INSTITUTO POLITÉCNICO NACIONAL ESCUELA SUPERIOR DE CÓMPUTO

SISTEMAS OPERATIVOS

## PROFESOR: CORTES GALICIA JORGE

CICLO ESCOLAR: 2024-2

***PRACTICA 6 – ADMINISTRADOR DE PROCESOS EN LINUX Y WINDOWS (2)***

# 20 DE MAYO DE 2024

## INTRODUCCION

En esta práctica de sistemas operativos, se profundiza en la gestión de hilos y la manipulación de archivos y directorios, aspectos fundamentales para la programación concurrente y el manejo eficiente de recursos en un sistema. La capacidad de crear, controlar y sincronizar hilos permite a las aplicaciones ejecutar múltiples tareas simultáneamente, mejorando su rendimiento y capacidad de respuesta.

Las funciones de la biblioteca pthread en Linux, tales como **pthread\_create**, **pthread\_join**, **pthread\_self** y **pthread\_exit**, proporcionan un marco robusto para el manejo de hilos. **pthread\_create** permite iniciar nuevos hilos dentro del mismo proceso, mientras que **pthread\_join** facilita la sincronización, asegurando que un hilo principal pueda esperar la finalización de otros hilos. **pthread\_self** devuelve el identificador del hilo actual, y **pthread\_exit** permite la terminación controlada de un hilo, devolviendo un valor que puede ser capturado por otros hilos. Estas funciones son esenciales para construir aplicaciones multihilo eficientes.

Además, se exploran funciones del sistema para la manipulación de archivos y directorios, como **scandir** y **stat**. **scandir** permite escanear un directorio y crear una lista de sus entradas, aplicando filtros y ordenaciones según sea necesario, lo que resulta útil para aplicaciones que necesitan interactuar con el sistema de archivos de manera dinámica y organizada. Por otro lado, **stat** proporciona información detallada sobre archivos, incluyendo su tamaño, permisos y fechas de modificación, permitiendo a los programas gestionar y verificar el estado de los archivos de manera precisa.

En el contexto de Windows, se utiliza la función **CreateThread** para la creación de hilos. Aunque su uso y algunos aspectos de implementación difieren de las funciones pthread en Linux, **CreateThread** cumple un rol similar en la gestión de la concurrencia dentro de un proceso, facilitando el desarrollo de aplicaciones multihilo en esta plataforma.

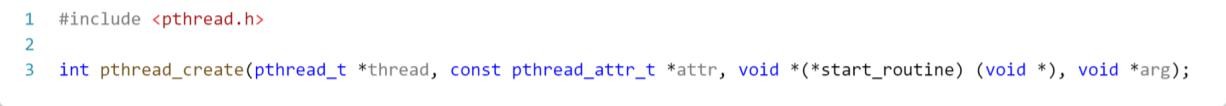
Un aspecto importante de la práctica es la comparación de tiempos de ejecución entre programas que utilizan procesos y aquellos que utilizan hilos. Los hilos, al compartir el mismo espacio de direcciones, suelen ser más ligeros y rápidos de crear y destruir en comparación con los procesos, que requieren mayor overhead para mantener espacios de direcciones separados. Esta comparación es crucial para entender las ventajas y limitaciones de cada enfoque y elegir el más adecuado para optimizar el rendimiento de las aplicaciones.

## DESARROLLO

*1.- A través de la ayuda en línea que proporciona Linux, investigue el funcionamiento de las funciones:* ***pthread\_create(), pthread\_join(), pthread\_self(), pthread\_exit(), scandir(), stat()****. Explique los argumentos y retorno de cada función.*

#### Pthread\_create:

Crea un nuevo hilo en el proceso actual, el nuevo hilo empieza su ejecución invocando **start\_routine()**; **arg** se pasa como el único argumento de **start\_routine()**.



* **thread**: Es un puntero a una variable de tipo **pthread\_t** donde se almacenará el identificador del hilo creado.
* **attr**: Es un puntero a una estructura de atributos que especifica los atributos del hilo que se está creando. Puede ser **NULL** para utilizar los atributos por defecto.
* **start\_routine**: Es una función que será ejecutada por el hilo. Debe ser de tipo **void \*función(void \*)** y tomar un puntero **void** como argumento y devolver un puntero **void**.
* **arg**: Es un puntero que se pasará como argumento a la función

### start\_routine.

El nuevo hilo termina su ejecución de las siguientes maneras:

* Llama **pthread\_exit(3)**, especificando un valor de estado de salida que en el mismo proceso que llama a **pthread\_join(3)**.
* Se retorna de **start\_routine()**. Este es el equivalente a llamar a

**pthread\_exit(3)** con el valor declarado en el sentencia **return**.

* Se cancela llamando **pthread\_cancel(3)**.
* Cualquiera de los hilos llama **exit(3)**, o el hilo principal realiza un retorno desde la función **main()**. Esto causa la terminación de todos los hilos en el proceso.

La función retorna **0** cuando se ejecuta exitosamente, cuando llega a un error devuelve un valor de error y el contenido de **\*thread** es indefinido.

#### Pthread\_join:

Espera a que el hilo especificado por **thread** termine de ejecutarse. Si ese hilo ya terminó su ejecución, entonces la función retorna un valor de manera inmediata.

* **thread**: Es el identificador del hilo que se desea esperar.
* **retval**: Es un puntero a un puntero donde se almacenará el valor de retorno del hilo. Puede ser **NULL** si no se necesita el valor de retorno.



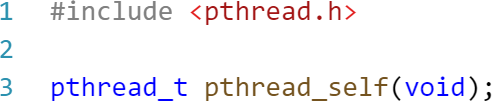
La función bloqueará la ejecución del hilo que la llama hasta que el hilo especificado por **thread** haya terminado su ejecución. Devuelve **0** si tiene éxito y un valor distinto de **0** en caso de error.

Si **retval** no es **NULL**, entonces la función copia el estado de salida del hilo objetivo (el valor que el hilo objetivo le envió a **pthread\_exit(3)**) a la localidad de memoria a la que apunta **retval**.

Si varios hilos hacen una llamada a **pthread\_join** con el mismo hilo, los resultados son indefinidos; si el hilo que llama a la función es cancelado, entonces al hilo objetivo se le podrá hacer join.

#### Pthread\_self:

Esta función devuelve el ID del hilo que la llama.



Esta función siempre se ejecuta de manera exitosa, devolviendo el identificador único del hilo.

#### Pthread\_exit:

Se utiliza para terminar la ejecución del hilo que la llama y devuelve un valor por medio de **retval**, ese valor podrá ser accedido por otro hilo que esté utilizando la función **pthread\_join()** en el hilo terminado.



* **retval**: Es un puntero que especifica el valor de salida del hilo. Puede ser

**NULL** si no se necesita devolver ningún valor.

Esta función siempre se ejecuta de manera exitosa y nunca devuelve nada al hilo que la llama.

#### Scandir:

Se utiliza para escanear un directorio y crear una lista de las entradas (archivos y subdirectorios) que contiene ese directorio, filtrando opcionalmente las entradas mediante una función de selección proporcionada por el usuario.



* **dir**: Es una cadena de caracteres que especifica la ruta del directorio que se va a escanear.
* **namelist**: Es un puntero a un puntero a una matriz de estructuras **dirent**. Después de que **scandir** tenga éxito, esta matriz contendrá las entradas del directorio.
* **filter**: Es un puntero a una función que se utilizará para filtrar las entradas del directorio. Puede ser **NULL** si no se desea aplicar ningún filtro.
* **compar**: Es un puntero a una función que se utiliza para comparar las entradas del directorio para ordenarlas en la lista resultante. Puede ser **NULL** si no se desea ordenar la lista.

