

ESCUELA SUPERIOR DE COMPUTO

INSTITUTO POLITÉCNICO NACIONAL

PRACT ICA 6

**Mecanismos de sincronización de procesos en Linux**

**y Windows (semáforos)**

S I S T E M A S O P E R A T I V O S

**Profesor:** Cortes Galicia Jorge

**Grupo:** 2CM9

**Integrantes:**

Beltran García Juan

Hernández Méndez Oliver Manuel

López López Rodrigo

Rangel Lozada Kevin Sebastián

Índice

[1 Introducción Teórica 1](#_Toc58450868)

[2 Desarrollo Experimental 1](#_Toc58450869)

[2.1 Ejercicio 1 1](#_Toc58450870)

[2.2 Ejercicio 2 2](#_Toc58450871)

[2.3 Ejercicio 3 2](#_Toc58450872)

[2.4 Ejercicio 4 3](#_Toc58450873)

[2.4.1 Codificación en Linux 3](#_Toc58450874)

[2.4.2 Codificación en Windows 3](#_Toc58450875)

[3 Conclusiones 3](#_Toc58450876)

# Introducción Teórica

El principio fundamental es este: dos o más procesos pueden cooperar por medio de simples señales, tales que un proceso pueda ser obligado a parar en un lugar específico hasta que haya recibido una señal especifica. Cualquier requisito complejo de coordinación puede ser satisfecho con la estructura de señales apropiada. Para la señalización se utilizan unas variables especiales llamadas semáforos.

El concepto de semáforo nace de la necesidad de crear un sistema operativo en el que puedan trabajar procesos cooperantes. No es un mecanismo de comunicación sino de sincronización y son utilizados para controlar el acceso a los recursos.

Un semáforo básico es una variable entera y dos operaciones atómicas (sin interrupciones) que la manejan:

* Espera (P): Se usa cuando un proceso quiere acceder a un recurso compartido y puede ocurrir:
* Si la variable entera es positiva, coge el recurso y decrementa dicho valor.
* En caso de que el valor sea nulo el proceso se duerme y espera a ser despertado.
* Señal (V): Se utiliza para indicar que el recurso compartido esta libre y despertar a los procesos que estén esperando por el recurso.

Problemas que resuelven principalmente los semáforos:

* La exclusión mutua.
* Sincronización de Procesos

A cada semáforo se le supone asociada una cola de procesos, donde estarán los procesos que esperan a que el valor del mismo no sea cero. Normalmente, la planificación de esta cola de procesos es normalmente de tipo FIFO (First-in First-out). En la mayoría de los casos, éstas decisiones se toman en el proceso de diseño del Sistema Operativo, ya que los semáforos son un mecanismo de bajo nivel, gestionado por el sistema operativo.

Un semáforo puede tomar valores enteros no negativos ( esto es, el valor 0 o un valor entero positivo). La semántica de estos valores es: 0 semáforo cerrado, y >0 semáforo abierto.

En función del rango de valores positivos que admiten, los semáforos se pueden

clasificar en:

* Semáforos binarios: Son aquellos que solo pueden tomar los valores 0 y 1.
* Semáforos generales: Son aquellos que pueden tomar cualquier valor no negativo.

Frecuentemente, el que un semáforo sea binario o general, no es función de su estructura interna sino de como el programador lo maneja.

# Desarrollo Experimental

## Ejercicio 1

A través de la ayuda en línea que proporciona Linux, investigue el funcionamiento de las funciones: **semget(), semop().** Explique los argumentos, retorno de las funciones y las estructuras y uniones relacionadas con dichas funciones

**Función semget()**

La función semget devuelve el identificador del semáforo correspondiente a la clave key. Puede ser un semáforo ya existente, o bien semget crea uno nuevo si se da alguno de estos casos:

1. key vale IPC\_PRIVATE. Este valor especial obliga a semget a crear un nuevo y único identificador, nunca devuelto por ulteriores llamadas a semget hasta que sea liberado con semctl.
2. key no está asociada a ningún semáforo existente, y se cumple que (semflg & IPC\_CREAT) es cierto.

A un semáforo puede accederse siempre que se tengan los permisos adecuados.

Si se crea un nuevo semáforo, el parámetro nsems indica cuántos semáforos contiene el conjunto creado; los 9 bits inferiores de semflg contienen los permisos estilo UNIX de acceso al semáforo (usuario, grupo, otros).

**Sintaxis:**

int semget ( key\_t key, int nsems, int semflg );

**Parámetros:**

**key**: Es la llave que indica a que grupo de semáforos queremos acceder. Se podrá obtener de una de las tres formas vistas en la introducción.

**nsems**: Es el número total de semáforos que forman el grupo devuelto por semget. Cada uno de los elementos dentro del grupo de semáforos puede ser referenciado por los números enteros desde 0 hasta nsems-1.

**semflg**: Es una máscara de bits que indica en qué modo se crea el semáforo, siendo:

• IPC CREAT Si este flag está activo, se creará el conjunto de semáforos, en caso de que no hayan sido ya creados.

• IPC EXCL Esta bandera se utiliza en conjunción con IPC CREAT, para lograr que semget de un error en el caso de que se intente crear un grupo de semáforos que ya exista.En este caso, semget devolvería -1 e inicializaría la variable errno con un valor EEXIST.

**Retorno:**

Si hubo éxito, el valor devuelto será el identificador del conjunto de semáforos (un entero no negativo), de otro modo, se devuelve -1 con errno indicando el error.

**Permisos del semáforo**: Los 9 bits menos significativos de semflg indican los permisos del semáforo. Sus posibles valores son:

* 0400 - Permiso de lectura para el usuario.
* 0200 - Permiso de modificación para el usuario.
* 0060 - Permiso de lectura y modificación para el grupo.
* 0006 - Permiso de lectura y modificación para el resto de los usuarios

Cabe destacar que el identificador del semáforo tiene asociado una estructura de datos llamada **semid\_ds** la cual esta definida de la siguiente manera:

struct semid\_ds {

struct ipc\_perm sem\_perm; /\* estructura permiso \*/

int \*pad; /\* usado por sistema \*/

ushort sem\_nsems; /\* número semáforos \*/

time\_t sem\_otime; /\* tiempo último semop \*/

time\_t sem\_ctime; /\* tiempo último cambio \*/

}

Una vez creada, la estructura de datos semid\_ds asociada al nuevo identificador de semáforo se inicializa de la siguiente manera:

* En la estructura de permisos de operación sem\_perm.cuid, sem\_perm.uid, sem\_perm.cgid y sem\_perm.gid se establecerán igual al ID de usuario efectivo y al ID de grupo efectivo, respectivamente, del proceso de llamada.
* Los 9 bits de orden inferior de sem\_perm.mode se establecerán igual a los 9 bits de orden inferior de semflg.
* La variable sem\_nsems se establecerá igual al valor de nsems.
* La variable sem\_otime debe establecerse igual a 0 y sem\_ctime debe establecerse igual a la hora actual.
* No es necesario inicializar la estructura de datos asociada a cada semáforo del conjunto. La función semctl() con el comando SETVAL o SETALL puede ser utilizada para inicializar cada semáforo.

**Función semop()**

Un semáforo se representa por una estructura anónima que incluye los siguientes miembros:

unsigned short semval; /\* valor del semáforo \*/

unsigned short semzcnt; /\* # esperando por cero \*/

unsigned short semncnt; /\* # esperando por incremento \*/

pid\_t sempid; /\* proceso que hizo la última operación \*/

La función semop realiza operaciones sobre los miembros seleccionados del conjunto de semáforos indicado por semid. Cada uno de los nsops elementos en el array apuntado por sops especifica una operación a ser realizada en un semáforo mediante una estructura sembuf que incluye los siguientes miembros:

unsigned short sem\_num; /\* número de semáforo \*/

short sem\_op; /\* operación sobre el semáforo \*/

short sem\_flg; /\* banderas o indicadores para la operación \*/

**Sintaxis:**

int semop (int semid, struct sembuf\* sops, unsigned nsops );

**Parámetros:**

**semid**: Identificador del grupo de semáforos sobre el que se van a realizar las operaciones atómicas.

**sops**: Es un puntero a un array de estructuras que indican las operaciones que se van a realizar sobre los semáforos.

**nsops:** Es el número total de elementos que tiene el array de operaciones.

**Retorno:**

Devuelve 0 en caso de éxito y -1 en caso de error, conteniendo la variable errno el código correspondiente.

En caso de error errno tendrá uno de los siguientes valores:

E2BIG -- El argumento nsops es mayor que SEMOPM, el numero máximo de operaciones

permitidas por llamada del sistema.

EACCES -- El proceso invocador no tiene permisos de acceso al semáforo como se requiere

por una de las operaciones especificadas.

EAGAIN -- Una operación no puede ser ejecutada inmediatamente y IPC\_NOWAIT ha sido

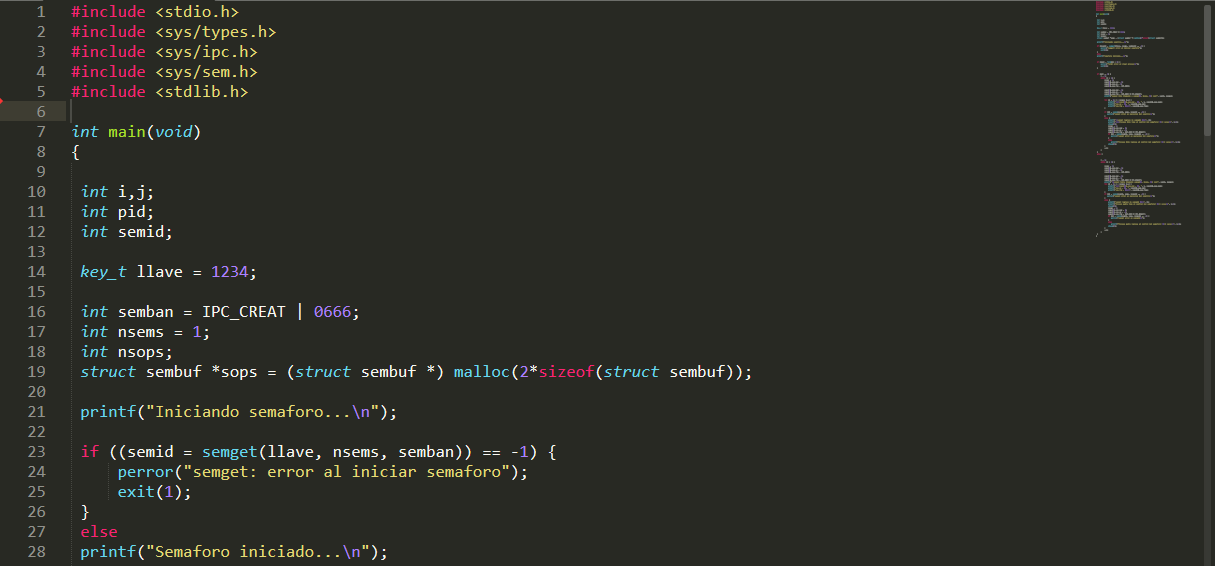
invocada en su sem\_flg.

