Instituto Politécnico Nacional

ESCOM

Practica 6

Mecanismos de sincronización de procesos en Linux y Windows (semáforos)

Sistemas Operativos – 2CM17

Integrantes:

Mora Ayala José Antonio

Ramírez Cotonieto Luis Fernando

Torres Carrillo Josehf Miguel Angel

Tovar Jacuinde Rodrigo

Profesor:

Cortes Galicia Jorge

INSTITUTO POLITÉCNICO NACIONAL

ESCOM

INTRODUCCIÓN

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 específica. 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 cómo el programador lo maneja.

Competencia.

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

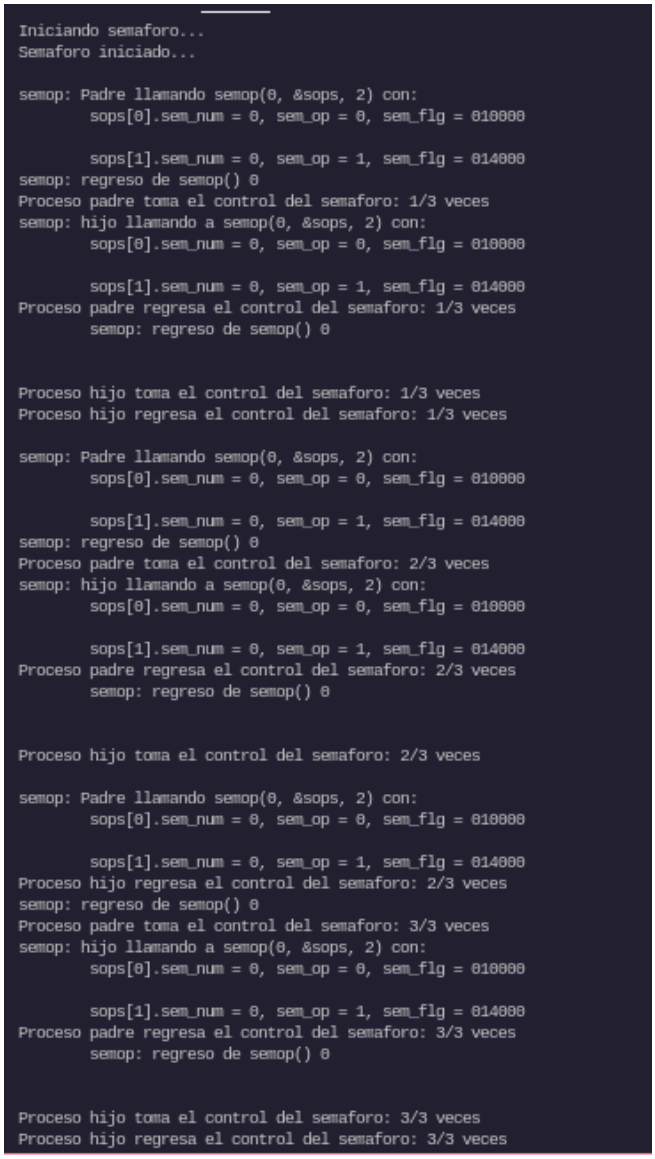
2. Capture, compile y ejecute el siguiente programa. Observe su funcionamiento y explique.

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int main(void)
{
    int i, j;
    int pid;
    int semid;
    key_t llave = 1234;
    int semban = IPC_CREAT | 0666;
    int nsems = 1;
    int nsops;
    struct sembuf *sops = (struct sembuf *)malloc(2 * sizeof(struct sembuf));
    printf("Iniciando semaforo...\n");
    if ((semid = semget(llave, nsems, semban)) == -1)
    {
        perror("semget: error al iniciar semaforo");
        exit(1);
    }
    else
        printf("Semaforo iniciado...\n");
    if ((pid = fork()) < 0)
    {
        perror("fork: error al crear proceso\n");
        exit(1);
    }
    if (pid == 0)
    {
        i = 0;
        while (i < 3)
        {
            nsops = 2;
            sops[0].sem_num = 0;
            sops[0].sem_op = 0;
            sops[0].sem_flg = SEM_UNDO;
```