La función devuelve el número de entradas escaneadas si tiene éxito, o -1 si ocurre un error. La lista de entradas escaneadas se almacena en **namelist**.

#### Stat:

Se utiliza para obtener información sobre un archivo especificado por su nombre de ruta. Esta función obtiene y almacena información sobre el archivo en una estructura **struct stat**.

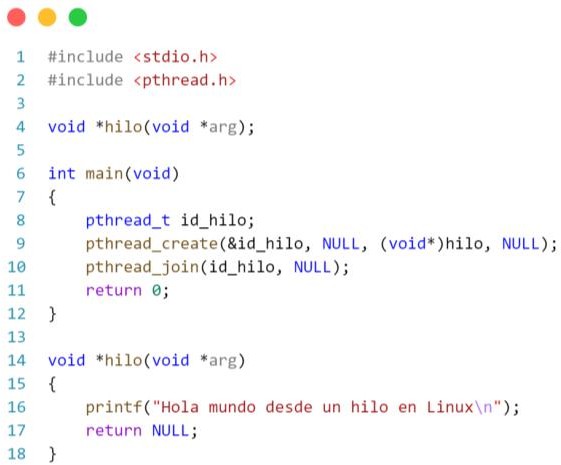
* **pathname**: Es una cadena de caracteres que especifica la ruta del archivo del cual se desea obtener información.
* **statbuf**: Es un puntero a una estructura **struct stat** donde se almacenará la información sobre el archivo.

La estructura **struct stat** contiene varios campos que proporcionan información sobre el archivo, como el tipo de archivo, los permisos, el tamaño, el tiempo de modificación, etc. Algunos de los campos comunes en la estructura **struct stat** incluyen:

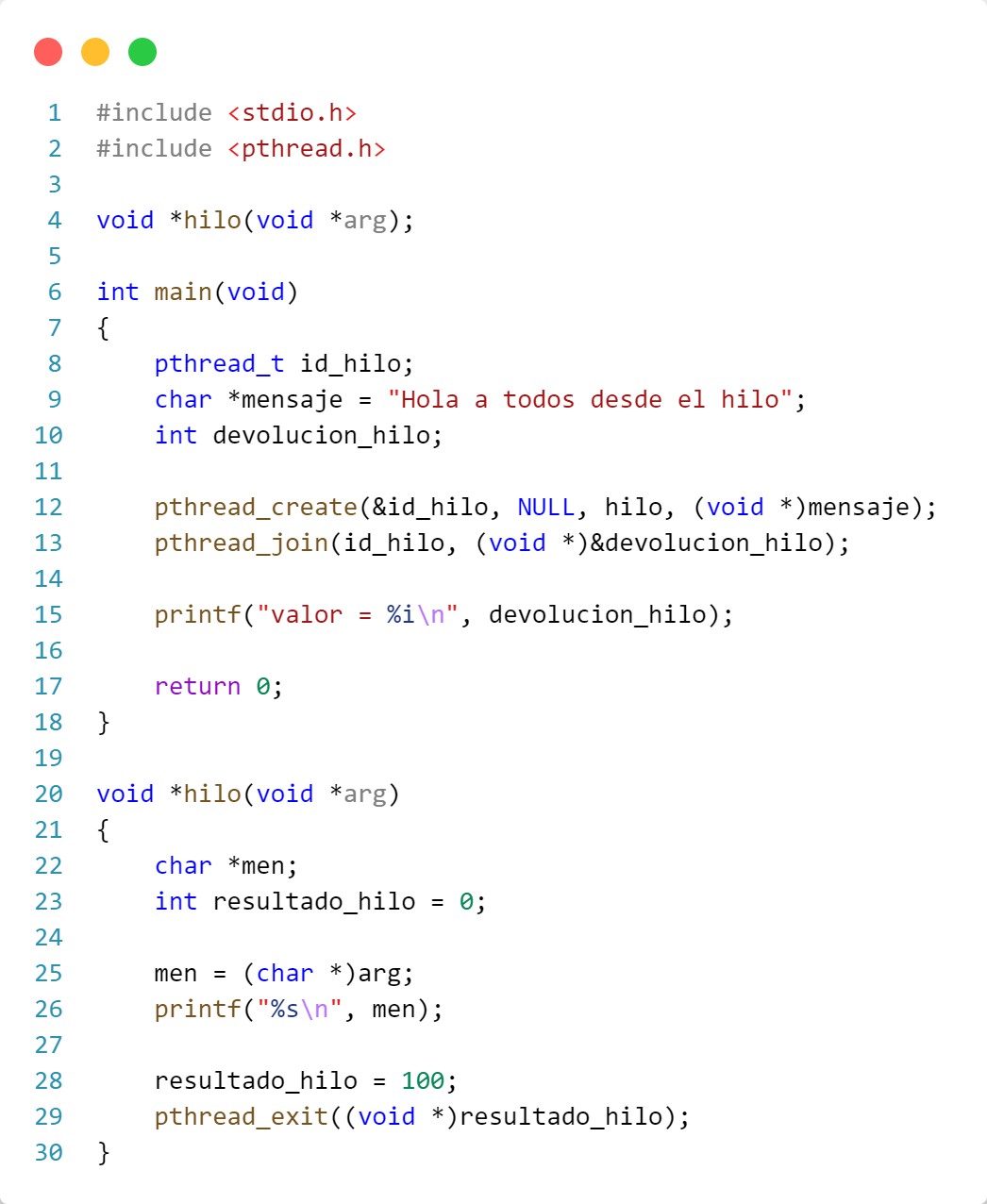
* **st\_mode**: Modo de archivo (permisos y tipo).
* **st\_size**: Tamaño del archivo en bytes.
* **st\_mtime**: Hora de la última modificación del archivo.
* **st\_uid**: Identificador de usuario del propietario del archivo.
* **st\_gid**: Identificador de grupo del propietario del archivo.

La función **stat** devuelve **0** si tiene éxito y **-1** si ocurre un error. En caso de error, se puede consultar la variable global **errno** para determinar la causa específica del error.

*2.- Capture, compile y ejecute el programa de creación de un nuevo hilo en Linux. Observe su funcionamiento.*



*3.- Capture, compile y ejecute el siguiente programa de creación de hilos en Linux. Observe su funcionamiento.*

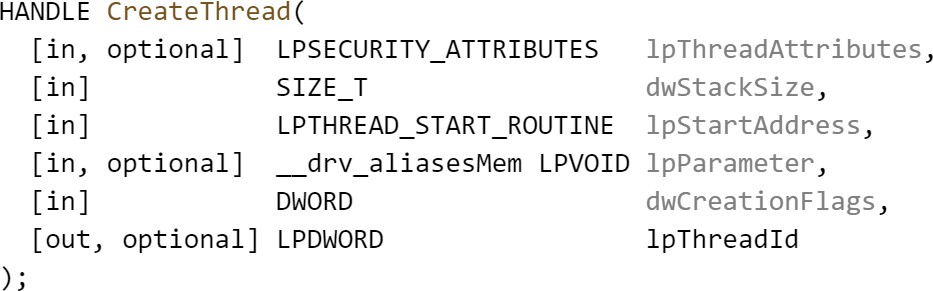


*4.- A través del sitio MSDN, investigue el funcionamiento de la llamada al sistema CreateThread().*

### Función CreateThread

Crea un hilo para ejecutar dentro del espacio de direcciones virtuales del proceso de llamada. Para crear un hilo que se ejecute en el espacio de direcciones virtuales de otro proceso, utilice la función **CreateRemoteThread**.