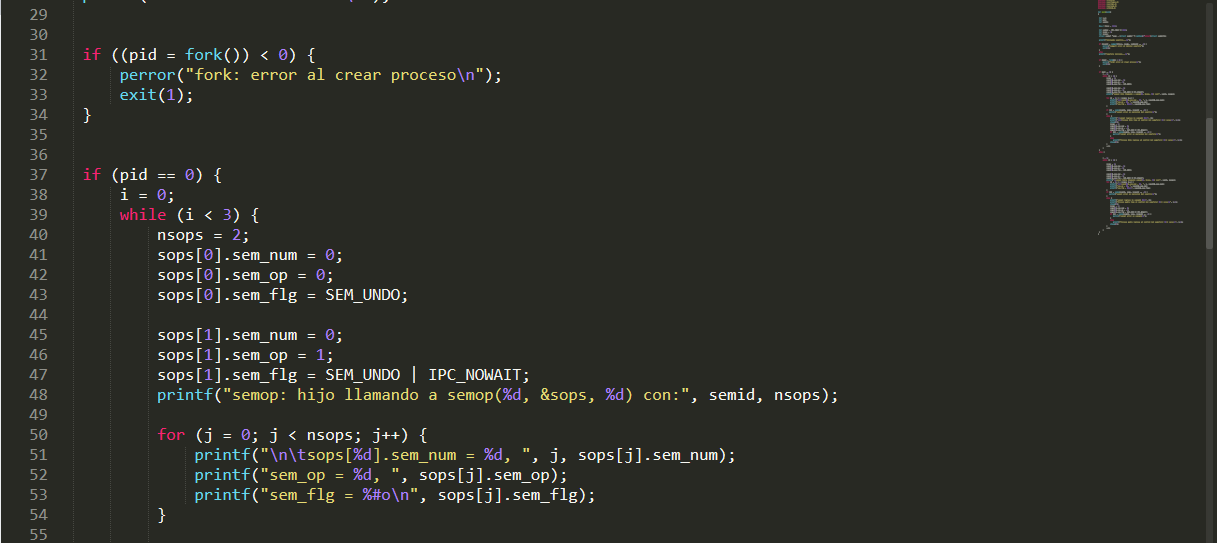
Los anteriores solo son algunos de los muchos códigos de error que nos puede arrojar errno

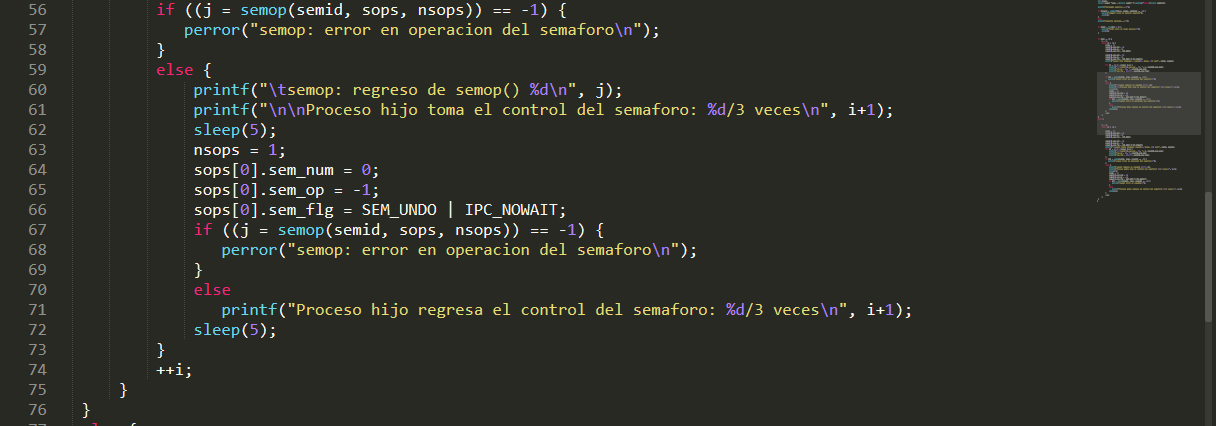
## Ejercicio 2

Capture, compile y ejecute el siguiente programa. Observe su funcionamiento y explique.

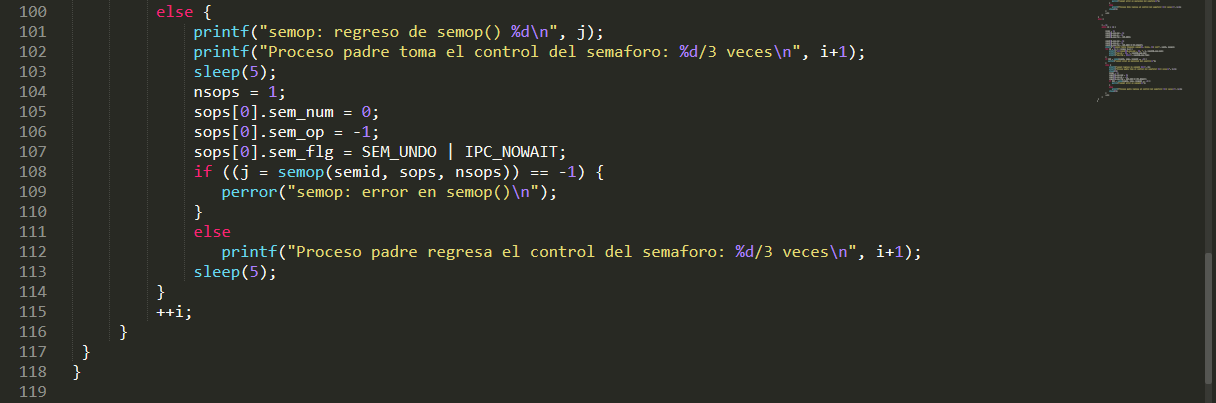
**Codigo capturado**



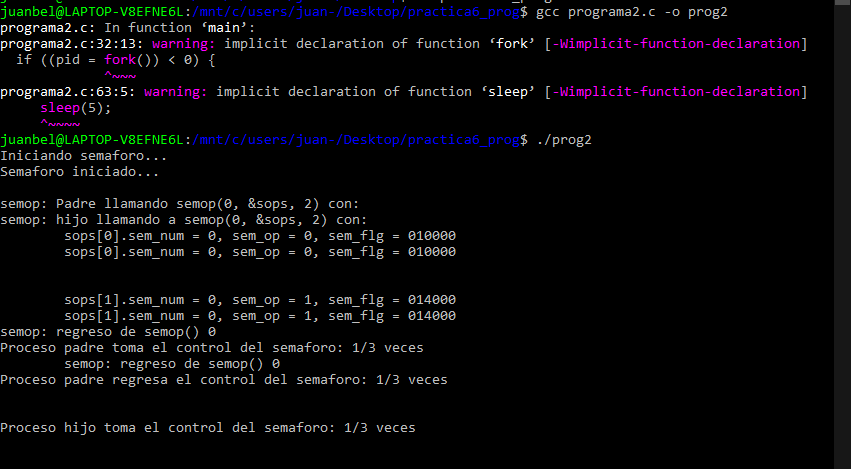


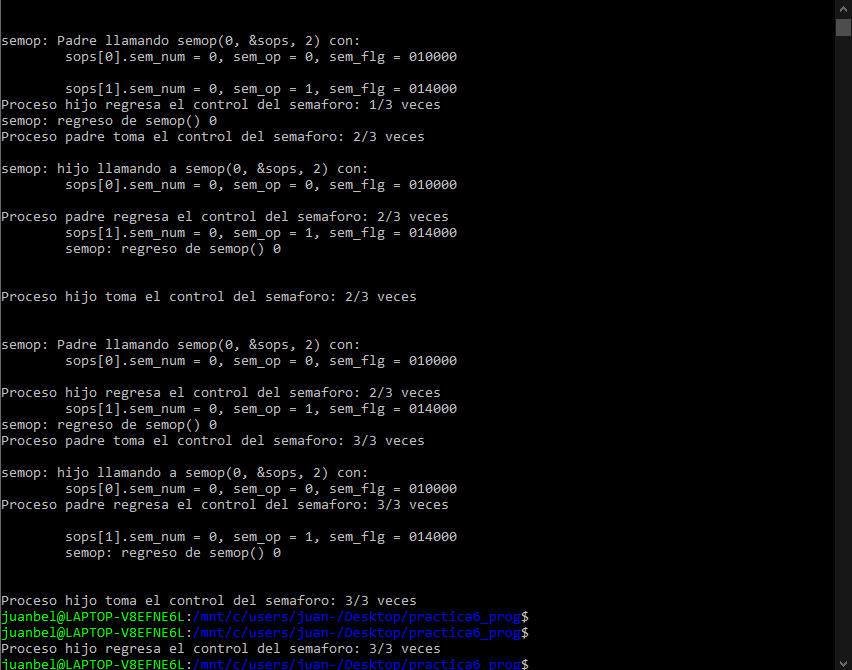






**Compilación y ejecución del programa**

****

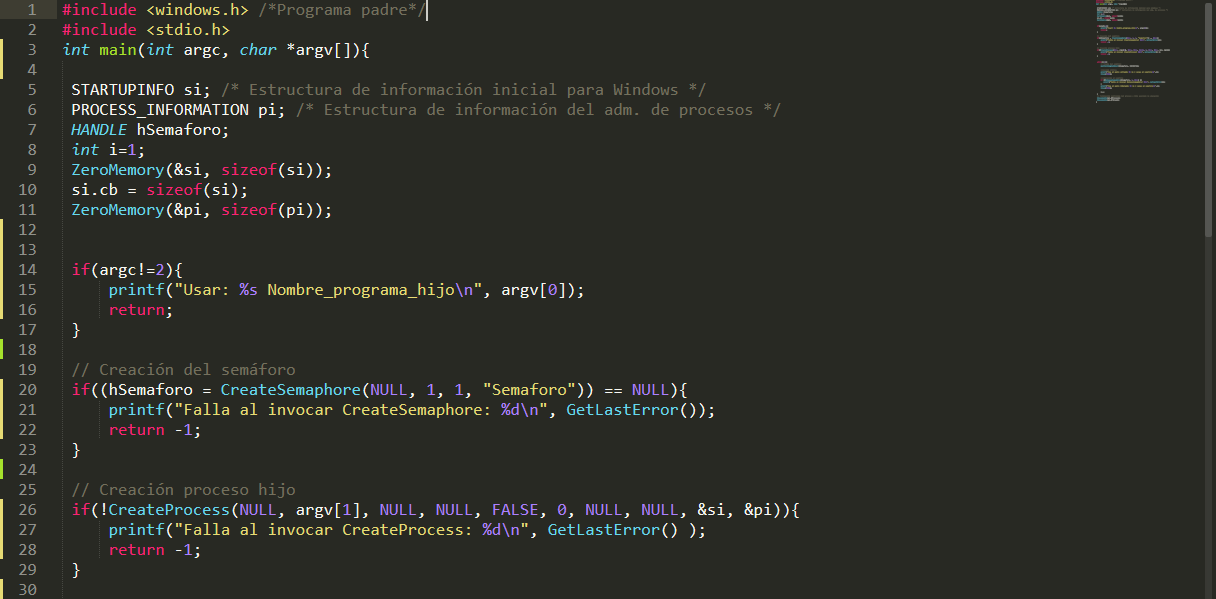
****

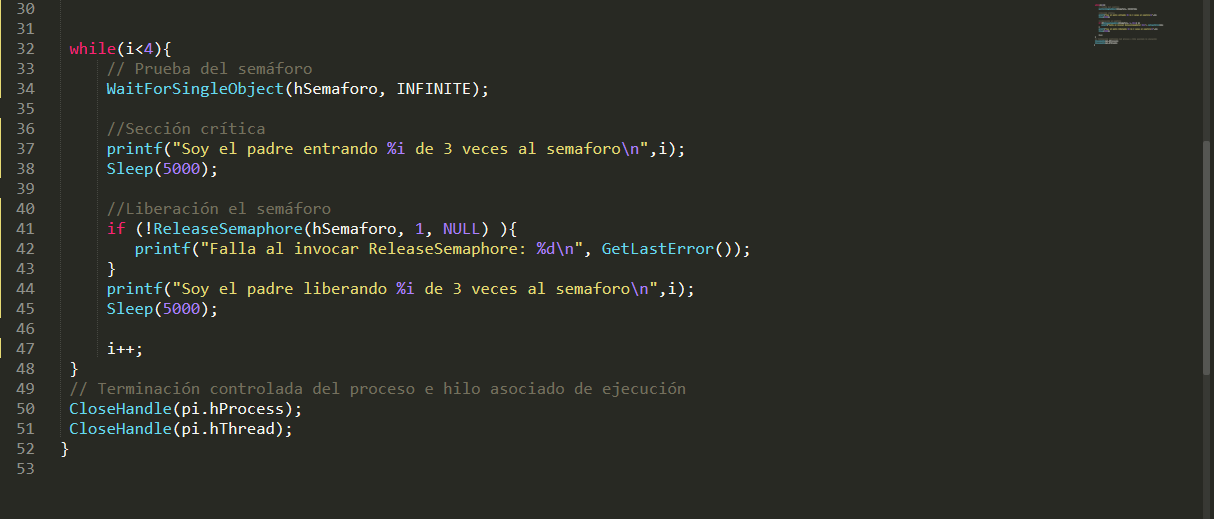
## Ejercicio 3

Capture, compile y ejecute los siguientes programas. Observe su funcionamiento. Ejecute de la siguiente forma: C:\>nombre\_programa\_padre nombre\_programa\_hijo

