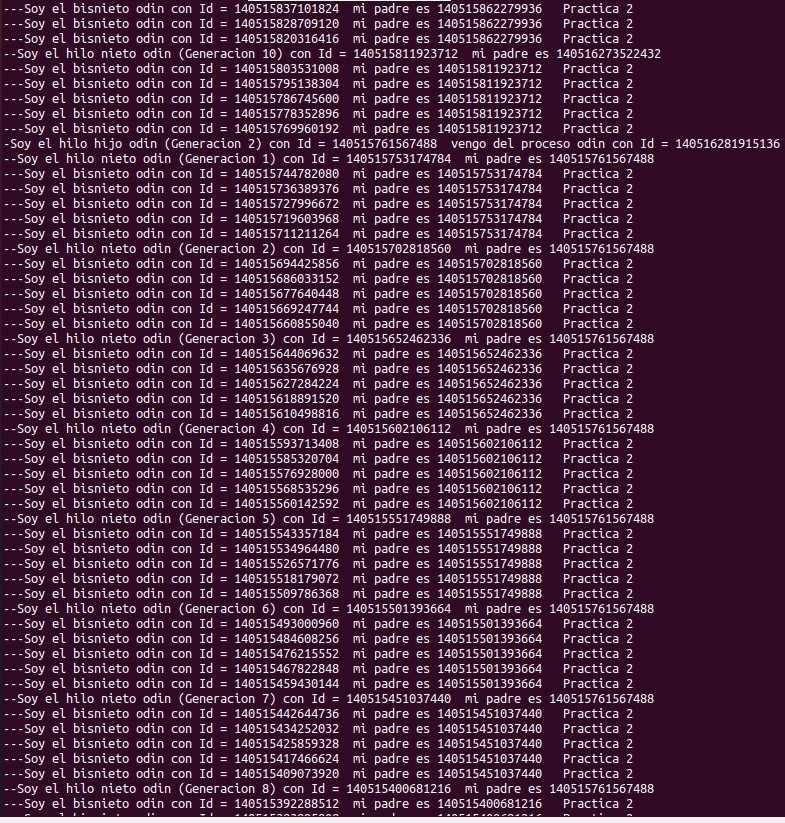
Código(ArbolHilos.c)