### Sintaxis:



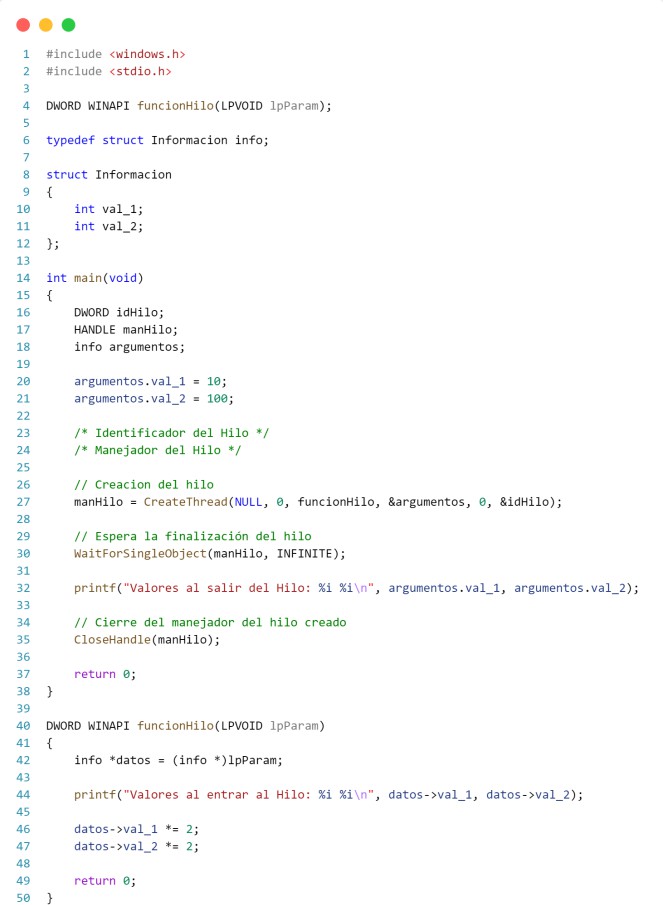
**Parámetros:**

* **[in, optional] lpThreadAttributes**: Un puntero a una estructura **SECURITY\_ATTRIBUTES** que determina si los procesos secundarios pueden heredar el identificador devuelto. Si **lpThreadAttributes** es **NULL**, el identificador no se puede heredar.
* **[in] dwStackSize**: El tamaño inicial de la pila, en bytes. El sistema redondea este valor a la página más cercana. Si este parámetro es cero, el nuevo hilo usa el tamaño predeterminado para el ejecutable.
* **[in] lpStartAddress**: Puntero a la función definida por la aplicación que ejecutará el subproceso. Este puntero representa la dirección inicial del hilo.
* **[in, optional] lpParameter**: Un puntero a una variable que se pasará al hilo.
* **[in] dwCreationFlags**: Las banderas que controlan la creación del hilo.
* **[out, optional] lpThreadId**: Puntero a una variable que recibe el identificador del hilo. Si este parámetro es **NULL**, no se devuelve el identificador del hilo.

### Valor de retorno:

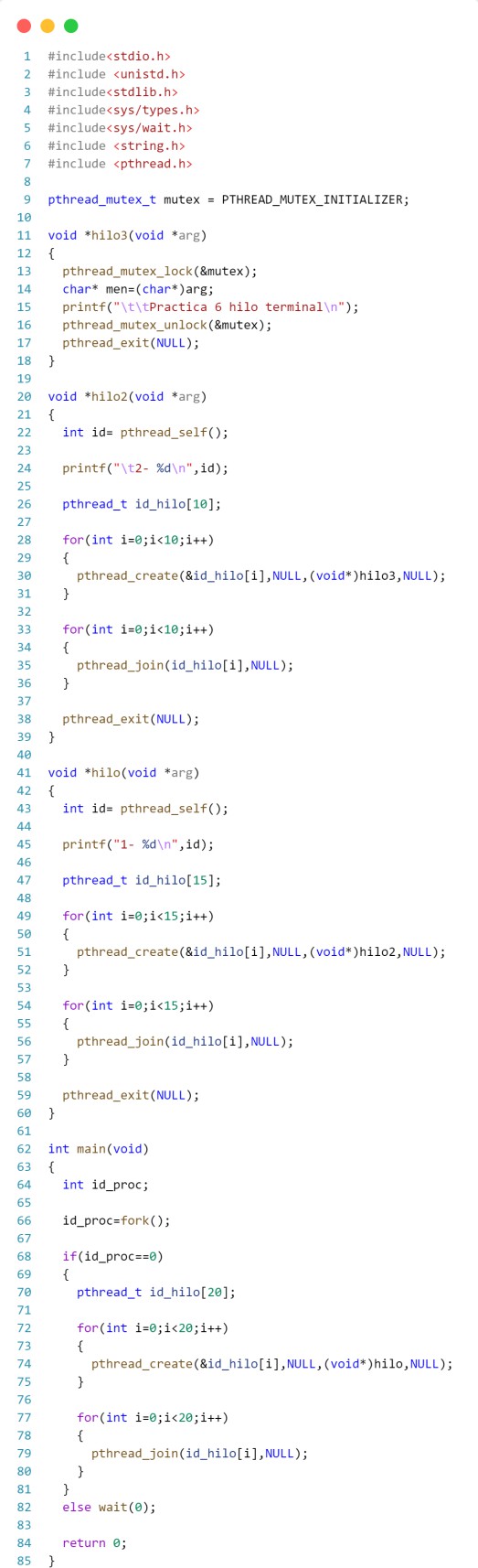
* Si la función tiene éxito, el valor de retorno es un identificador del nuevo hilo.
* Si la función falla, el valor de retorno es **NULL**. Para obtener información de error ampliada, llame a **GetLastError**.

*5.- Capture, compile y ejecute el siguiente programa de creación de hilos en Windows. Observe su funcionamiento.*

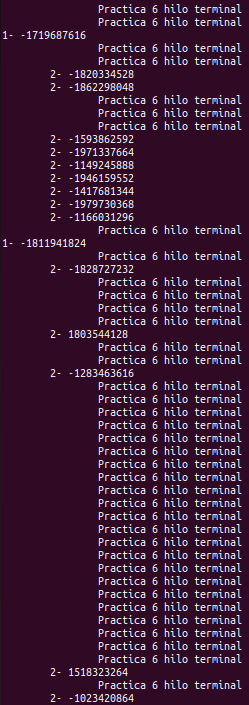
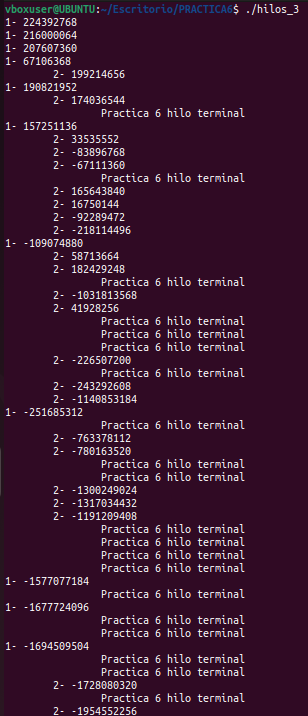