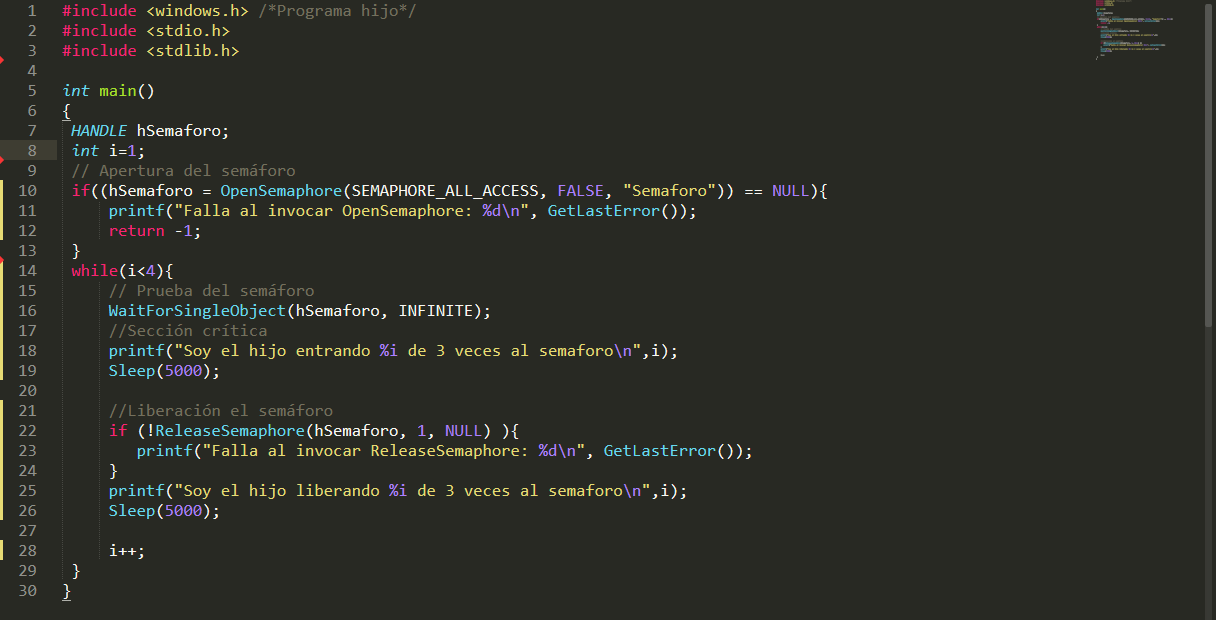
**Codigo capturado**

**Programa Padre**





**Programa Hijo**



**Compilación y ejecución del programa**



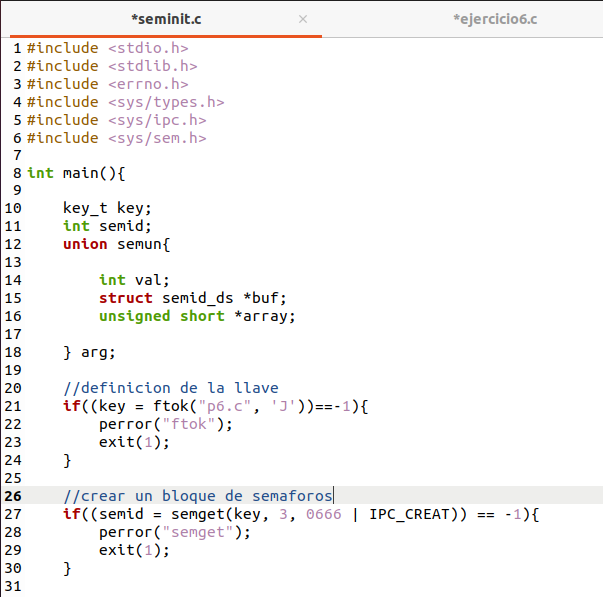
## Ejercicio 4

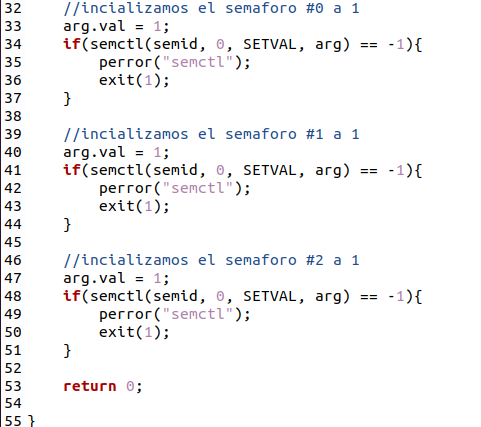
Programe la misma aplicación del punto 7 de la práctica 5 (tanto para Linux como para Windows), utilizando como máximo tres regiones de memoria compartida de 400 bytes cada una para almacenar todas las matrices requeridas por la aplicación. Utilice como mecanismo de sincronización los semáforos revisados en esta práctica tanto para la escritura y como para la lectura de las memorias compartidas. Úselos en los lugares donde haya necesidad de sincronizar el acceso a memoria compartida.

### Codificación en Linux

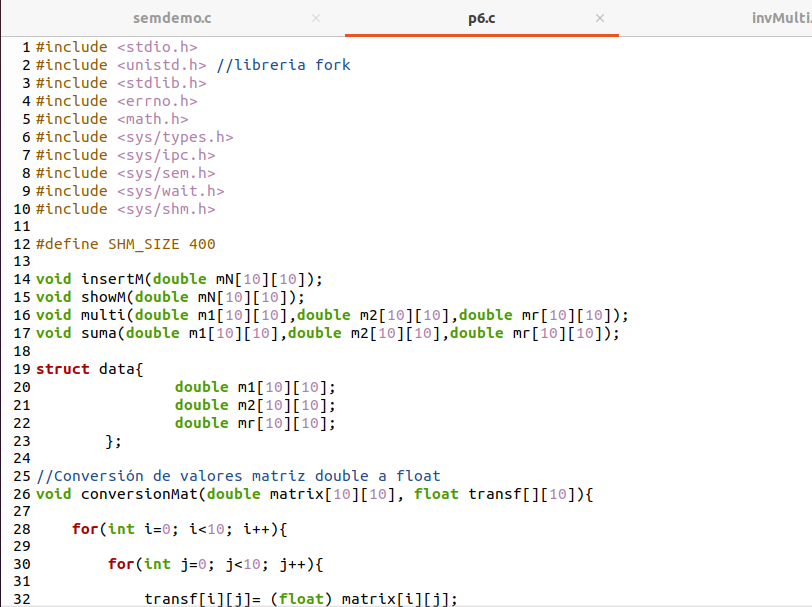
**Codigo fuente**

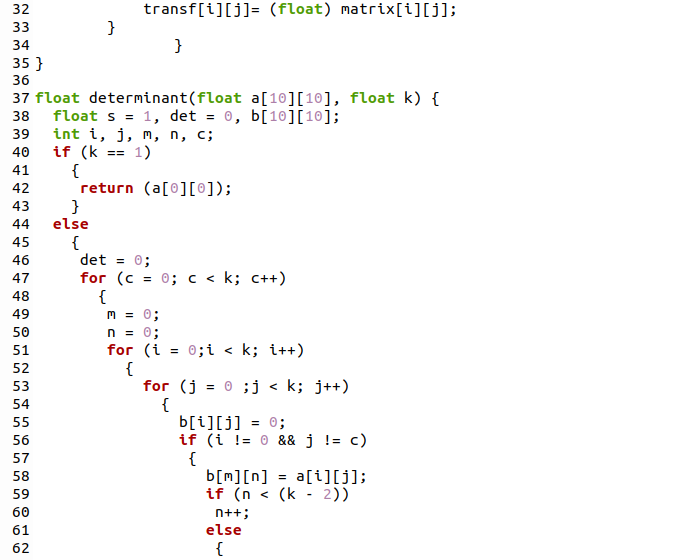
**Archivo: seminit.c (Definición del bloque de semáforos)**