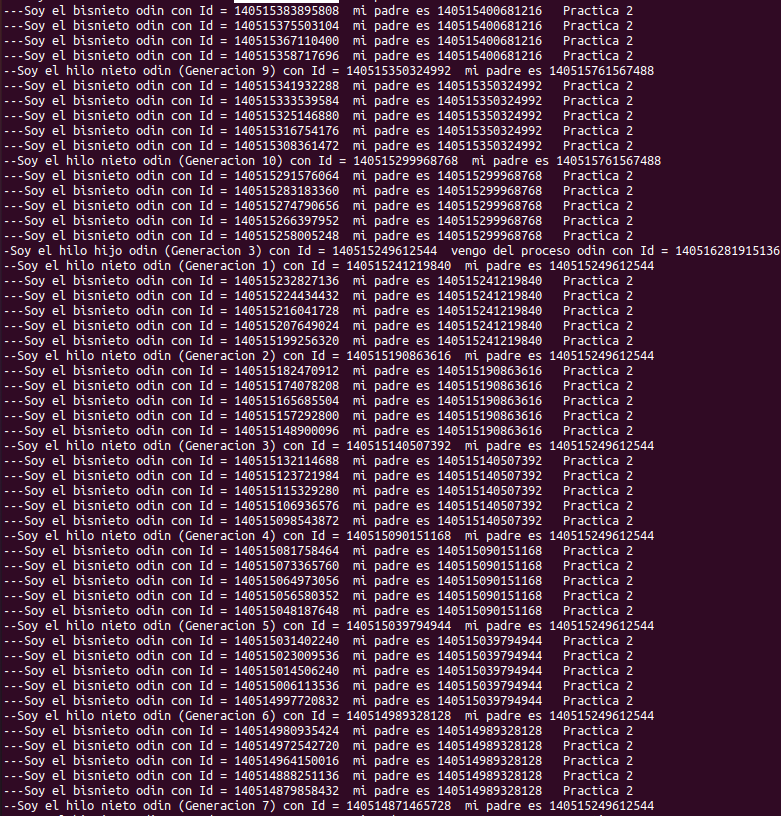


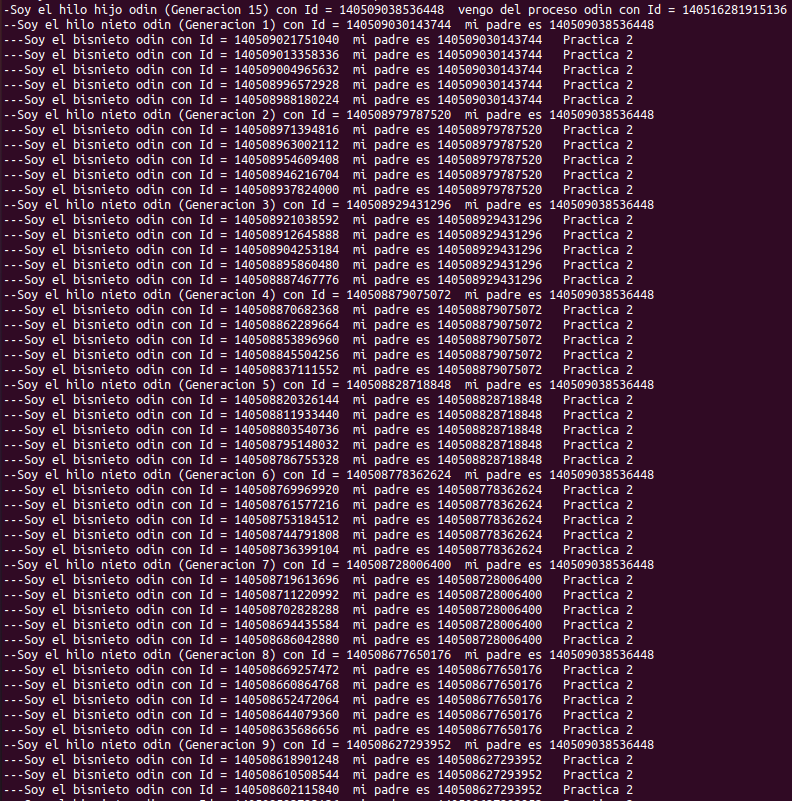


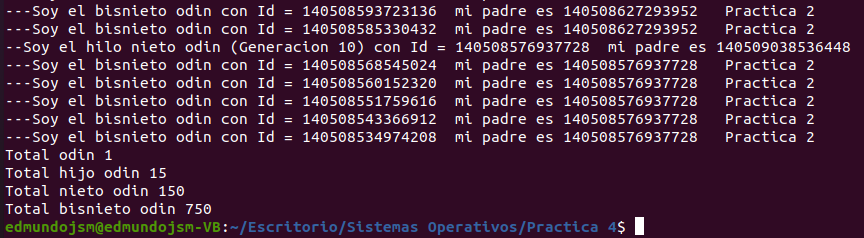
Compilación y ejecución del código.







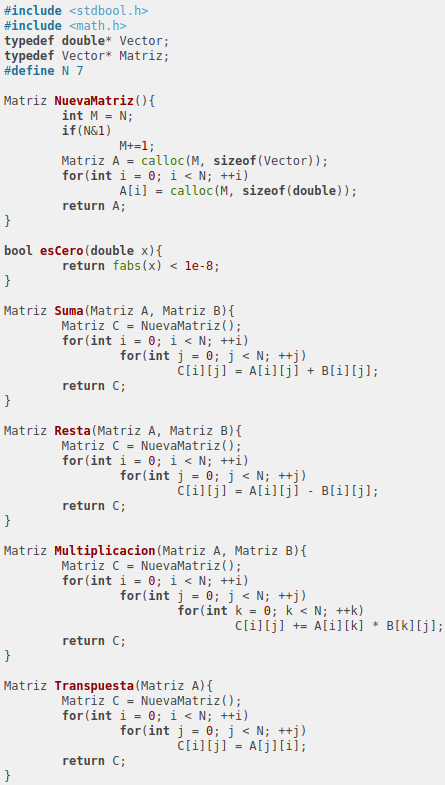




Creamos un hilo principal y después en ese hilo generamos 15 hilos con 10 hilos en cada uno, y en los 10 hilos creamos 5 hilos en cada hilo con la llamada al sistema *pthread\_create*() y terminamos cada hilo *pthread\_exit*(). En cada hilo se manda su respectivo *pthread\_t* que es su id, el id hilo de su padre, la generación de cada hilo para el caso de los hijos y nietos de Odín, sin embargo, para los bisnietos de Odín también se muestra ’Practica 4’. Al final se colocan los valores de cuantos Odín, hijos de Odín, nietos de Odín y bisnietos de Odín se crearon para comprobar que el árbol de hilos se creo correctamente.

Nota: Debido a la cantidad de datos imprimidos en pantalla se decidió no poner todo el resultado ya que se imprimieron 919 líneas las cuales son muchas y más de la mitad del documento seria de puras capturas de pantalla.