```
            sops[1].sem_num = 0;
            sops[1].sem_op = 1;
            sops[1].sem_flg = SEM_UNDO | IPC_NOWAIT;
            printf("semop: hijo llamando a semop(%d, &sops, %d) con:", semid, nsops);
            for (j = 0; j < nsops; j++)
            {
                printf("\n\tsops[%d].sem_num = %d, ", j, sops[j].sem_num);
                printf("sem_op = %d, ", sops[j].sem_op);
                printf("sem_flg = %#o\n", sops[j].sem_flg);
            }
            if ((j = semop(semid, sops, nsops)) == -1)
            {
                perror("semop: error en operacion del semaforo\n");
            }
            else
            {
                printf("\tsemop: regreso de semop() %d\n", j);
                printf("\n\nProceso hijo toma el control del semaforo: %d/3 veces\n", i +
1);
                sleep(5);
                nsops = 1;
                sops[0].sem_num = 0;
                sops[0].sem_op = -1;
                sops[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
                if ((j = semop(semid, sops, nsops)) == -1)
                {
                    perror("semop: error en operacion del semaforo\n");
                }
                else
                    printf("Proceso hijo regresa el control del semaforo: %d/3 veces\n", i
+ 1);
                sleep(5);
            }
            ++i;
        }
    }
    else
    {
```

```
        i = 0;
        while (i < 3)
        {
            nsops = 2;
            sops[0].sem_num = 0;
            sops[0].sem_op = 0;
            sops[0].sem_flg = SEM_UNDO;

            sops[1].sem_num = 0;
            sops[1].sem_op = 1;
            sops[1].sem_flg = SEM_UNDO | IPC_NOWAIT;
            printf("\nsemop: Padre llamando semop(%d, &sops, %d) con:", semid, nsops);
            for (j = 0; j < nsops; j++)
            {
                printf("\n\tsops[%d].sem_num = %d, ", j, sops[j].sem_num);
                printf("sem_op = %d, ", sops[j].sem_op);
                printf("sem_flg = %#o\n", sops[j].sem_flg);
            }
            if ((j = semop(semid, sops, nsops)) == -1)
            {
                perror("semop: error en operacion del semaforo\n");
            }

            else
            {
                printf("semop: regreso de semop() %d\n", j);
                printf("Proceso padre toma el control del semaforo: %d/3 veces\n", i + 1);
                sleep(5);
                nsops = 1;
                sops[0].sem_num = 0;
                sops[0].sem_op = -1;
                sops[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
                if ((j = semop(semid, sops, nsops)) == -1)
                {
                    perror("semop: error en semop()\n");
                }
                else
                    printf("Proceso padre regresa el control del semaforo: %d/3 veces\n", i
```

```
+ 1);
                sleep(5);
            }
            ++i;
        }
    }
}
```



```
Iniciando semaforo...
Semaforo iniciado...

semop: Padre llamando semop(0, &sops, 2) con:
        sops[0].sem_num = 0, sem_op = 0, sem_flg = 010000

        sops[1].sem_num = 0, sem_op = 1, sem_flg = 014000
semop: regreso de semop() 0
Proceso padre toma el control del semaforo: 1/3 veces
semop: hijo llamando a semop(0, &sops, 2) con:
        sops[0].sem_num = 0, sem_op = 0, sem_flg = 010000

        sops[1].sem_num = 0, sem_op = 1, sem_flg = 014000
Proceso padre regresa el control del semaforo: 1/3 veces
        semop: regreso de semop() 0