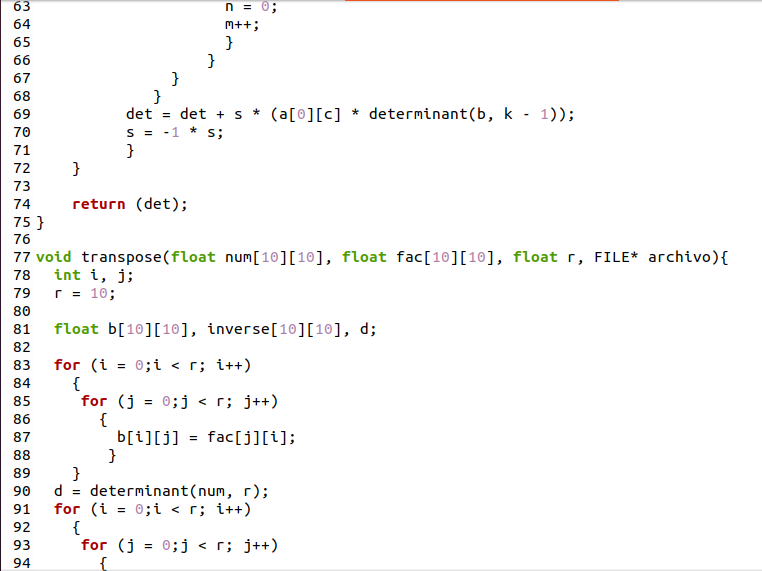


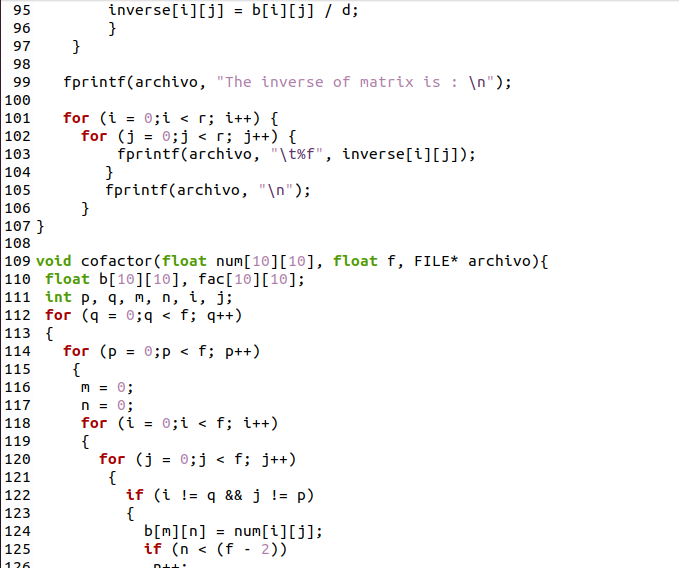


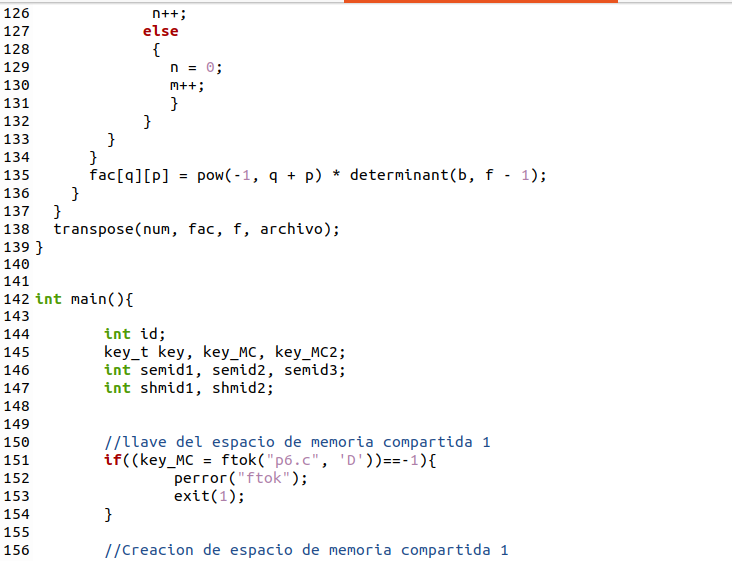
**Archivo: p6.c (Código principal)**

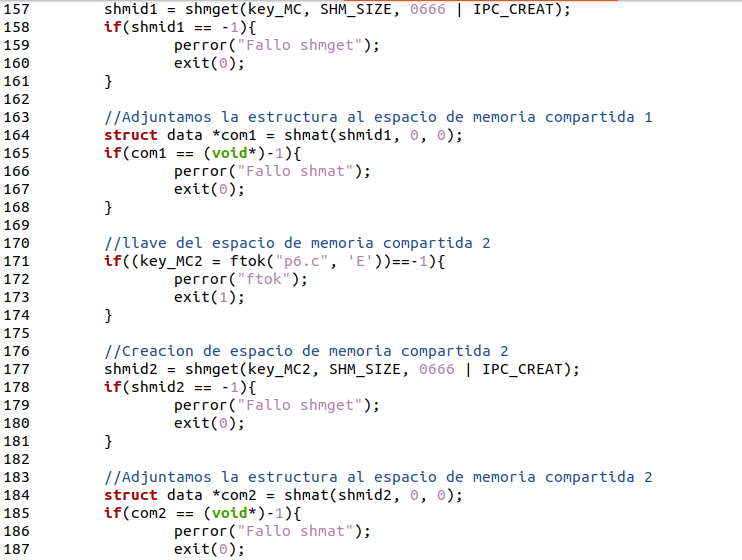


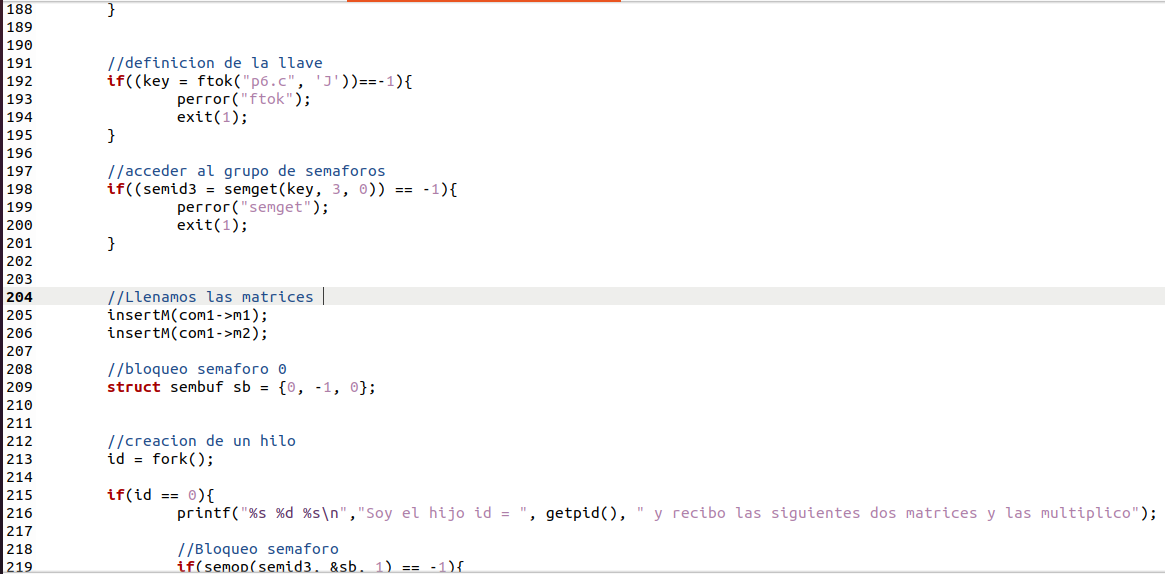




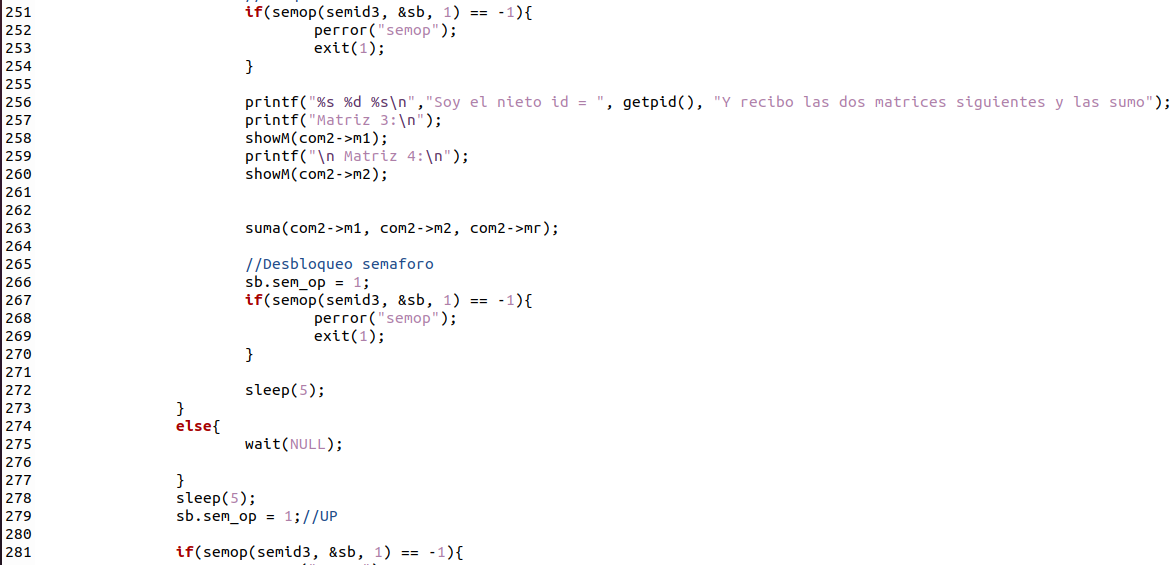


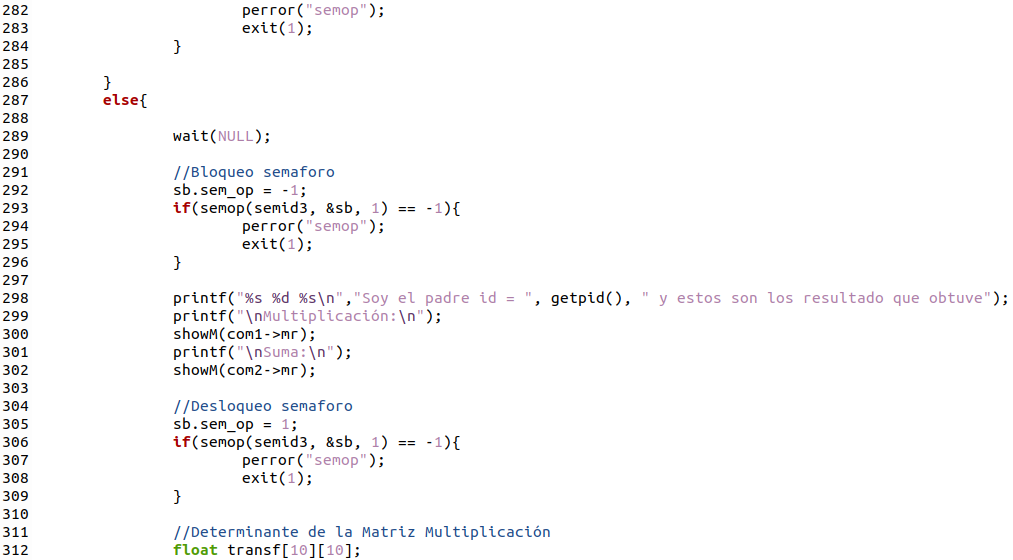


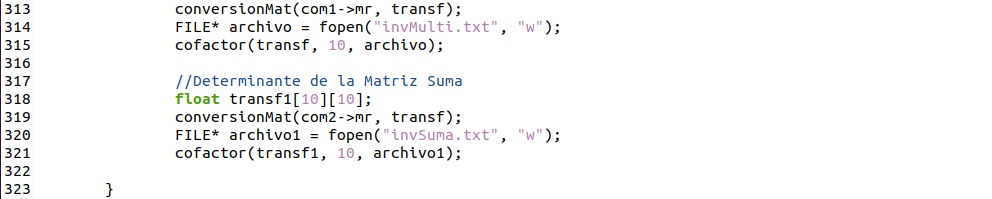


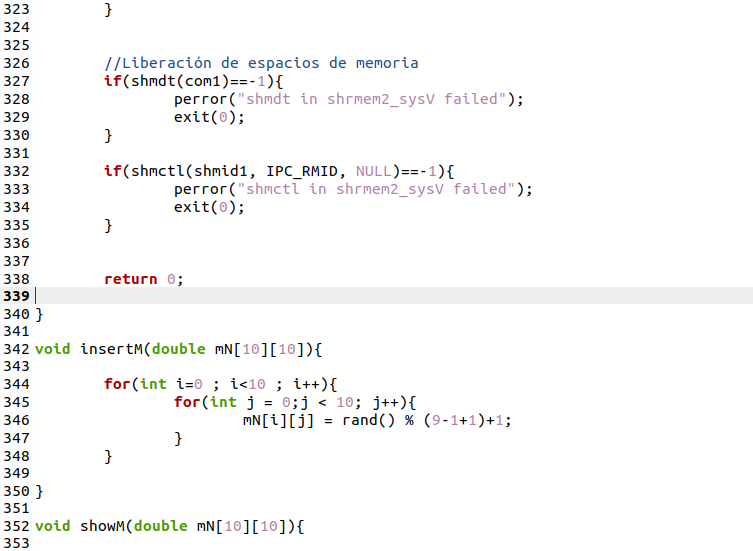


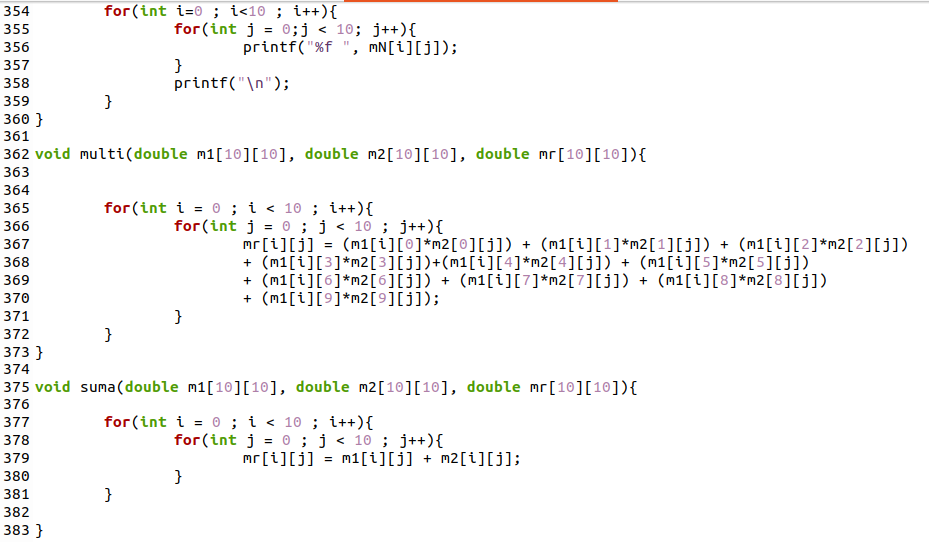




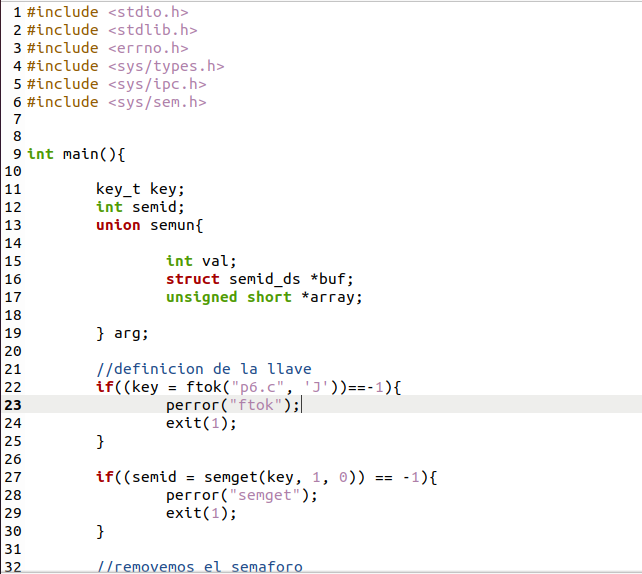


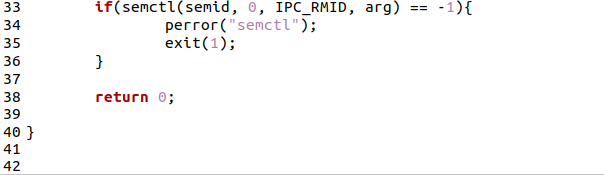






**Archivo: semrm.c (Borra semáforo)**





**Compilación y ejecución del programa**

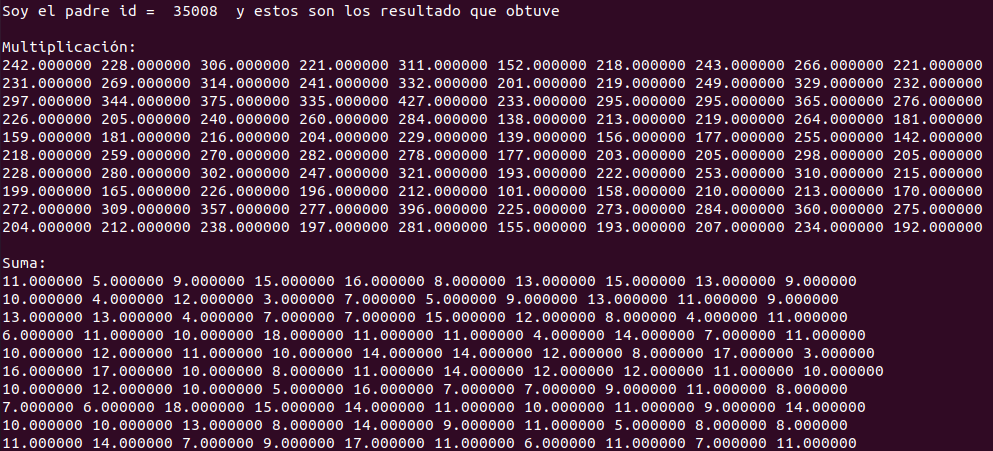
Primero ejecutamos el archivo que inicia los semáforos