**2.1.5. Aplicación con 5 hilos (Punto 5 practica 3 pero ahora son hilos)**

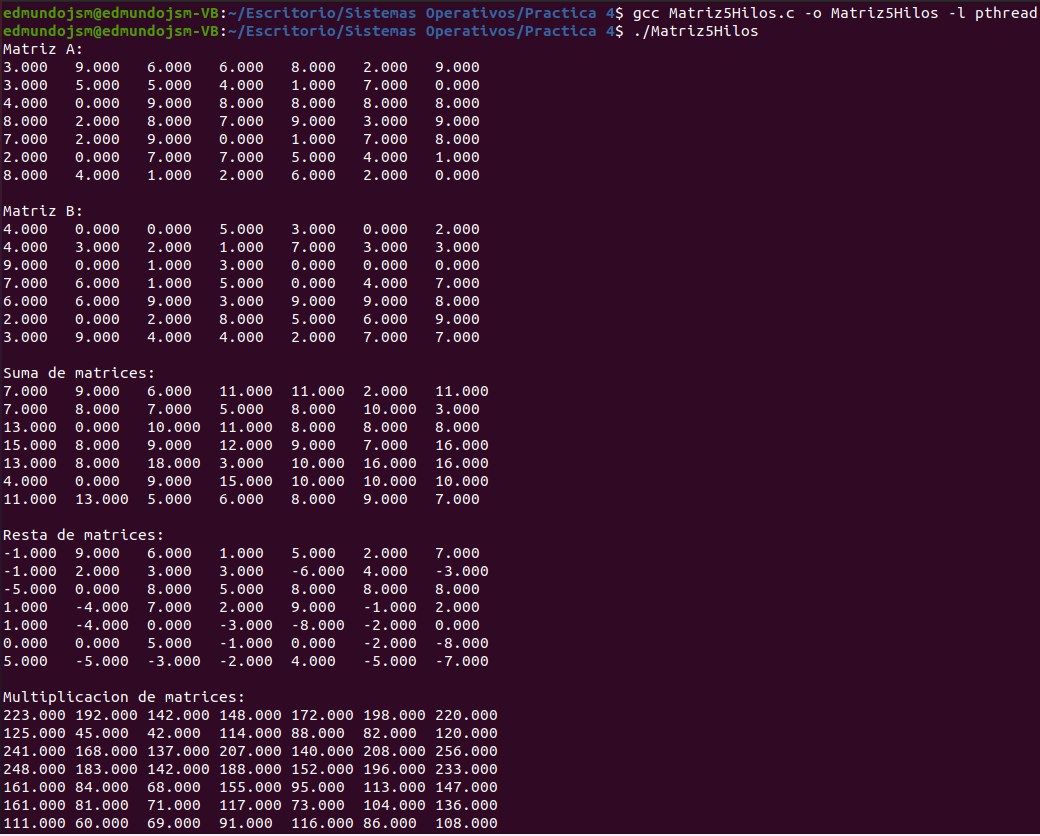
Código(EstructuraMatriz.h)

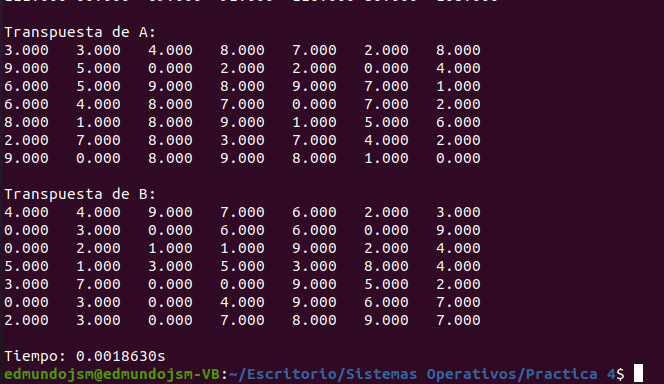
Código(Matriz5Hilos.c)



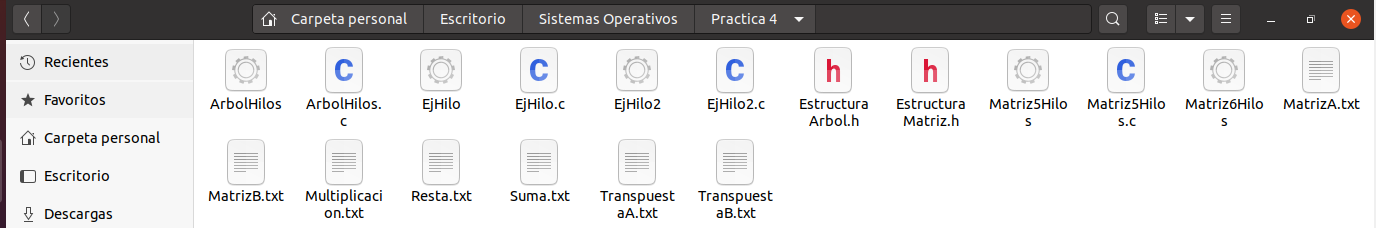


Compilación y ejecución del código.