Proceso hijo toma el control del semaforo: 1/3 veces
Proceso hijo regresa el control del semaforo: 1/3 veces

semop: Padre llamando semop(0, &sops, 2) con:
        sops[0].sem_num = 0, sem_op = 0, sem_flg = 010000

        sops[1].sem_num = 0, sem_op = 1, sem_flg = 014000
semop: regreso de semop() 0
Proceso padre toma el control del semaforo: 2/3 veces
semop: hijo llamando a semop(0, &sops, 2) con:
        sops[0].sem_num = 0, sem_op = 0, sem_flg = 010000

        sops[1].sem_num = 0, sem_op = 1, sem_flg = 014000
Proceso padre regresa el control del semaforo: 2/3 veces
        semop: regreso de semop() 0


Proceso hijo toma el control del semaforo: 2/3 veces

semop: Padre llamando semop(0, &sops, 2) con:
        sops[0].sem_num = 0, sem_op = 0, sem_flg = 010000

        sops[1].sem_num = 0, sem_op = 1, sem_flg = 014000
Proceso hijo regresa el control del semaforo: 2/3 veces
semop: regreso de semop() 0
Proceso padre toma el control del semaforo: 3/3 veces
semop: hijo llamando a semop(0, &sops, 2) con:
        sops[0].sem_num = 0, sem_op = 0, sem_flg = 010000

        sops[1].sem_num = 0, sem_op = 1, sem_flg = 014000
Proceso padre regresa el control del semaforo: 3/3 veces
        semop: regreso de semop() 0


Proceso hijo toma el control del semaforo: 3/3 veces
Proceso hijo regresa el control del semaforo: 3/3 veces
```

EJECUCIÓN DEL CÓDIGO EN LINUX

Como podemos observar el código funciona de manera adecuada, vemos cómo es que el proceso padre y el proceso hijo toman control del semáforo cada cierto período de tiempo
De igual manera la ejecución de este código de prueba nos permite comprender de mejor forma la manera en que se ejecutan y trabajan las funciones anteriormente analizadas ya no solo de una forma teórica sino práctica también
El programa consta de la comunicación entre tres procesos a través de tres secciones de memoria compartida de 400 bits. El control de acceso a el recurso de memoria compartida está implementando a través de semáforos, a través de la llamada semop, nosotros podemos bloquear o desbloquear el semáforo, esto dependiendo del valor colocado en el atributo sem_op de la estructura sembuf, como se puede observar al recibir las matrices a multiplicar el proceso hijo bloquea el semáforo, realiza las operaciones pertinentes y lo desbloquea, lo mismo ocurre con el proceso nieto, en este caso con las matrices a sumar y finalmente con el proceso padre que despliega los resultados y guarda en un archivo.

3. Capture, compile y ejecute los siguientes programas. Observe su funcionamiento. Ejecute de la siguiente forma: C:\>nombre_programa_padre nombre_programa_hijo

Padre.c

```
#include <windows.h> /*Programa padre*/
#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 */
 HANDLE hSemaforo;
 int i=1;
 ZeroMemory(&si, sizeof(si));
 si.cb = sizeof(si);
 ZeroMemory(&pi, sizeof(pi));
 if(argc!=2)
 {
 printf("Usar: %s Nombre_programa_hijo\n", argv[0]);
 return;
 }
 // Creación del semáforo
 if((hSemaforo = CreateSemaphore(NULL, 1, 1, "Semaforo")) == NULL)
 {
 printf("Falla al invocar CreateSemaphore: %d\n", GetLastError());
 return -1;
 }
 // Creación proceso hijo
 if(!CreateProcess(NULL, argv[1], NULL, NULL, FALSE, 0, NULL, NULL,
&si, &pi))
 {
 printf("Falla al invocar CreateProcess: %d\n", GetLastError() );
 return -1;
 }
while(i<4)
 {
 // Prueba del semáforo
```

```
 WaitForSingleObject(hSemaforo, INFINITE);

 //Sección crítica
 printf("Soy el padre entrando %i de 3 veces al semaforo\n",i);
 Sleep(5000);

 //Liberación el semáforo
 if (!ReleaseSemaphore(hSemaforo, 1, NULL) )
 {
 printf("Falla al invocar ReleaseSemaphore: %d\n", GetLastError());
 }
 printf("Soy el padre liberando %i de 3 veces al semaforo\n",i);
 Sleep(5000);

 i++;
 }
 // Terminación controlada del proceso e hilo asociado de ejecución
 CloseHandle(pi.hProcess);
 CloseHandle(pi.hThread);
}
```

Hijo.c