Ejecución código principal



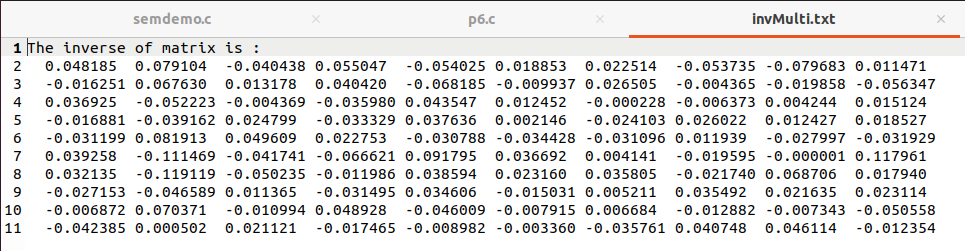




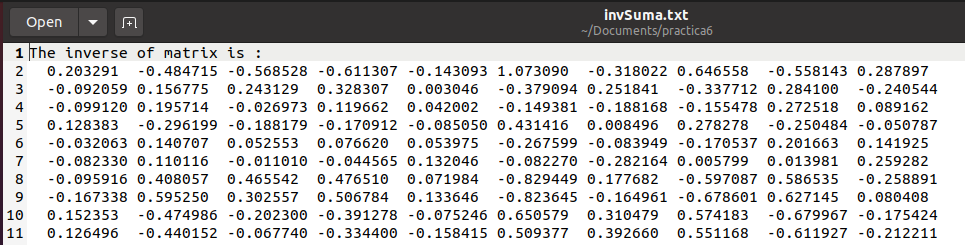
Archivos creados:



Inversa de la multiplicación



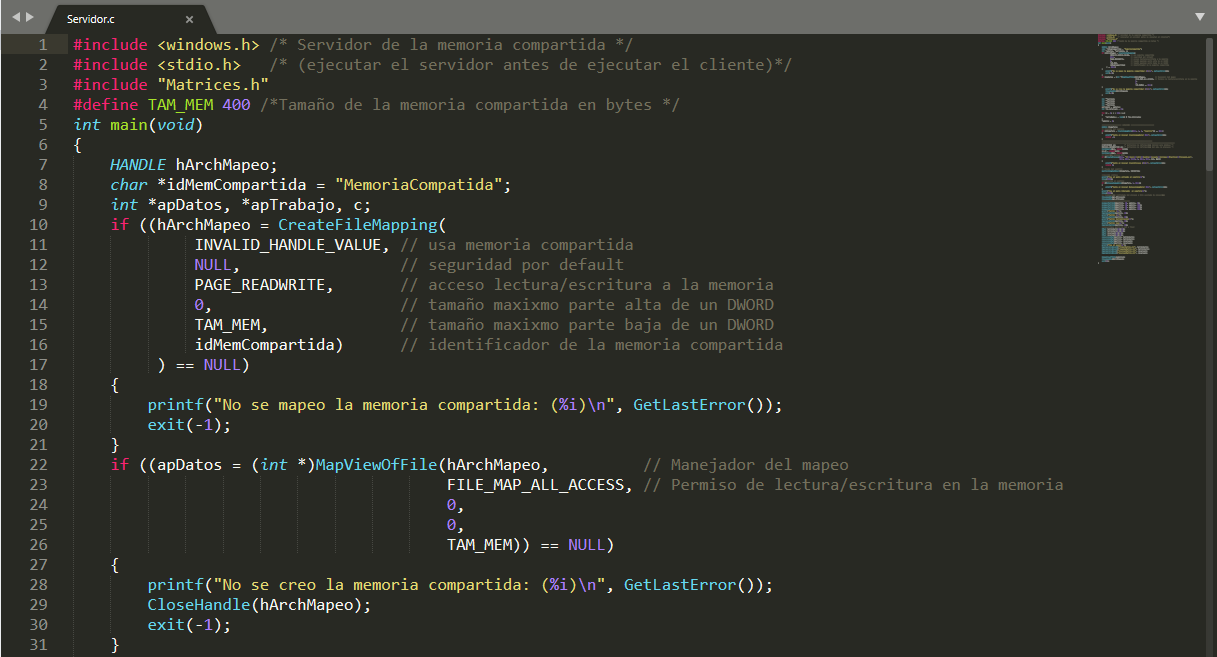
Inversa de la suma

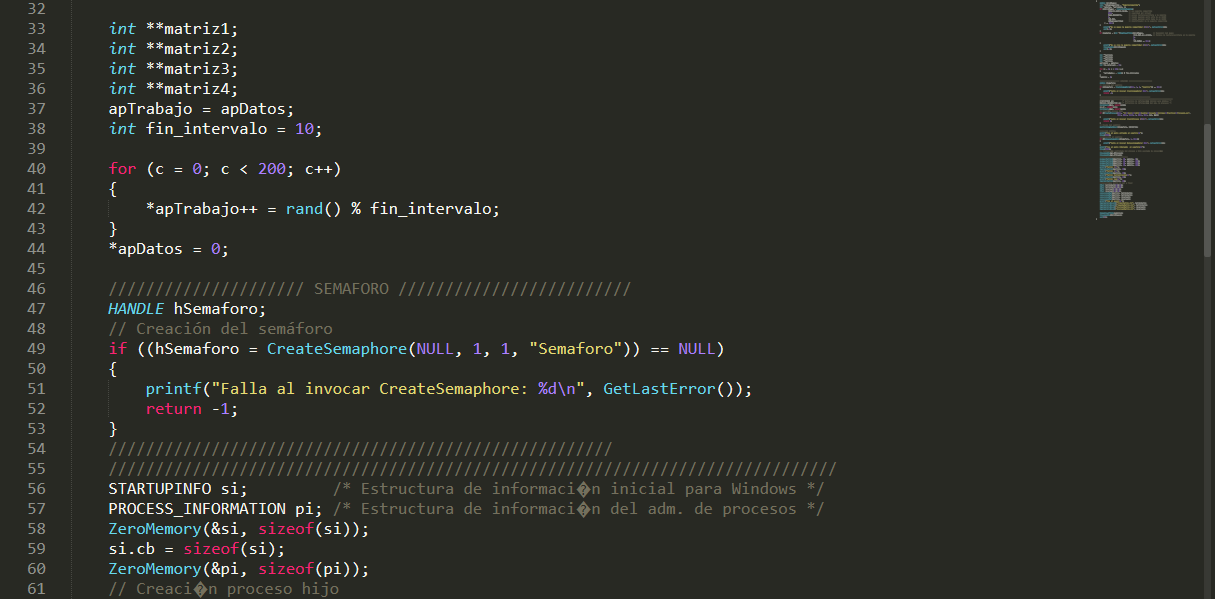


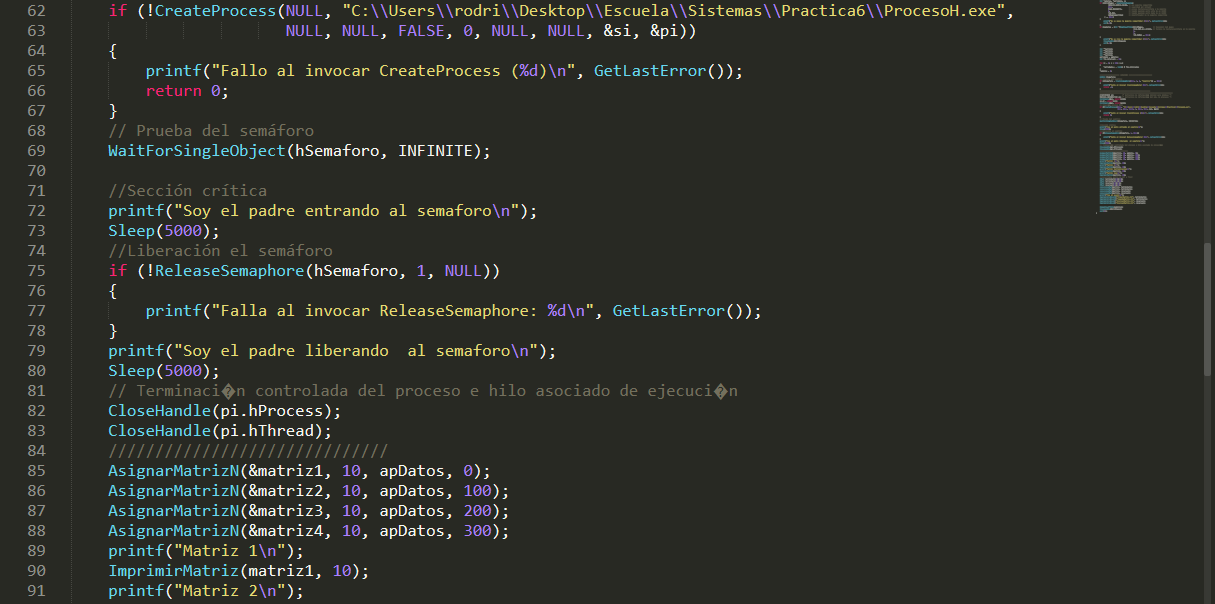
### Codificación en Windows

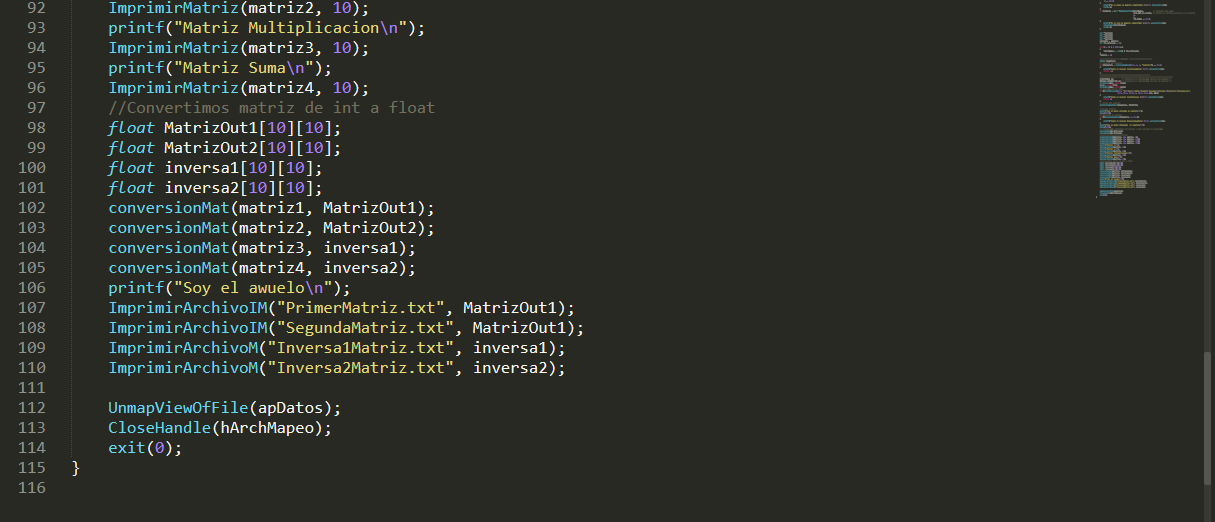
**Codigo fuente**

**Archivo: Servidor.c**

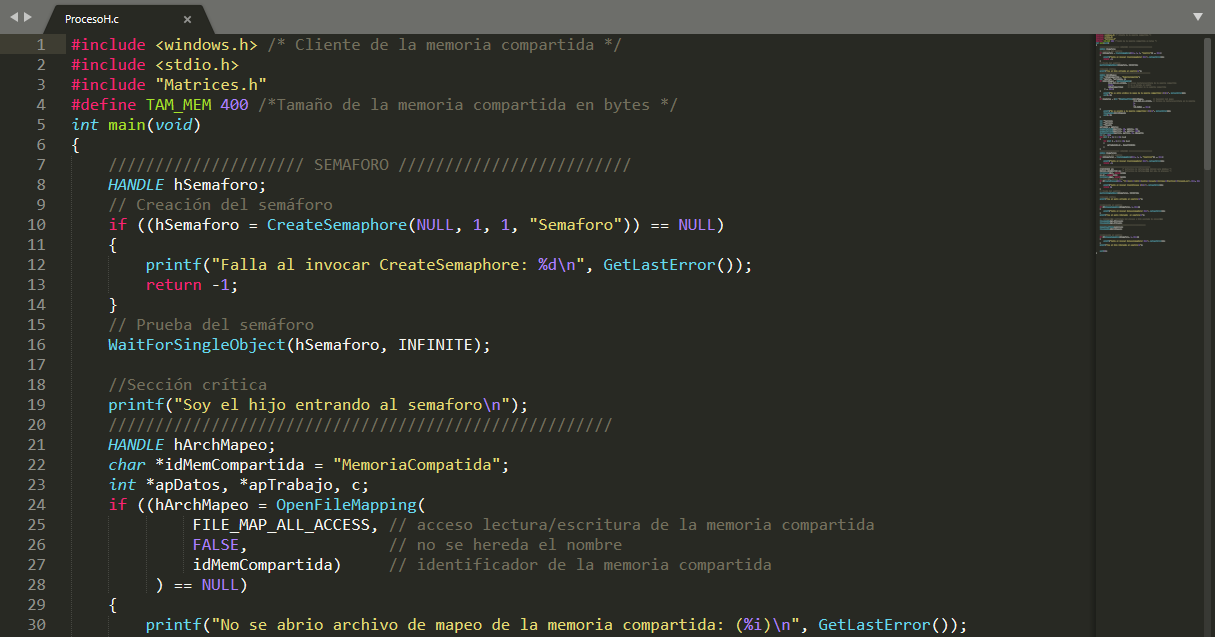


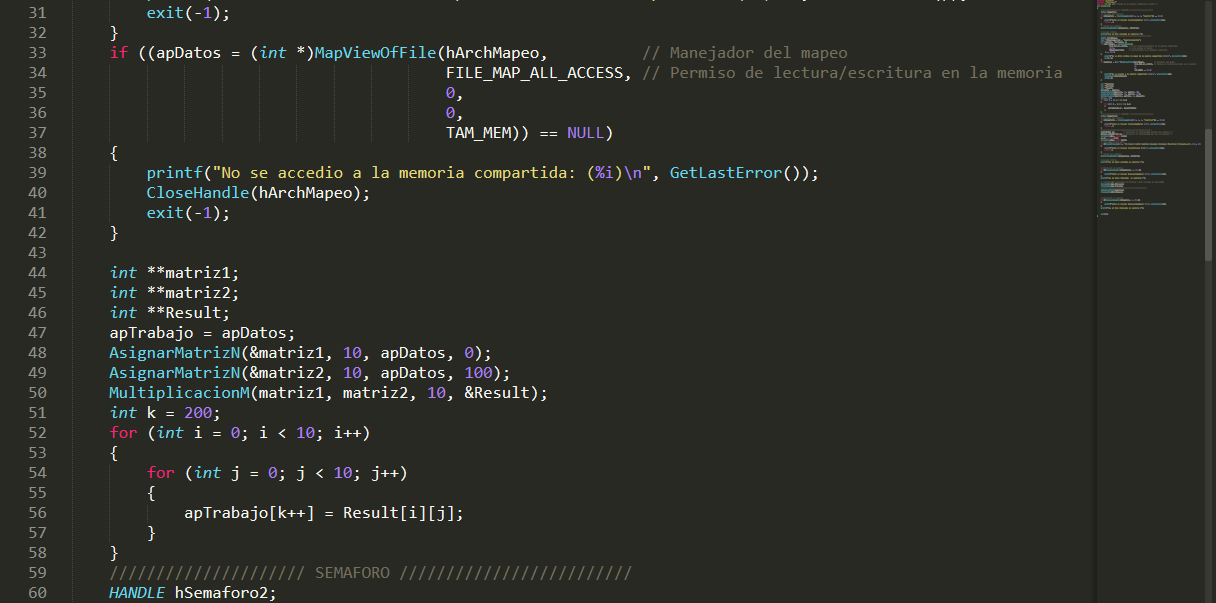


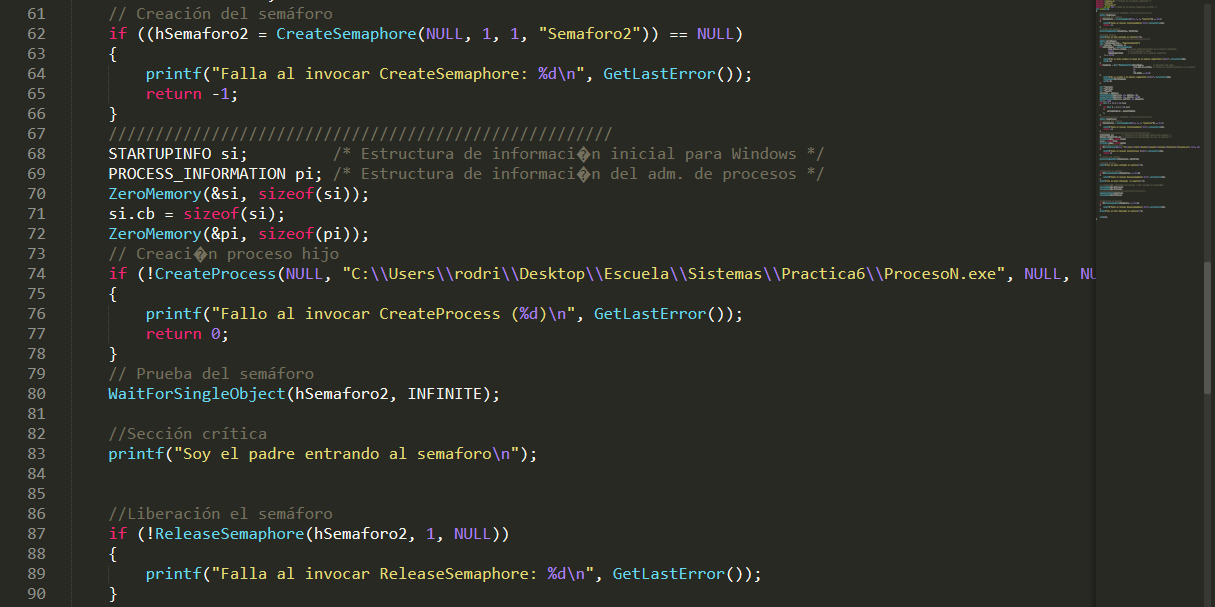


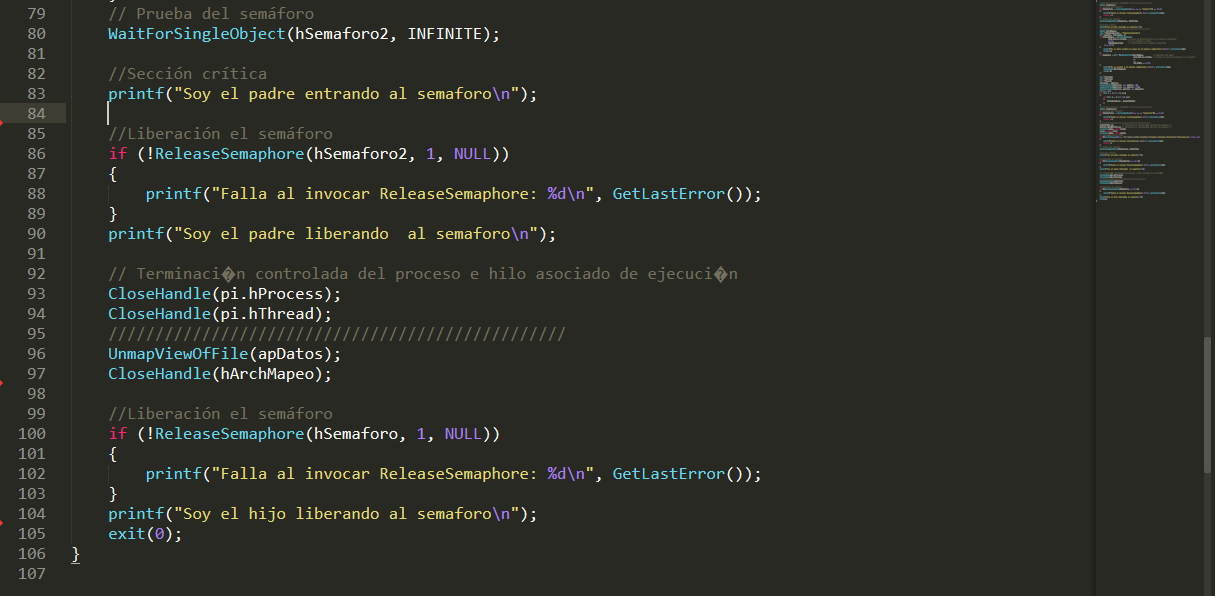


**Archivo: ProcesoH.c**

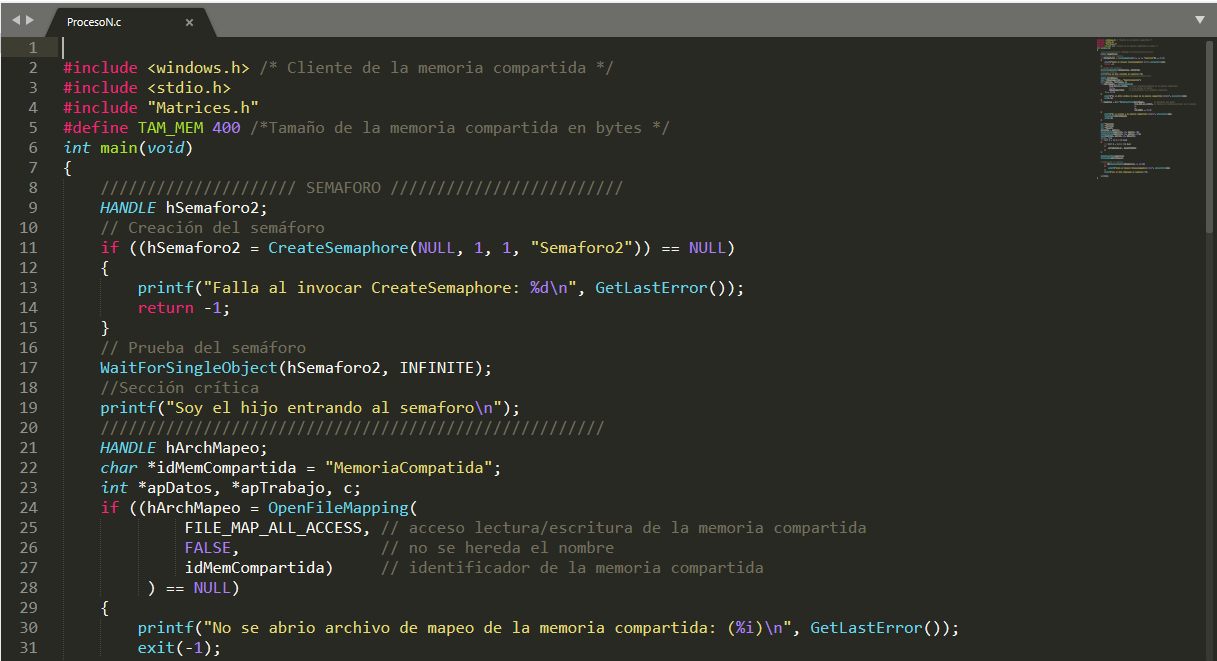


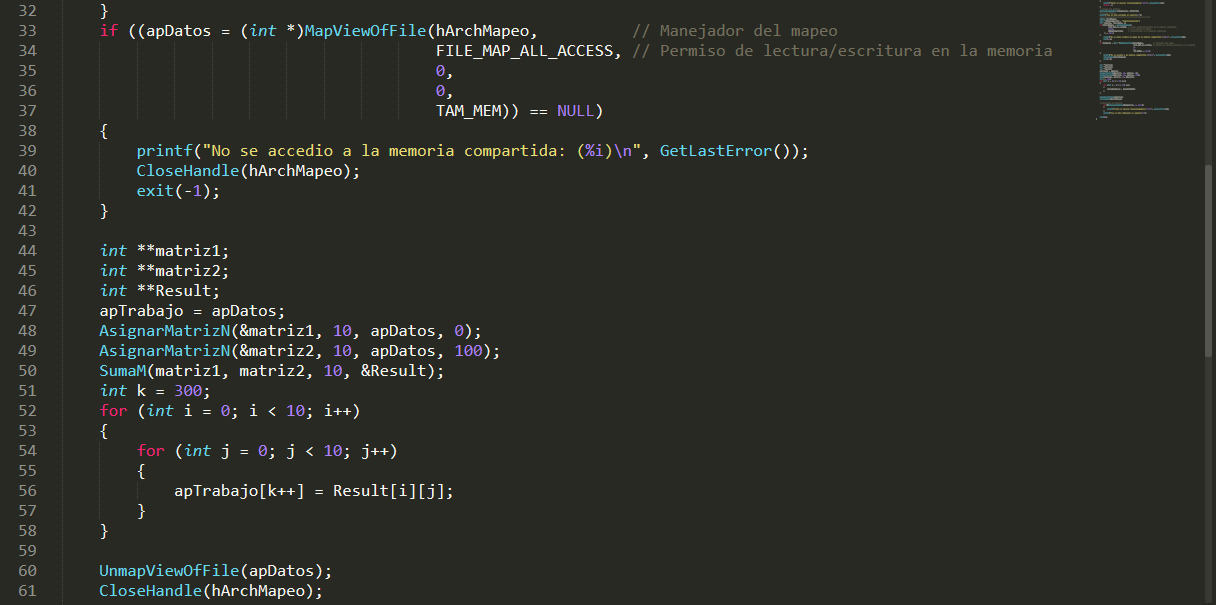


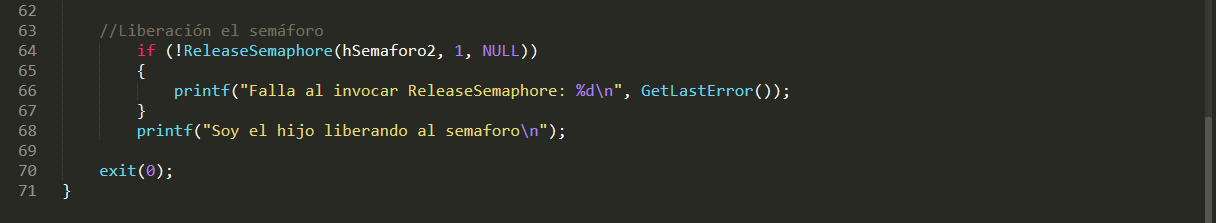