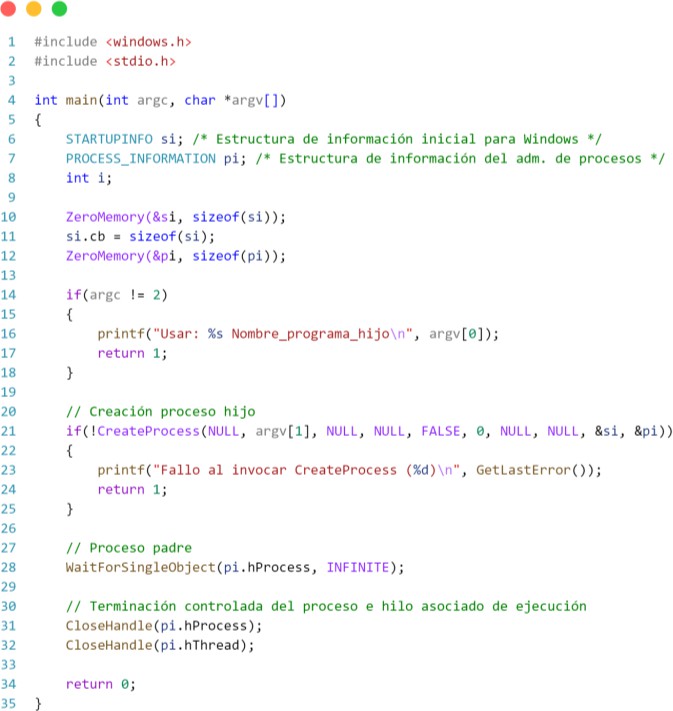


*6. Programe una aplicación (tanto en Linux como en Windows), que cree un proceso hijo a partir de un proceso padre, el hijo creado a su vez creará 20 hilos. A su vez cada uno de los 20 hilos creará 15 hilos más. A su vez cada uno de los 15 hilos creará 10 hilos más. Cada uno de los hilos creados imprimirá en pantalla “Práctica 6 Hilo Terminal” si se trata de un hilo terminal o los identificadores de los hilos creados si se trata de un proceso o hilo padre.*



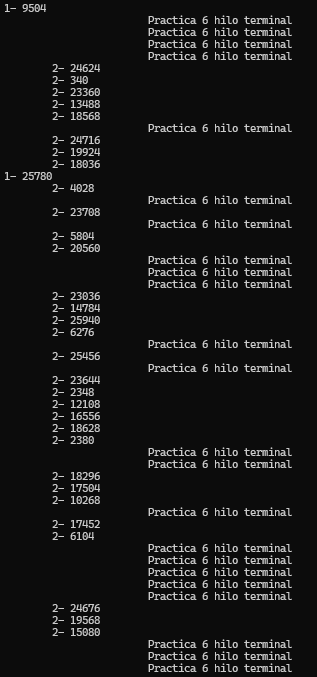
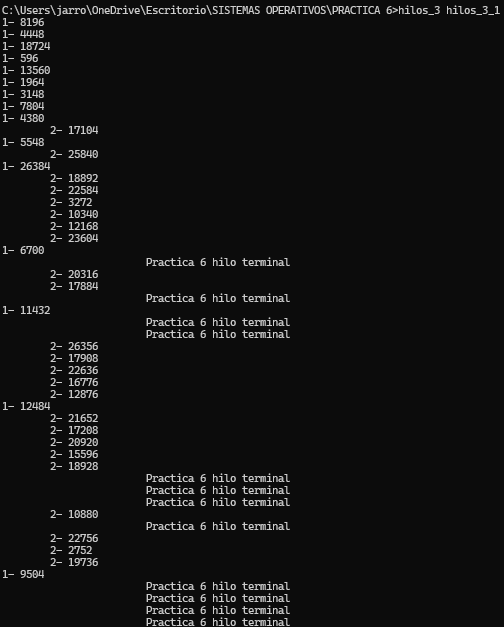
#### Salida recortada (extensión completa muy grande):