Carpeta donde se crearon los respectivos archivos operación generada por el respectivo hilo (aquí también se hicieron las pruebas de los programas anteriores que se realizaron):

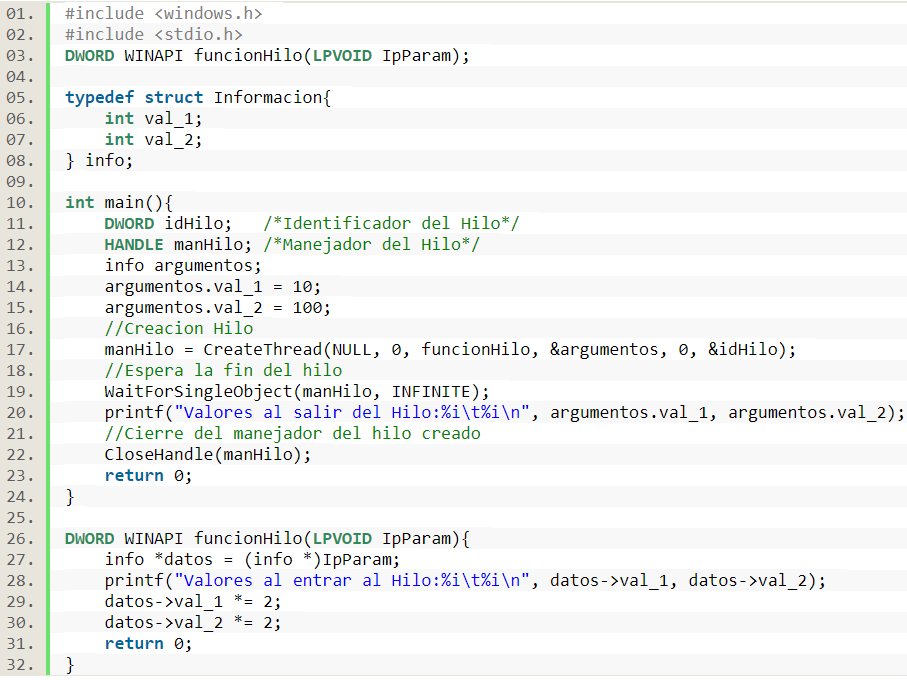


Creamos 5 hilos diferentes con la llamada al sistema pthread\_create() los cuales realizaran distintas operaciones con matrices en cada hilo mencionar que se tuvieron unos problemas al momento de crear y leer las matrices creadas ya que al no estar en una misma función no se puede usar de forma global ya que las matrices se arman de forma dinámica, pero el problema fue solucionado al momento de que el hilo ejecuta su tarea correspondiente. Y a diferencia de los procesos, los hilos son más tardados en el momento de ejecución.

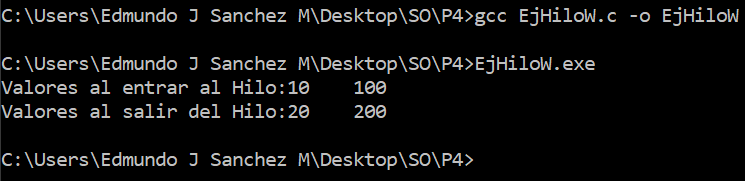
**2.2. Sección Windows**

**2.2.1. Ejemplo de hilos en Windows**

Código(EjHiloW.c)



Compilación y ejecución del código.