```
#include <windows.h> /*Programa hijo*/
#include <stdio.h>
int main()
{
 HANDLE hSemaforo;
 int i=1;

 // Apertura del semáforo
 if((hSemaforo = OpenSemaphore(SEMAPHORE_ALL_ACCESS, FALSE,
"Semaforo")) ==
NULL)
 {
```

```
 printf("Falla al invocar OpenSemaphore: %d\n", GetLastError());
 return -1;
 }

 while(i<4)
 {
 // Prueba del semáforo
 WaitForSingleObject(hSemaforo, INFINITE);

 //Sección crítica
 printf("Soy el hijo entrando %i de 3 veces al semaforo\n",i);
 Sleep(5000);

 //Liberación el semáforo
 if (!ReleaseSemaphore(hSemaforo, 1, NULL) )
 {
 printf("Falla al invocar ReleaseSemaphore: %d\n", GetLastError());
 }
 printf("Soy el hijo liberando %i de 3 veces al semaforo\n",i);
 Sleep(5000);

 i++;
 }
}
```

EJECUCION DEL CODIGO EN WINDOWS

Como se dijo, los semáforos sirven para sincronizar los procesos, en este caso de ejemplo, creamos un proceso con la función CreateSemaphore y aquí como parámetros debemos colocar la seguridad y acceso, el inicia del contador, el máximo del contador y su nombre, con ello podemos identificar al semáforo ya sea para disminuirlo con la función down, la cual en Windows se puede tomar la función waitForSingleObject(), como parámetro el semáforo y el tiempo de espera. Para incrementar el semáforo utilizamos en Windows la función ReleaseSemaphore, como se sabe estas acciones de incrementar y decrementar se hace ya que el semáforo es un indicador de entero, por lo que cuando el semáforo sea 0 el proceso podrá ejecutarse cuando sea diferente tendrá valores enteros como en este caso 1. En nuestro código de la imagen se inicializa el semáforo con 1, y de decrementa cuando sale de la función waitForSingleObject().

```
Soy el padre entrando 1 de 3 veces al semaforo
Soy el padre liberando 1 de 3 veces al semaforo
Soy el hijo entrando 1 de 3 veces al semaforo
Soy el hijo liberando 1 de 3 veces al semaforo
Soy el padre entrando 2 de 3 veces al semaforo
Soy el padre liberando 2 de 3 veces al semaforo
Soy el hijo entrando 2 de 3 veces al semaforo
Soy el hijo liberando 2 de 3 veces al semaforo
Soy el padre entrando 3 de 3 veces al semaforo
Soy el padre liberando 3 de 3 veces al semaforo
Soy el hijo entrando 3 de 3 veces al semaforo
Soy el hijo liberando 3 de 3 veces al semaforo
```

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. Úselelos en los lugares donde haya necesidad de sincronizar el acceso a memoria compartida.

PROGRAMACION LINUX

Padre.c

```c
#include <stdio.h>      //LIBRERIAS BASICAS PARA C
#include <stdlib.h>
#include <sys/types.h>  //LIBRERIAS PARA LA CERACION DE PROCESOS
#include <sys/wait.h>
#include <sys/ipc.h>    //LIBRERIAS PARA LA CREACION DE MEMORIA COMPARTIDA
#include <sys/shm.h>
#include <unistd.h>     //LIBRERIAS VARIAS PARA EL FUNCIONAMIENTO DEL PROGRAMA
#include <errno.h>
#include <sys/sem.h>

#define TAM_MEM 400

void obtencion ();
void lock ();
void unlock ();
void enviaMatrizAB ();
void reciboMatrizM ();
void reciboMatrizS ();
void inversa();

// OBTENER SEMAFORO P
key_t key;
int semid;
struct sembuf sb = {0, -1, 0};