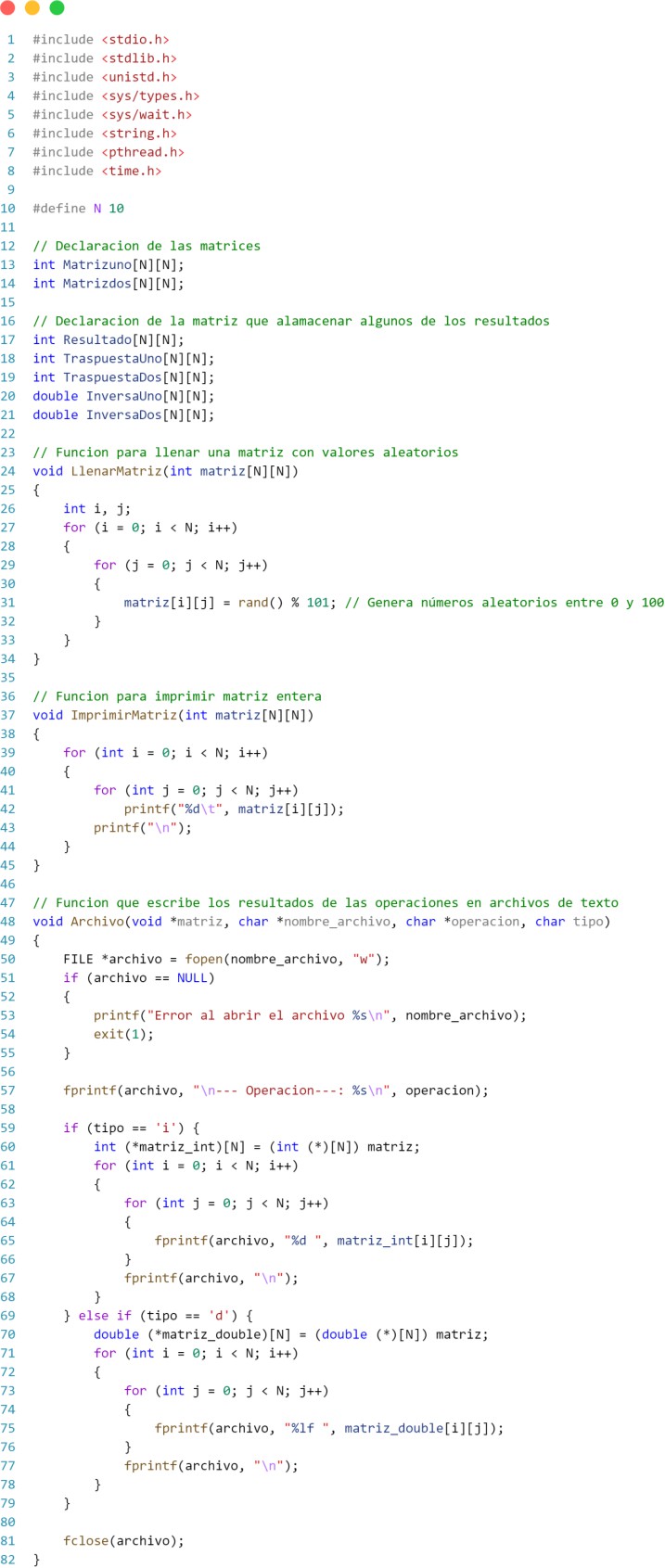


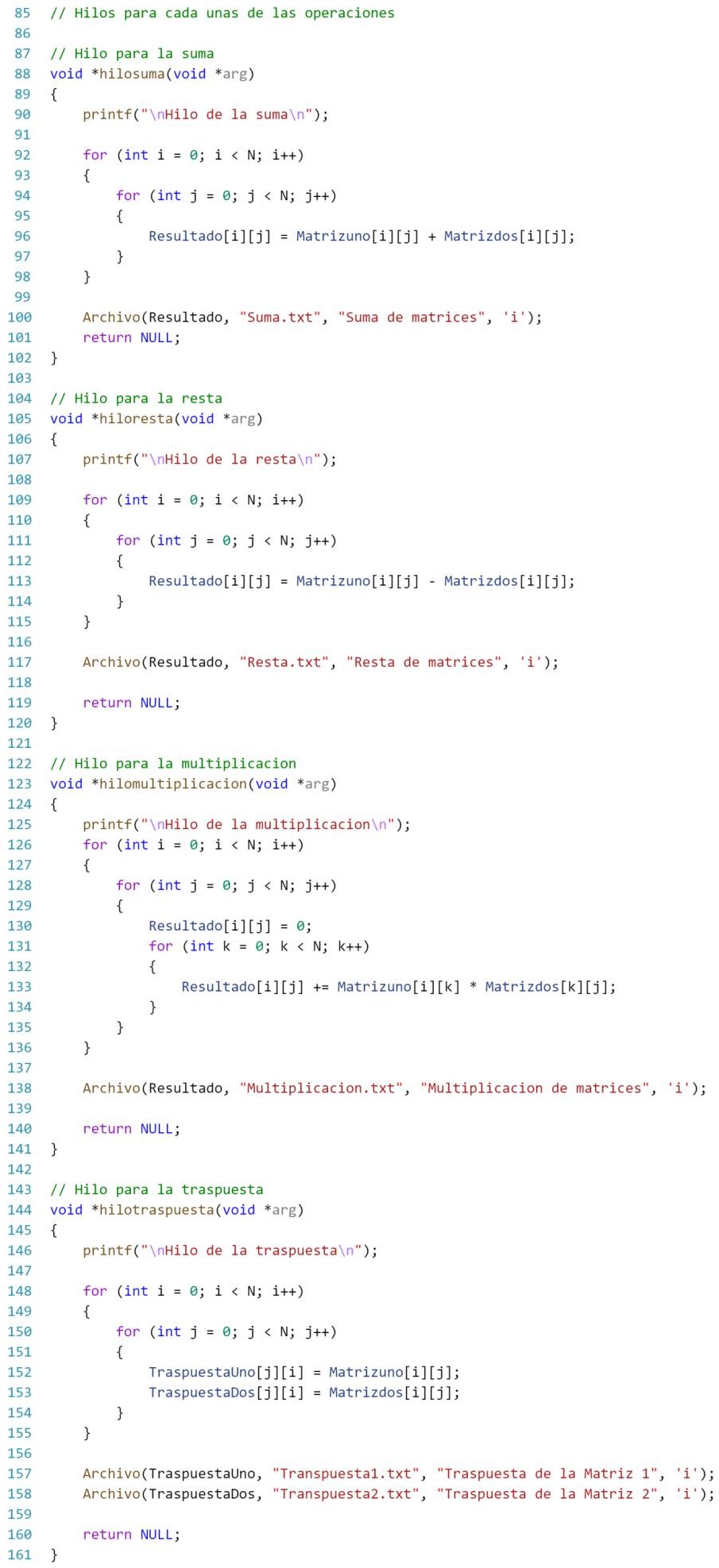


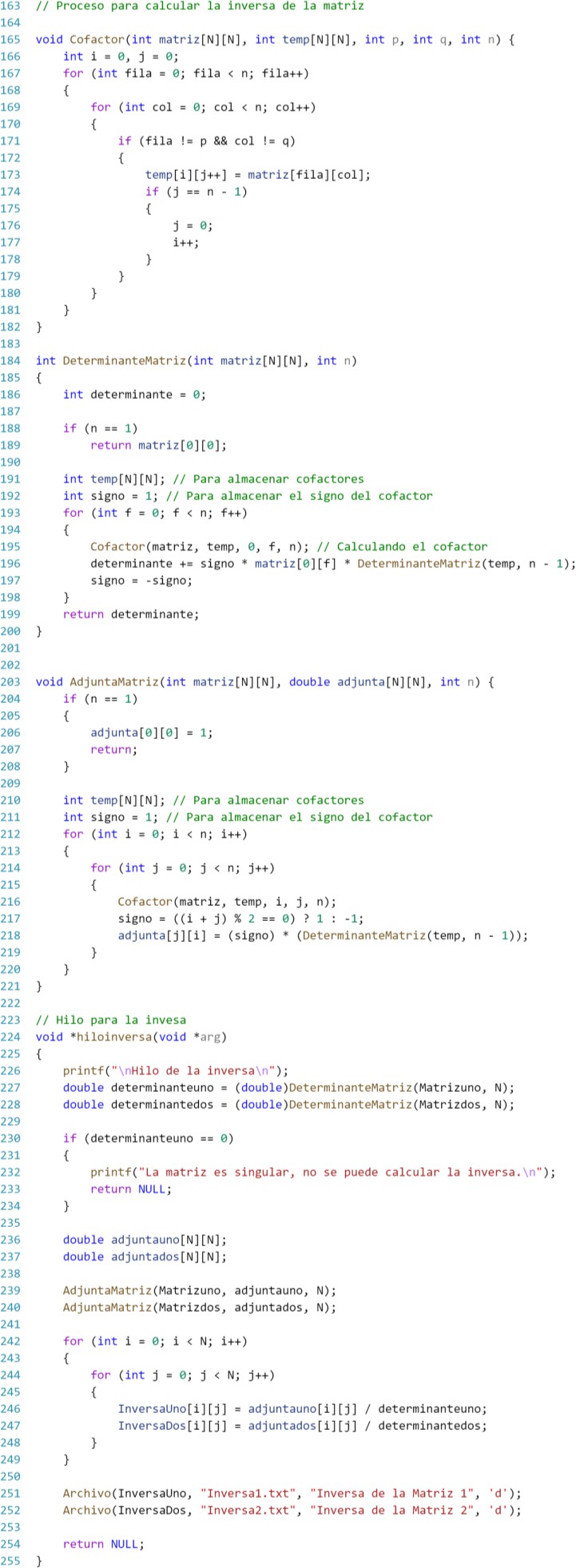


***Salida recortada (extensión completa muy grande):***

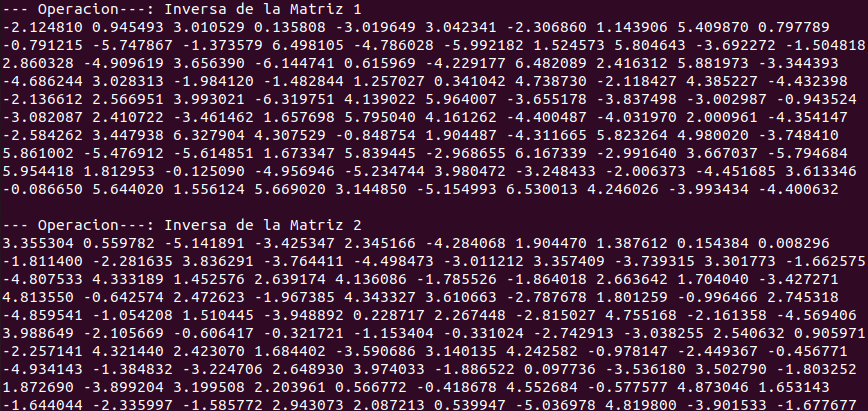
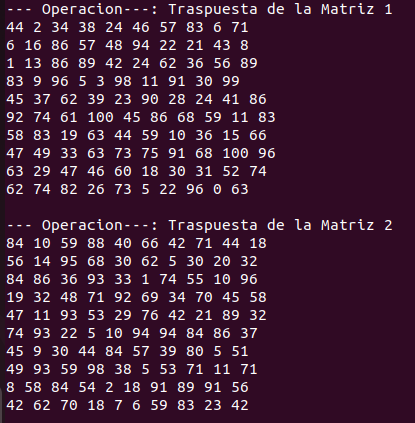
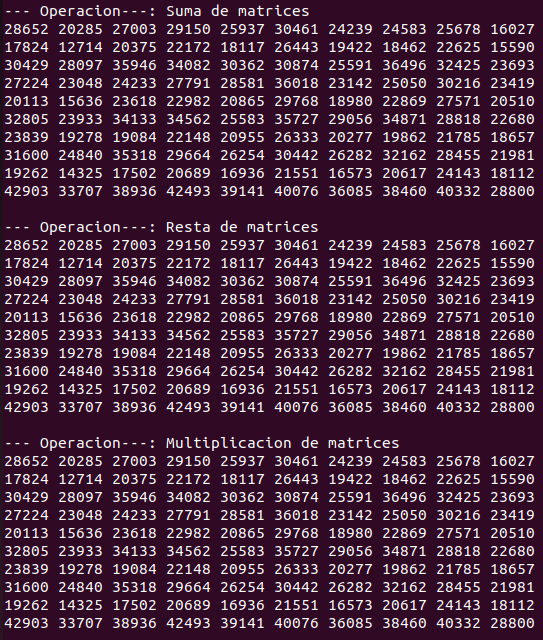
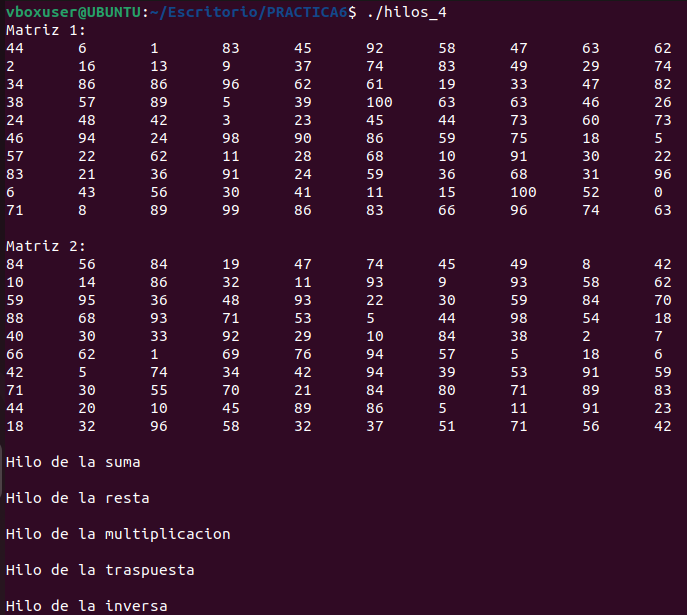