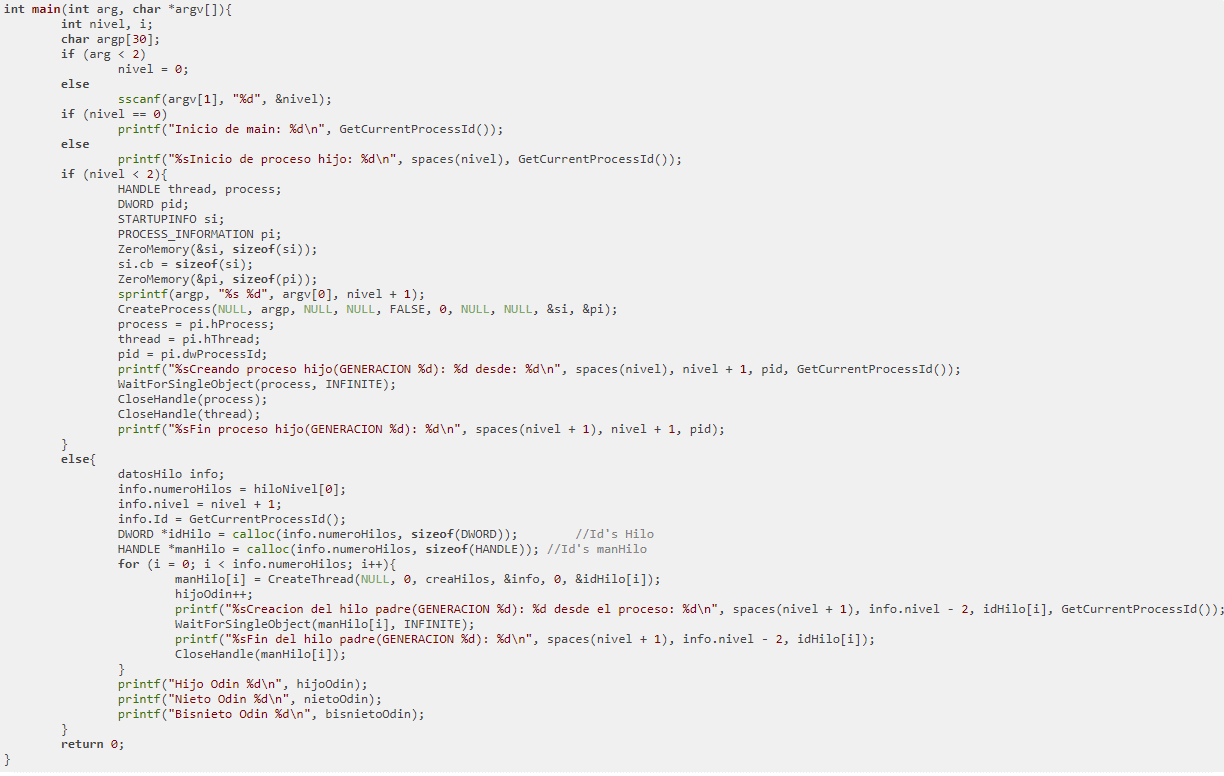


Creamos un hilo y a través de este al entrar al hilo nos muestra el valor inicial de los valores después antes de salir duplicamos esos valores para que al final al salir del hilo se vea esa acción.

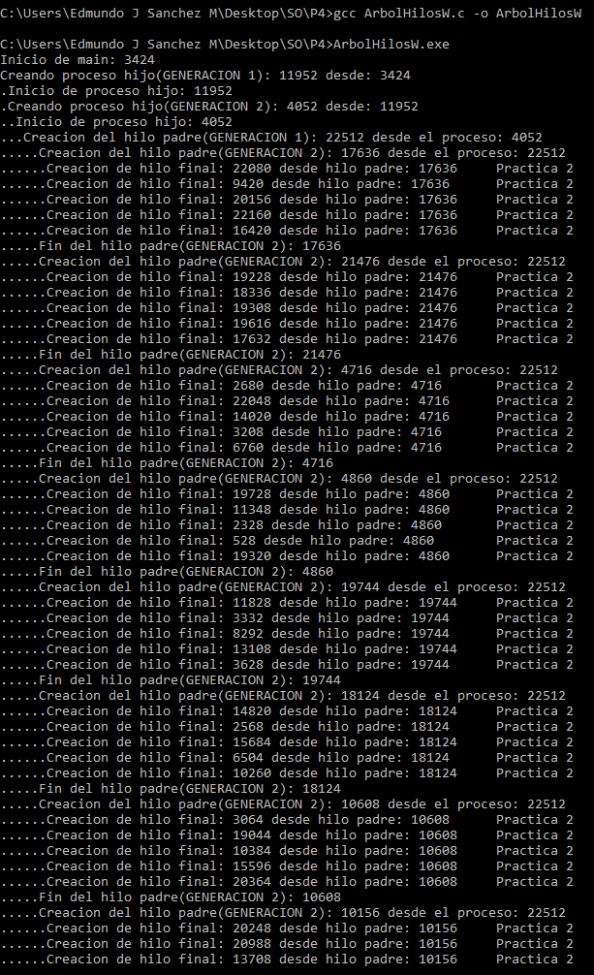
**2.2.2. Árbol de hilos**

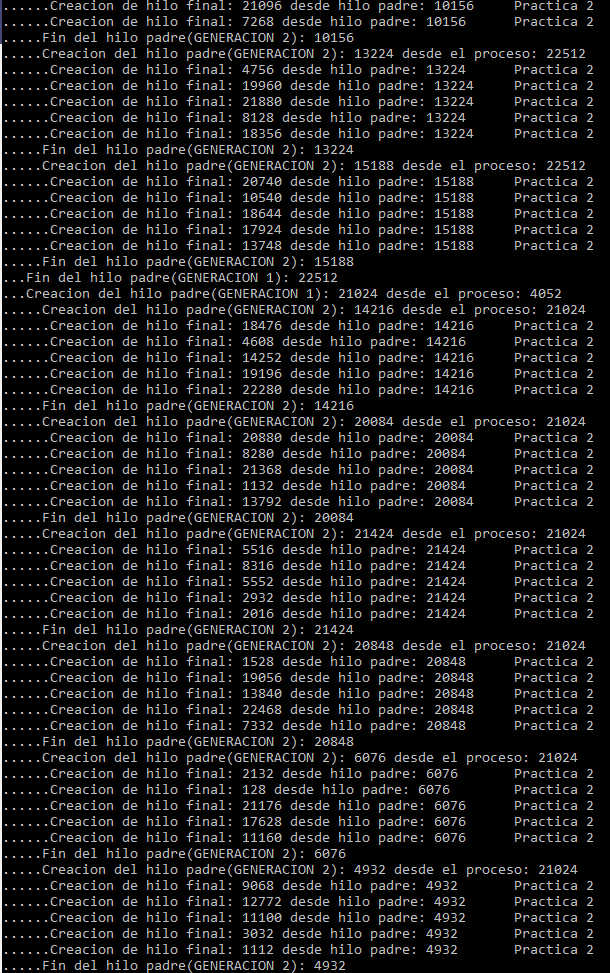
Código(ArbolHilosW.c)