**Archivo: ProcesoN.c**

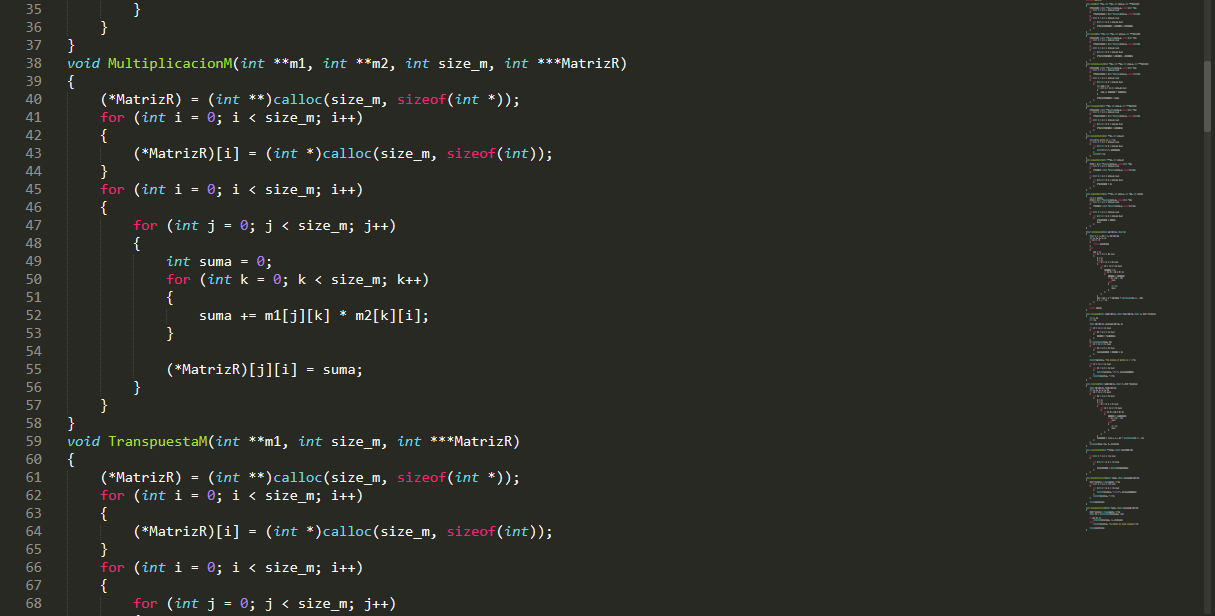
****

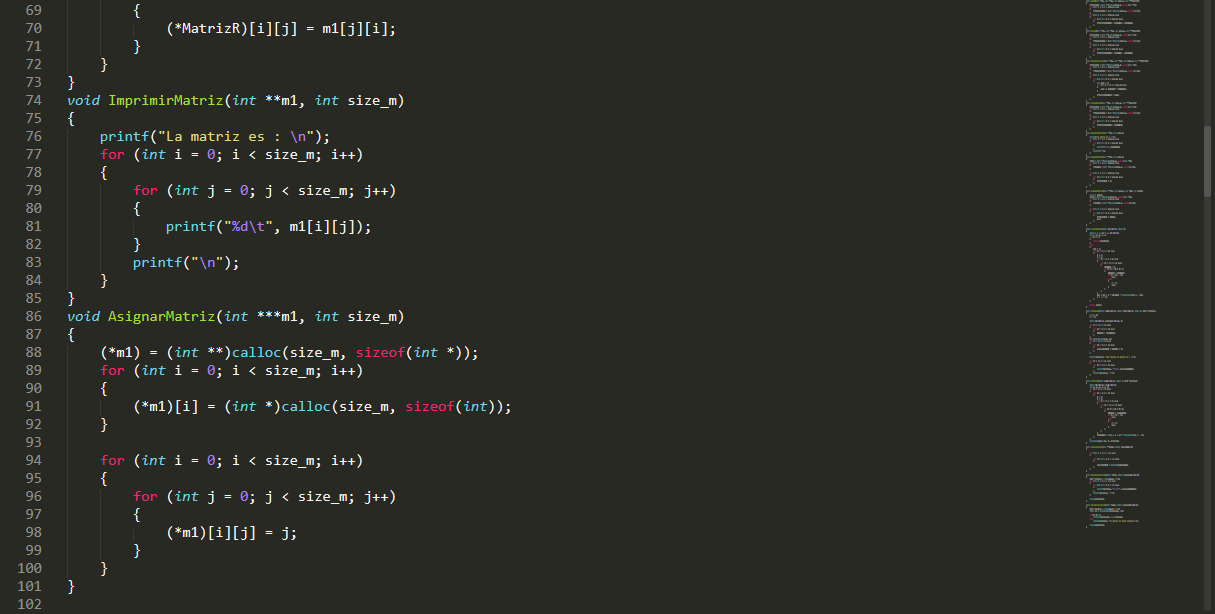
****

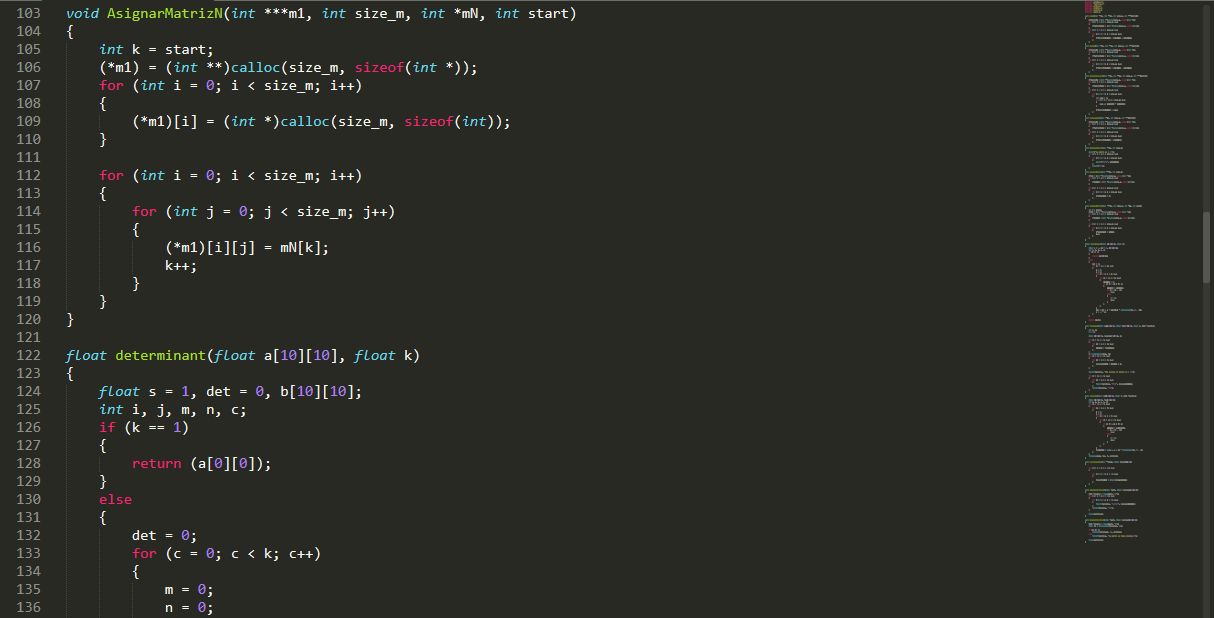
****

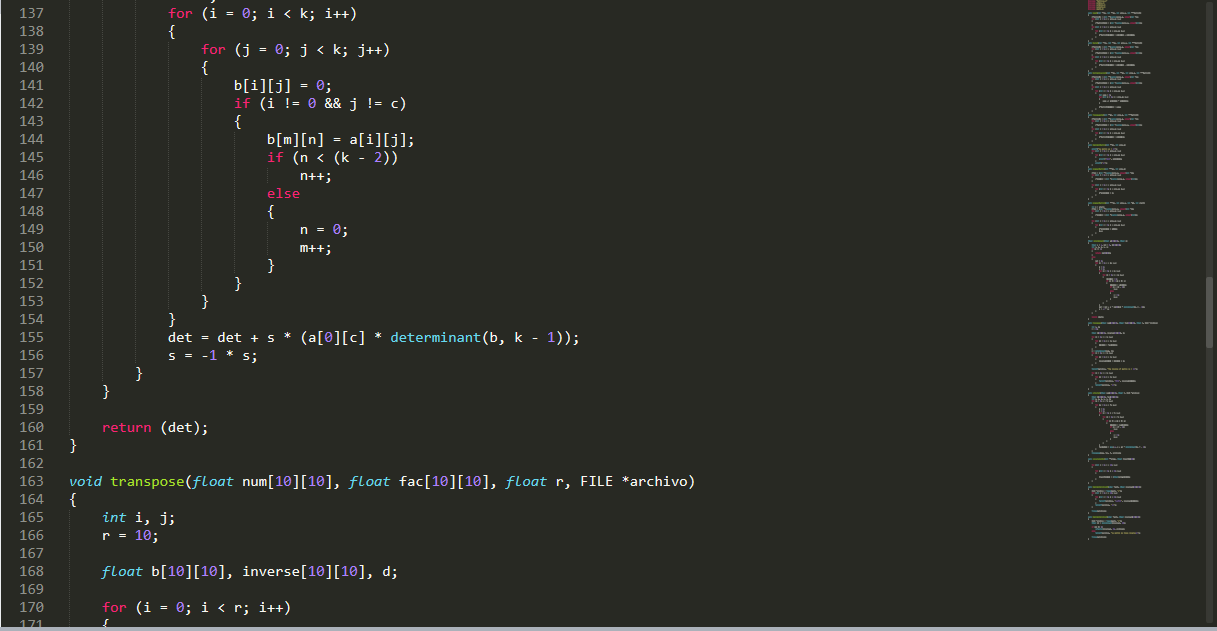
**Archivo: Matrices.c**

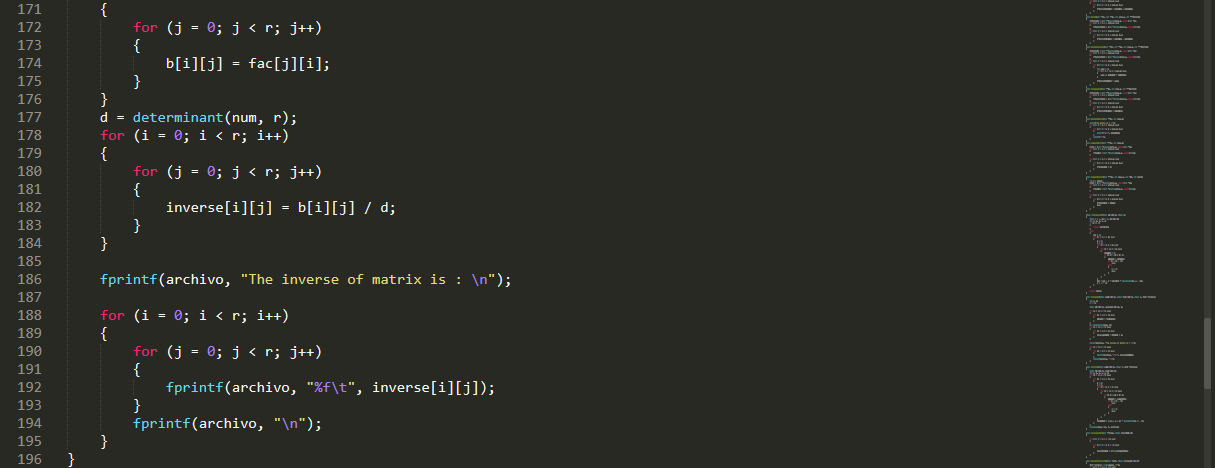
****

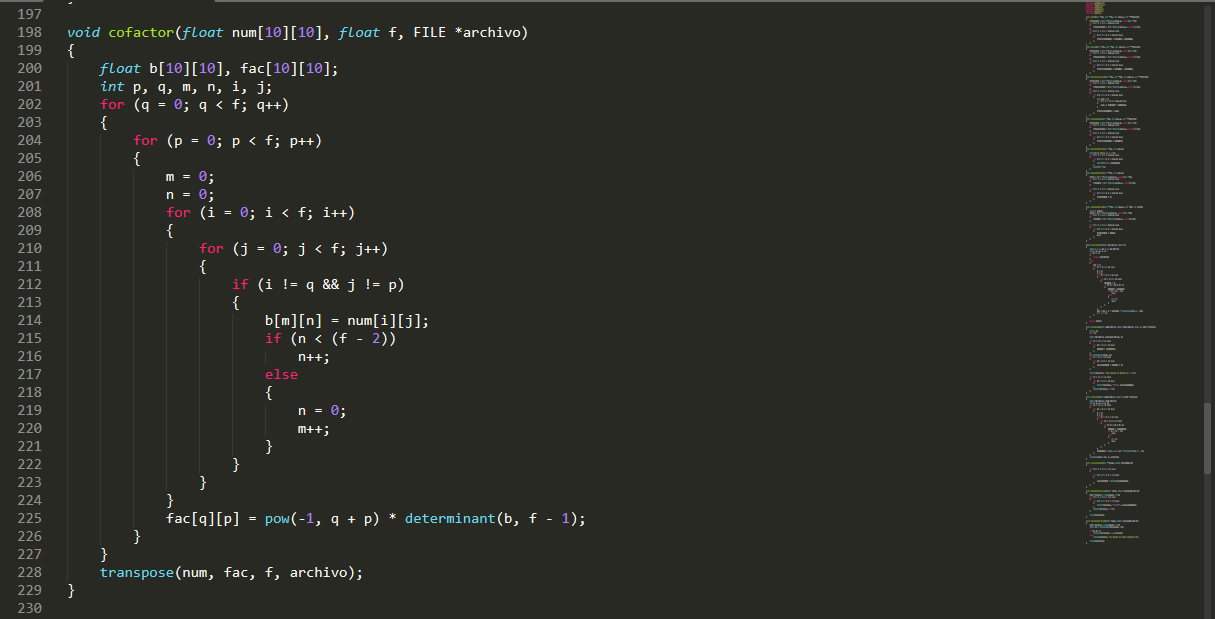
****

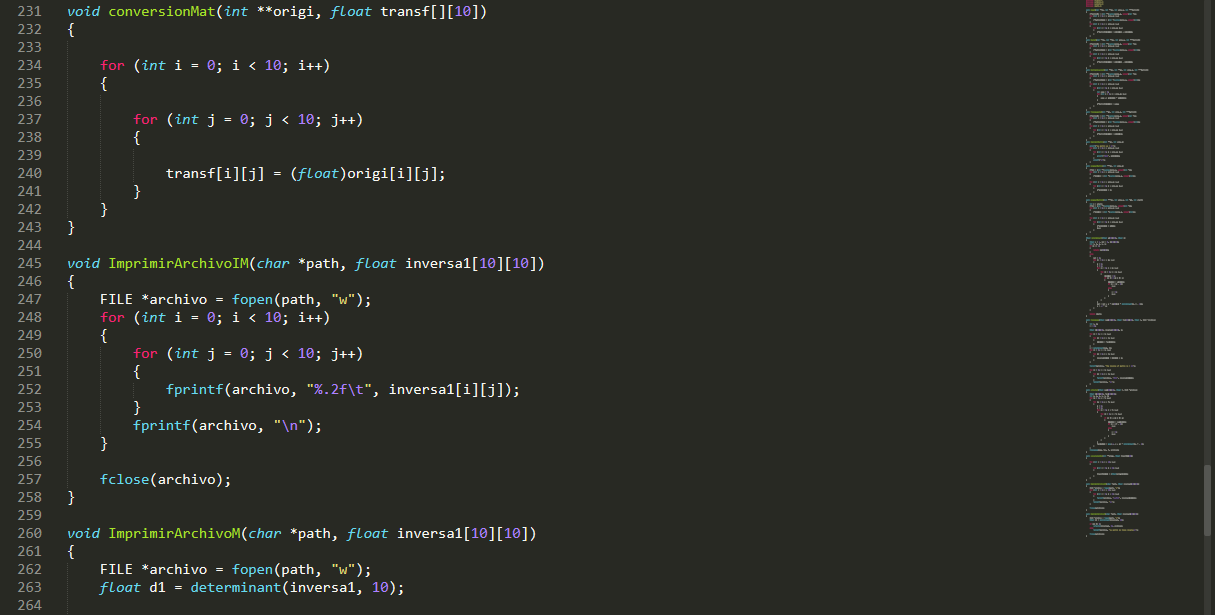
****

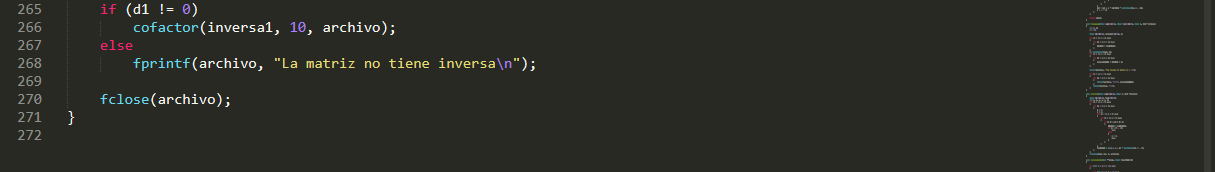
****

****



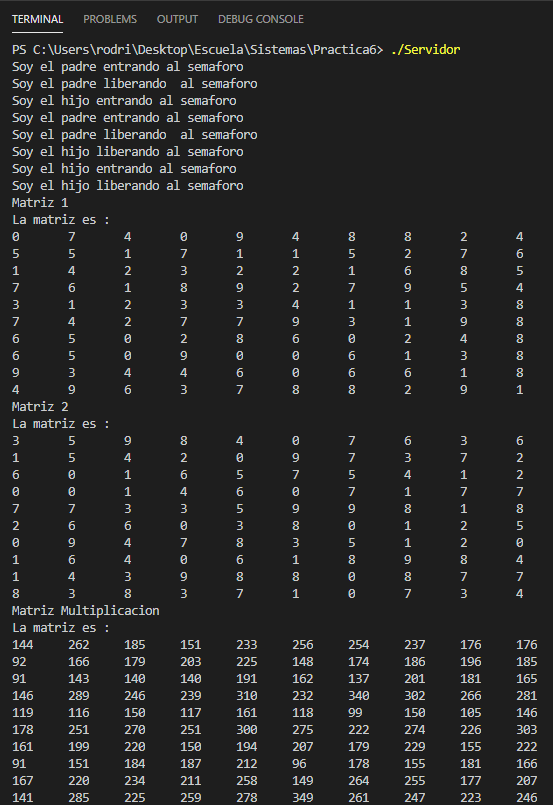


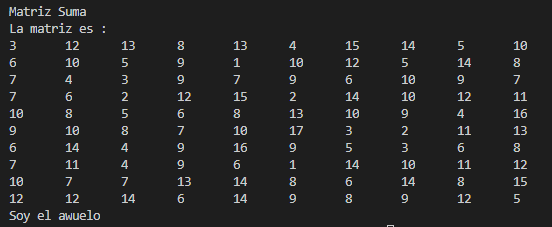




**Compilación y ejecución del programa**

Ejecución código principal

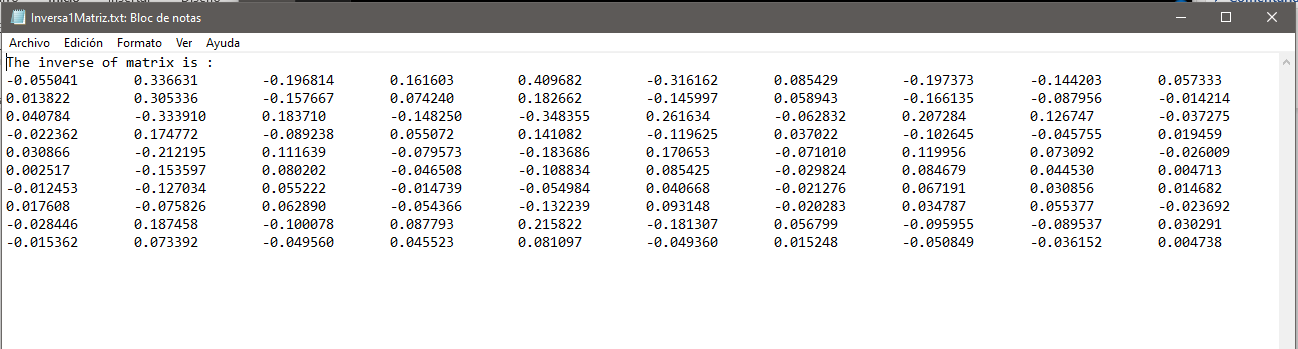