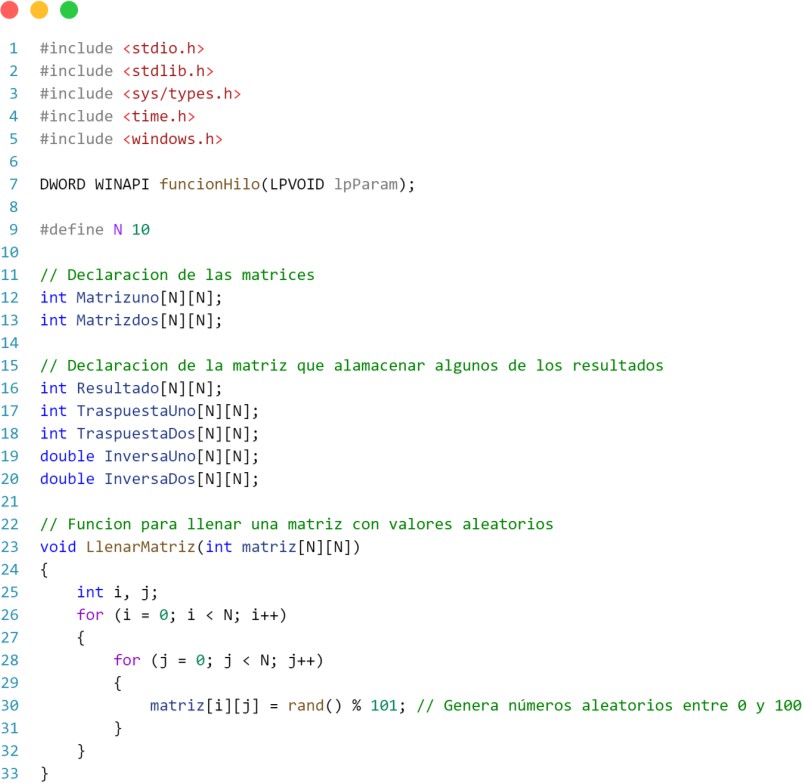
*7.- Programe la misma aplicación del punto 5 de la práctica 5 en su sección de Linux pero utilizando hilos (tanto en Linux como en Windows) en vez de procesos, obtenga el tiempo de ejecución del programa con hilos. Compare ambos programas en su tiempo de ejecución (el creado en la práctica 5 y el creado en esta práctica), y dé sus observaciones tanto de funcionamiento como de los tiempos de ejecución resultantes.*