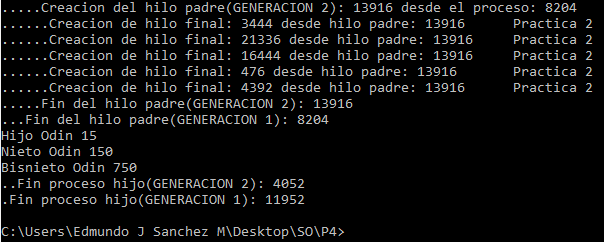




Compilación y ejecución del código.







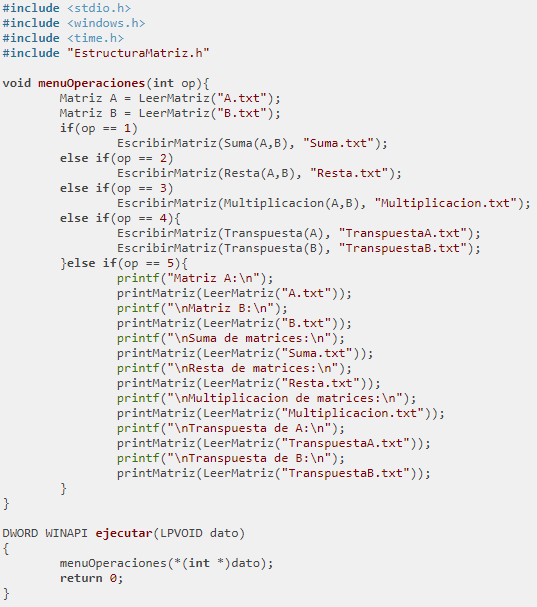
Primero se crea un proceso padre a partir del main, después desde el proceso padre se crea un proceso hijo. Desde el proceso hijo se crean 15 hilos, cada uno de estos crea 10 hilos y a su vez cada uno crea 5 hilos; estos imprimirán ‘Practica 4’. Para crear proceso se usa la llama al sistema: *CreteProcess*() antes estudiada y para la creación de hilos se usa: *CreateThread*(). Al final se colocan los valores de cuantos hijos de Odín, nietos de Odín y bisnietos de Odín se crearon para comprobar que el árbol de hilos se creó correctamente.

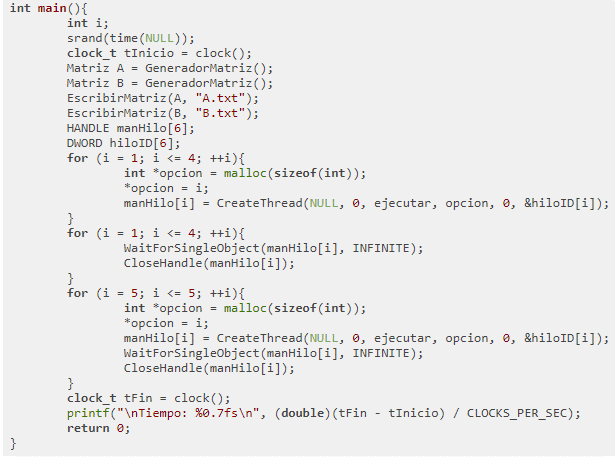
Nota: Debido a la cantidad de datos imprimidos en pantalla se decidió no poner todo el resultado ya que se imprimieron poco más de 910 líneas las cuales son muchas y más de la mitad del documento seria de puras capturas de pantalla.