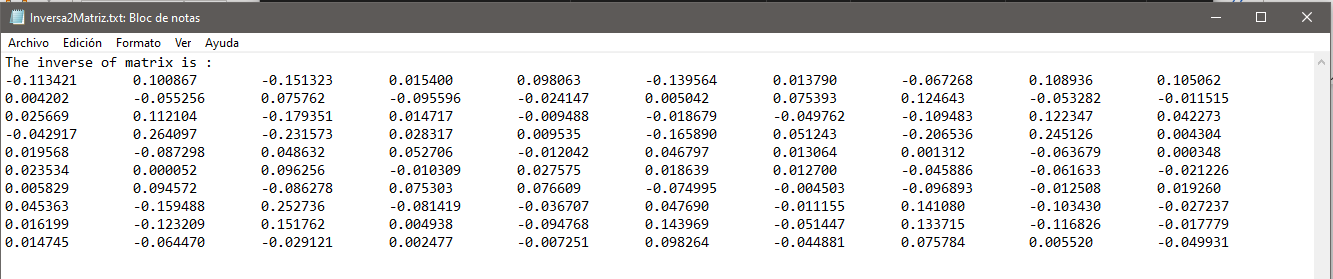
Archivos creados:



Inversa de la multiplicación



Inversa de la suma



# Conclusiones

Con el desarrollo de esta practica se pudo poner en práctica la utilización de los semáforos que nos proveen tanto Windows como Linux, con el objetivo de observar como se pueden hacer que los procesos cooperen entre sí, aunado a lo anterior, también se pudo observar la diferencia de implementar semáforos tanto en Linux como en Windows, sin embargo también se pudo identificar que pese a que la manera de implementarlos fuera diferente, la lógica detrás de estos semáforos seguía siendo exactamente la misma.