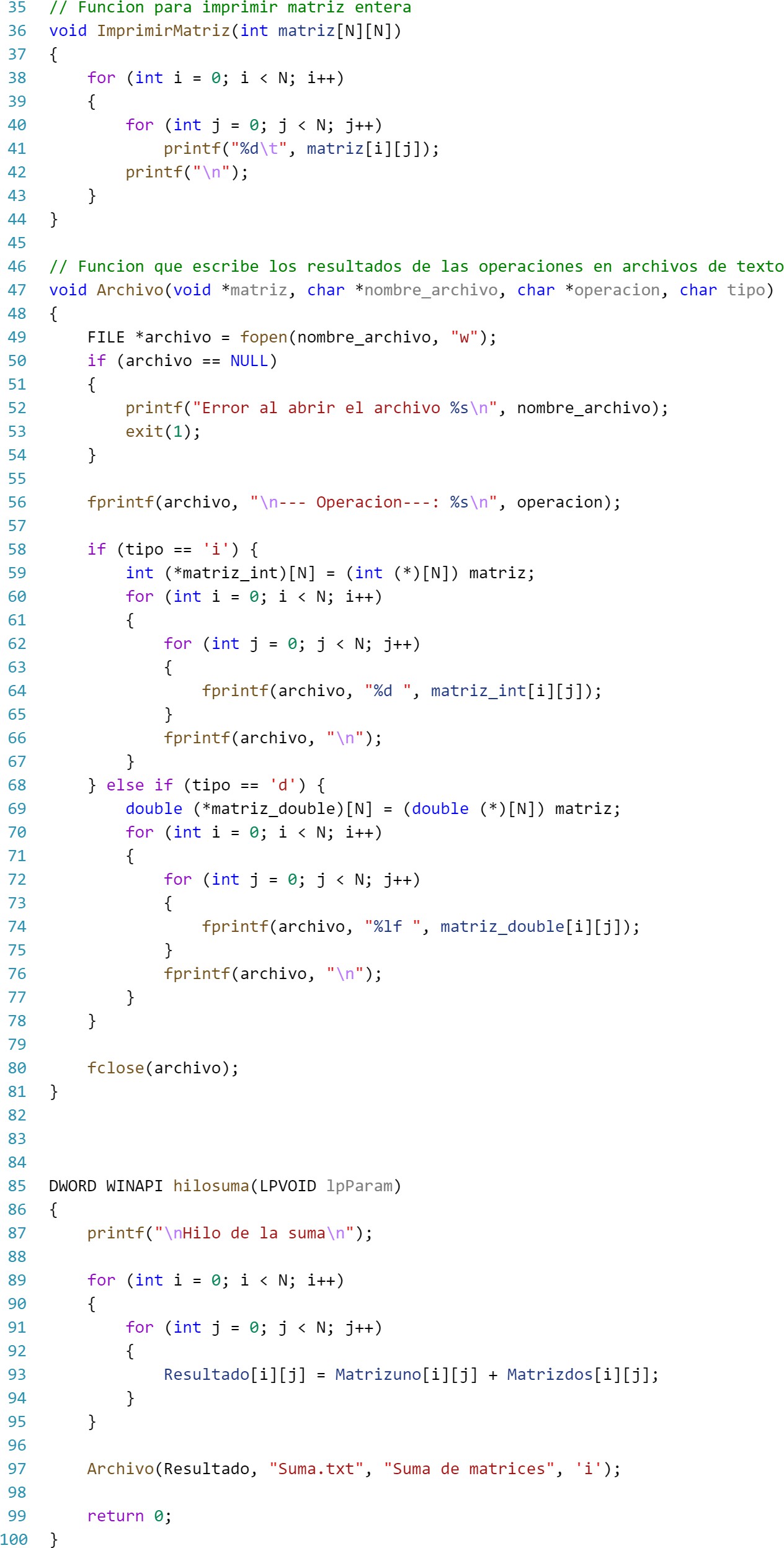




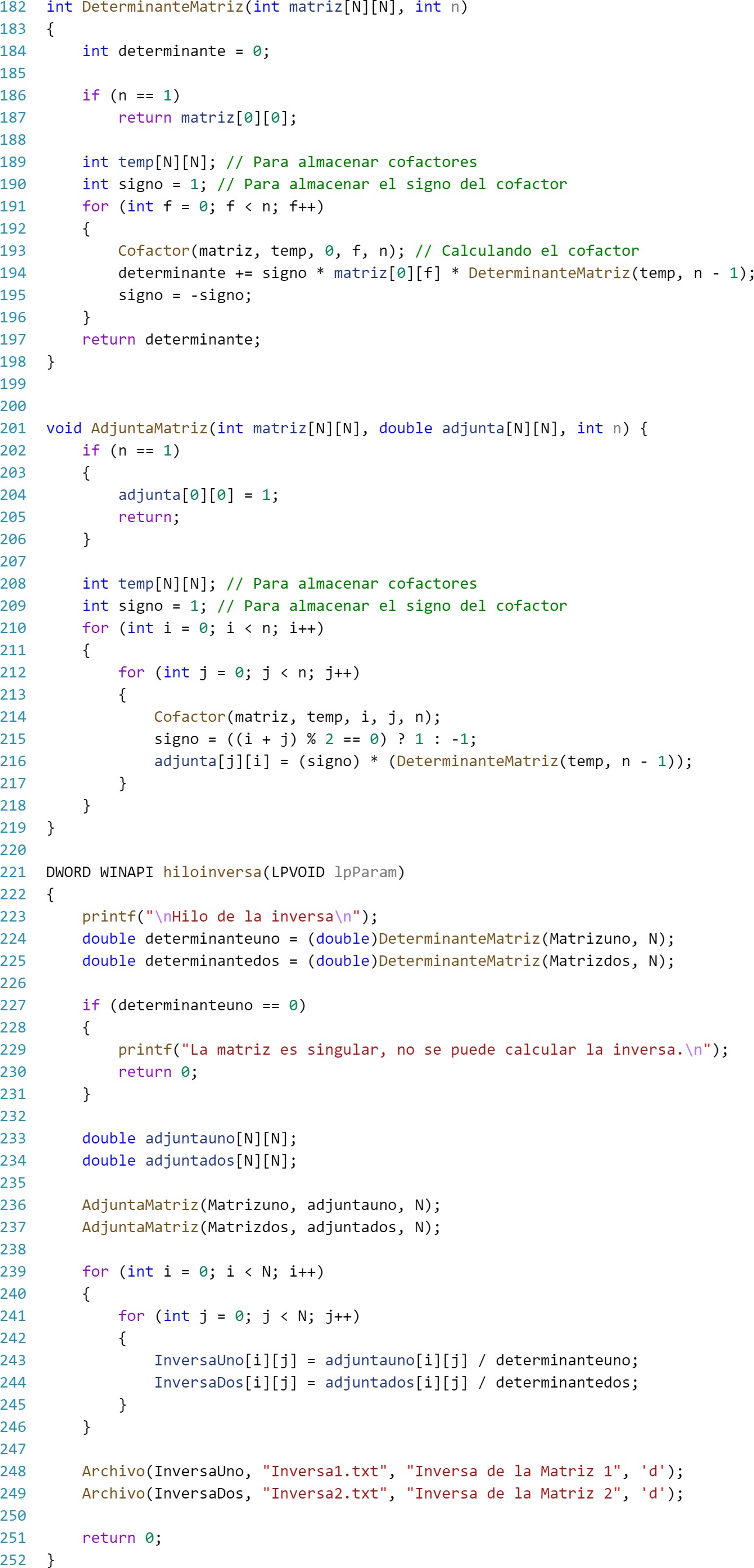


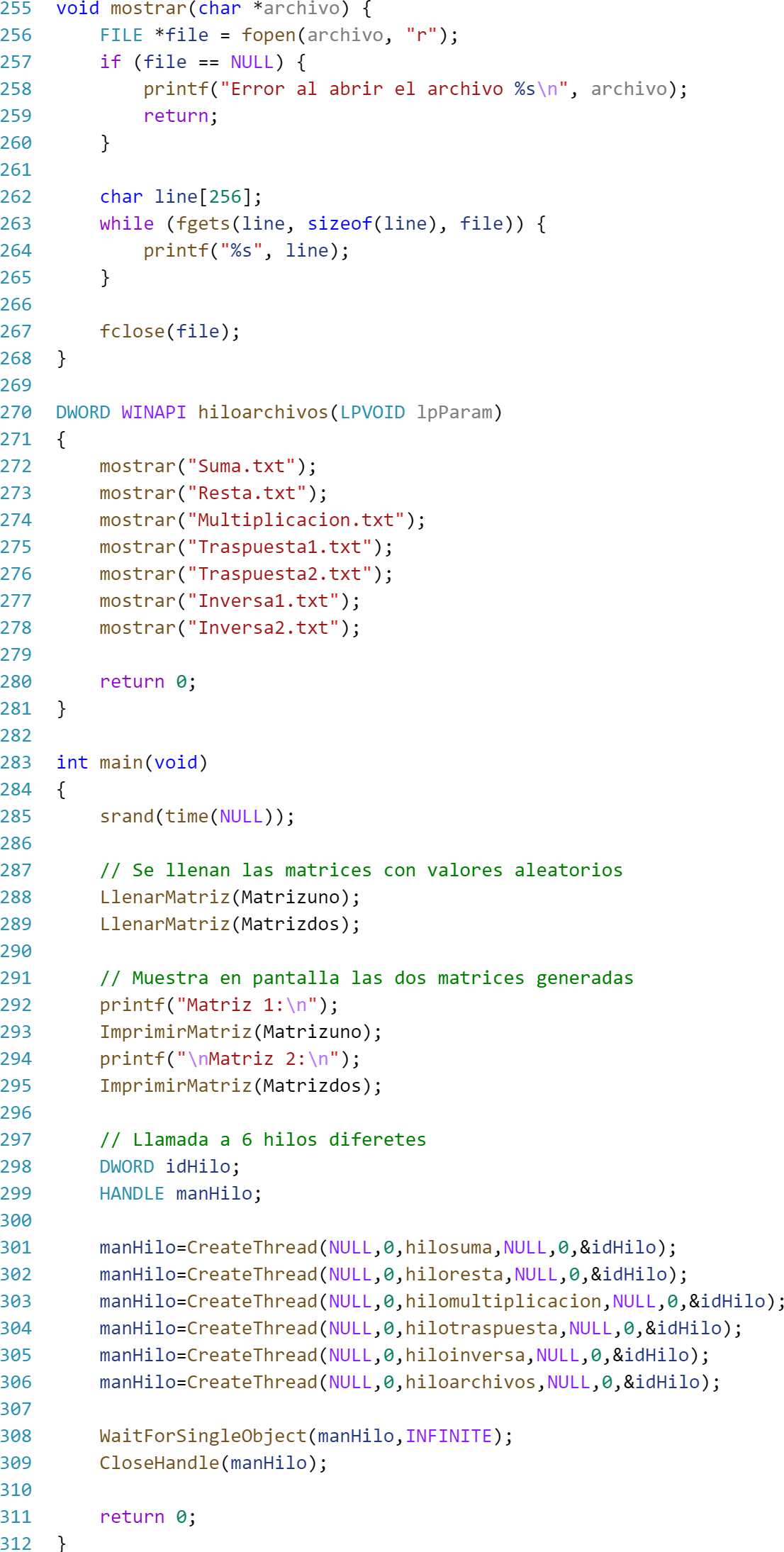


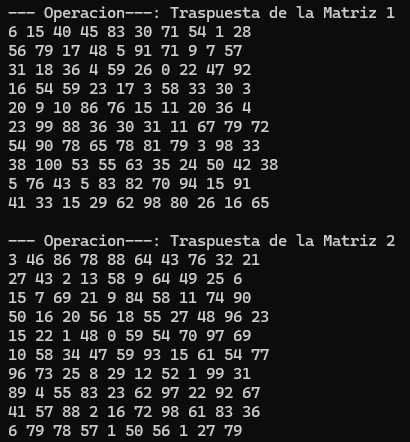
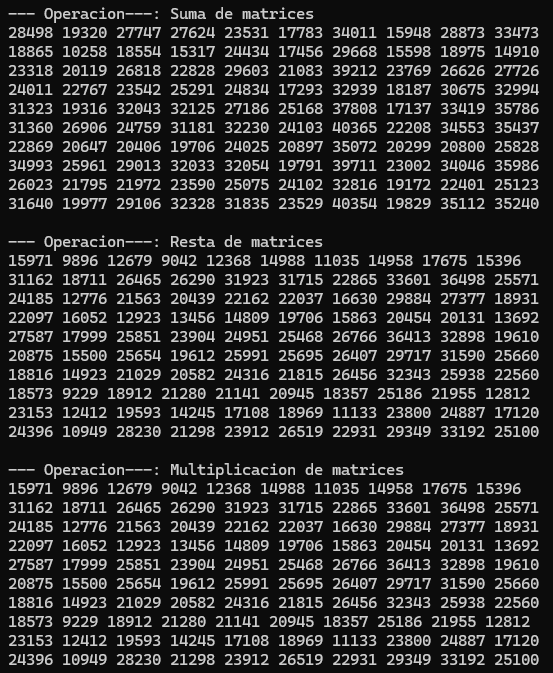
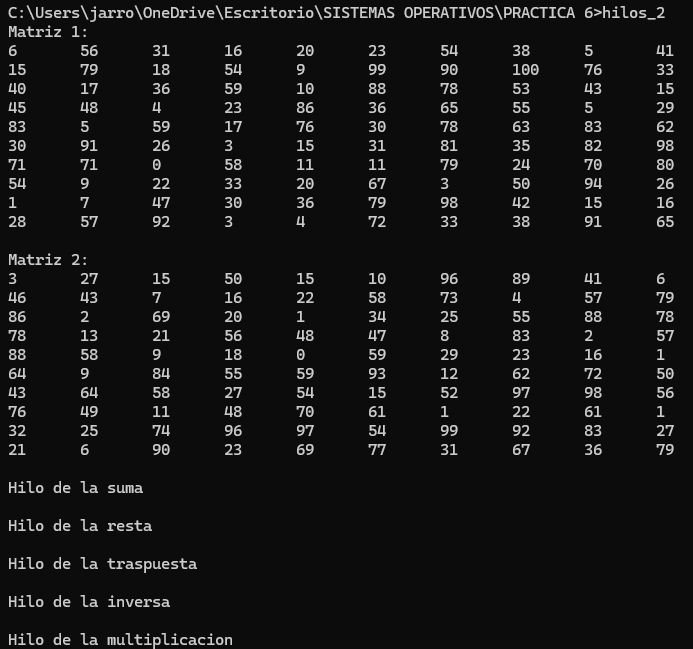


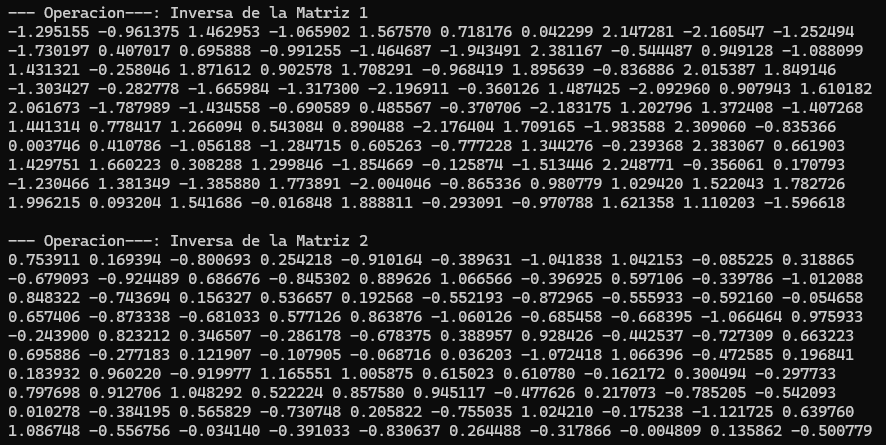








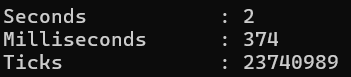




*Comparación y observaciones:*

Practica 5 Practica 6

Windows



Linux



Los programas realizados con hilos en lugar de procesos, tanto en Linux como en Windows, fueron significativamente más rápidos. Su tiempo de ejecución fue mucho menor debido a que los hilos comparten el mismo espacio de direcciones dentro de un proceso, lo que reduce la sobrecarga de creación y destrucción en comparación con los procesos.

Además, la comunicación y sincronización entre hilos es más eficiente, ya que no requieren mecanismos de comunicación interprocesos más lentos. Los hilos también consumen menos recursos del sistema, permitiendo gestionar un mayor número de ellos de manera más eficiente.

En nuestras pruebas, los programas multihilo completaron sus tareas mucho más rápido que los programas multiproceso, especialmente en escenarios con muchas tareas concurrentes. La eficiencia en la creación, terminación y comunicación de hilos contribuyó significativamente a este mejor desempeño.