**2.2.3. Aplicación con 5 hilos (Punto 5 practica 3 pero ahora son hilos)**

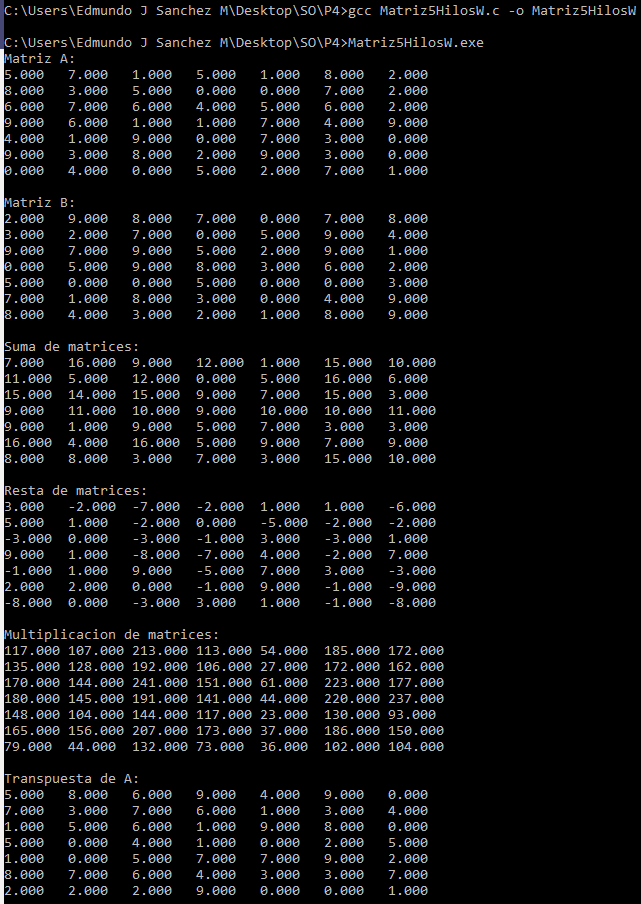
Se hace uso del mismo archivo EstructuraMatriz.h para las operaciones correspondientes con las matrices, por lo que no se vuelva a agregar para no alargar sin necesidad este documento.

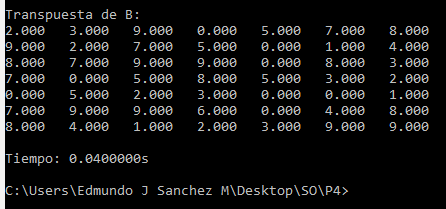
Código(Matriz5HilosW.c)



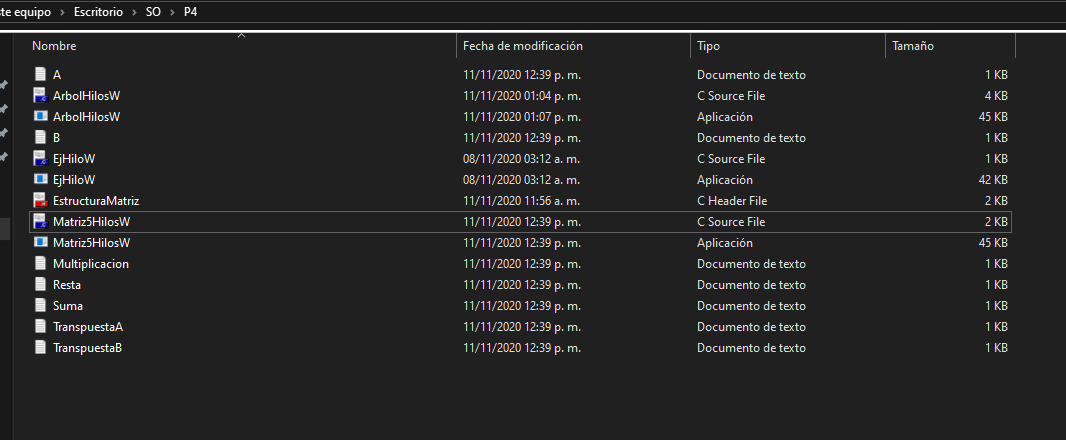


Compilación y ejecución del código.





Carpeta donde se crearon los respectivos archivos operación generada por el respectivo hilo (aquí también se hicieron las pruebas de los programas anteriores que se realizaron):



Como vemos solo se adaptó la aplicación para que se ejecute con hilos. Se puede observar en el tiempo de ejecución del programa que es más bajo que el de procesos, sin embargo, creo recordar que la vez que realice el de procesos tenía más programas abiertos de los que actualmente tengo, algo que sin duda alguna afecta a la ejecución del problema.

**3. Análisis de la práctica**

**3.1 Linux**

En esta práctica se hizo uso de llamadas al sistema para la creación y manipulación de hilos los cuales se ejecutan de manera paralela y concurrente, se pueden crear uno tras otro o varios en el mismo momento. Las funciones de las llamadas al sistema de los hilos son:

* Crear un hilo, varias veces el mismo o crear distintos con diferente *pthread\_t*.
* Identificar un hilo con su *pthread\_t*.
* Finalizar un hilo.
* Esperar a que termine un hilo.

Son diferentes la forma de crear un hilo tras otro al crear varios hilos a la vez, su manejo de los *pthread\_t* son distintos. Los hilos pueden esperar a que finalice otro hilo o finalizar en un momento establecido.

**3.2 Windows**

Al igual que Windows se deben usar llamadas al sistema para crear hilos. Lo mínimo para crear hilos, mediante llamadas al sistema, son:

* HANDLE para manejar los hilos creados.
* Crear el hilo: *CreatThread*().
* Función en donde se va a utilizar el hilo.
* Parámetros para la funcion donde se va a utilizar el hilo; que después, dentro la función se debe castear al tipo de puntero original.

**4. Observaciones**

**4.1. Linux**

* Al crear un hilo este tiene un *pthread\_t* que es un identificador el cual se puede conocer con la llamada al sistema *pthread\_self*().
* Se pueden crear distintos hilos creando un arreglo de *pthread\_t* con el número de hilos que se desea crear y mandar cada *pthread\_t* a la llamada al sistema pthread\_create().
* Se debe tener cuidado al querer crear distintos hilos, ya que no funcionan de la misma manera que en la creación de procesos con la llamada al sistema *fork*() el cual genera un *id\_proc* diferente si se invoca de la misma manera en diferentes líneas del código.
* Las llamadas al sistema pthread\_join() y pthread\_exit() para hilos son similares con las llamadas al sistema para la creación de procesos que son *wait*() y *exit*().
* Se debe tener cuidado con la llamada al sistema pthread\_join() si se quiere crear varios hilos diferentes porque, aunque tengamos un arreglo de *pthread\_t* puede ocupar el mismo *pthread\_t* en la llamada al sistema pthread\_create(). También se debe tener cuidado con la llamada al sistema pthread\_exit () por que puede dejar de crear los hilos antes de la cantidad solicitada.
* La mayoría de las llamadas al sistema para la manipulación de hilos se encuentran en la biblioteca ”pthread.h”.
* Si un hilo padre finaliza también finalizara el hilo hijo.

**4.2. Windows**

* Así mismo como la funcion *CreateProcess*(), para usar *CreateThread*() se usa la biblioteca ”windows.h”.
* El orden ”jerárquico” de espera es que primero debe finalizar el hilo hijo antes del hilo padre, de lo contrario el hilo hijo se destruirá sin importar en que se esté ocupando.
* Es relativamente más fácil usar hilos debido a la simplicidad de la funcion *CreateThread*() en comparación con *CreateProcess*(), a parte se necesitan menos argumentos para *CreateThread*().
* Es posible pasar de argumentos de una forma “indirecta” para que los trabaje el hilo.
* Se le puede especificar al hilo en que función se quiere que esté trabajando.
* La funcion *CreateThread* devuelve un HANDLE.
* Con los hilos solo se debe cerrar un HANDLE, que es el que devuelve la funcion *CreateThread*. Con procesos se debían cerrar 2 HANDLES.
* La funcion donde entra el hilo que se crea recibe como argumento un LPVOID que es un tipo de puntero ”genérico”, o sea que es un puntero a cualquier tipo de dato. Es por eso por lo que se debe castear dentro de la función al tipo de dato original que se la pasó a dicha funcion.
* Para que se lleve a cabo la tarea asignada al hilo en cuestión se usa la funcion *WaitForSingleObject*(), la misma usada para procesos.

**4.3. Generales**

* Los hilos reciben como argumento principal la funcion que se ejecutara en él. Vemos que en la creación de procesos por sustitución de código requeríamos la ruta al ejecutable, de ahí el porqué de que los hilos son “procesos ligeros”.
* Si finaliza el programa principal, también lo harán todos los hilos que creó, pues son dependientes a él.
* La mejor forma de pasarle argumentos a una función dentro de un hilo es por medio de una estructura.
* Si creamos hilos dentro de un bucle *for* y a cada uno le pasamos como argumento una referencia al iterador (por ejemplo, en el programa de las matrices) no funcionara, pues la variable estará cambiando de valor y puede que cada uno de los hilos no reciba el valor que le tocaba. Para solucionarlo hay que copiar la variable en otra y pasarle la referencia de esta nueva variable al hilo.
* Para hacer que realmente los hilos se ejecuten concurrentemente, primero se crean todos los hilos necesarios y después esperamos a cada uno de ellos. Hacerlo de forma anidada es incorrecto.

**5. Conclusiones**

En ambos sistemas operativos Linux y Windows tenemos diversas llamadas al sistema para la creación y manipulación de hilos, ambos sistemas crean, finalizan y esperan a los hilos. Las llamadas al sistema tienen un valor de retorno el cual ayuda al control de los hilos.

Es apropiado usar hilos cuando se quieren tareas simples y cortas, por el mismo hecho que es más fácil manejar hilos que procesos, aunque es importante recalcar que los hilos son más lentos que los procesos por lo menos esto sucedió en la parte de Linux, se esperaba que ocurriera lo mismo en Windows, pero vaya sorpresa nos dio los hilos fueron más rápidos.