# CONCLUSION

Durante esta práctica, hemos tenido la oportunidad de experimentar con la programación concurrente y el manejo de archivos en sistemas operativos, tanto en Linux como en Windows. Trabajar con hilos ha sido especialmente interesante, ya que pudimos ver de primera mano cómo los hilos pueden hacer que nuestras aplicaciones sean mucho más rápidas y eficientes al permitir la ejecución de múltiples tareas al mismo tiempo.

Utilizar las funciones de la biblioteca pthread en Linux, como pthread\_create y pthread\_join, nos ha dado las herramientas necesarias para gestionar hilos de manera efectiva. También exploramos CreateThread en Windows, que, aunque es un poco diferente, cumple la misma función básica de permitirnos trabajar con hilos en nuestras aplicaciones.

Además, trabajar con funciones como scandir y stat nos ha enseñado cómo interactuar con el sistema de archivos de manera más profunda. Estas funciones nos permiten listar y obtener información detallada sobre los archivos y directorios, algo que es fundamental para cualquier aplicación que necesite gestionar grandes cantidades de datos.

Una de las partes más interesantes de la práctica fue comparar el rendimiento entre programas que usan procesos y aquellos que usan hilos. Descubrimos que los hilos son generalmente más rápidos y consumen menos recursos que los procesos, lo cual es un conocimiento valioso para optimizar nuestras aplicaciones en el futuro.