// ENVIO PARA LAS MATRICES A Y B

int shmidAB;
key_t llaveAB = 5678;
char *shmAB, *sAB;

// RECIBO PARA LA MATRIZ M

int shmidM;
key_t llaveM = 5680;
char *shmM, *sM;

// RECIBO PARA LA MATRIZ S

int shmidS;
key_t llaveS = 5684;
char *shmS, *sS;

// MATRICES

int matrizA[10][10] = {

    {1, 3, 5, 2, 4, 1, 2, 5, 5, 3},
    {1, 3, 2, 4, 5, 1, 2, 3, 4, 1},
    {3, 2, 1, 1, 3, 3, 1, 4, 2, 4},
    {1, 1, 2, 4, 2, 5, 2, 1, 2, 4},
    {5, 5, 1, 1, 3, 1, 2, 4, 2, 3},
    {1, 4, 2, 3, 4, 1, 3, 1, 5, 2},
    {1, 2, 5, 5, 2, 4, 1, 3, 3, 2},
    {5, 4, 1, 4, 1, 4, 1, 3, 2, 5},
    {1, 5, 3, 5, 1, 2, 1, 3, 1, 1},
    {5, 1, 3, 1, 4, 2, 2, 4, 3, 1},
};

int matrizB[10][10] = {

    {2, 1, 3, 1, 4, 2, 5, 1, 2, 1},
    {1, 5, 1, 5, 3, 1, 2, 4, 1, 3},
    {3, 2, 3, 5, 1, 2, 1, 2, 4, 5},
    {2, 1, 4, 1, 2, 3, 5, 4, 4, 2},
    {1, 2, 2, 3, 3, 4, 1, 1, 5, 4},
    {3, 4, 2, 1, 5, 1, 4, 1, 4, 5},
    {2, 5, 4, 2, 3, 4, 1, 1, 2, 1},
    {1, 2, 4, 1, 1, 2, 3, 3, 1, 1},
    {3, 1, 4, 4, 1, 4, 1, 1, 2, 1},
    {3, 4, 1, 2, 1, 5, 2, 2, 1, 3},
};
```

```c
double identidadM[10][10] = {

    {1, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 1, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 1, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 1, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 1, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 1, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
};

double identidadS[10][10] = { ...
};

int matrizM[10][10];

int matrizS[10][10];

// VARIABLES VARIAS

int i = 0, j = 0, k = 0, aux1 = 0, aux2 = 0, aux3 = 0, aux4 = 0;

double auxS = 0, pivoteS = 0, saveS[10][10];
double auxM = 0, pivoteM = 0, saveM[10][10];

int main(){
//CODIGO PARA LA CREACION DE PROCESOS

    pid_t hijo;
    char *argv[2];

    argv[0]="/home/jomiantc/Escritorio/HIJO.exe";
    argv[1]="/home/jomiantc/Escritorio/NIETO.exe";
    argv[2]=NULL;

//CODIGO QUE SE EJECUTA ANTES DE CREAR EL HIJO

    obtencion();
    enviaMatrizAB();
    lock();

    if((hijo=fork())==-1)
        printf("Error al crear el proceso hijo\n");

    if(hijo==0){

        execv(argv[0],argv);
    }
    else{

        wait(0);
        unlock();
        reciboMatrizM();
        reciboMatrizS();
        lock();
        inversa();
        unlock();
        exit(0);
    }
}
```

```
149  void obtencion (){
150  // OBTENCION E INICIALIZACION PARA SEMAFORO PADRE
151
152      if ((key = ftok("Padre.c", 'P')) == -1){
153
154          perror("ftokP");
155          exit(1);
156      }
157
158      if ((semid = semget(key, 1, 0)) == -1){
159
160          perror("semgetP");
161          exit(1);
162      }
163  }
164
165  void lock (){
166  // BLOQUEO DE SEMAFORO
167      if ((semop(semid, &sb, 1)) == -1){
168
169          perror("semopP");
170          exit(1);
171      }
172  }
173
174  void unlock (){
175  // DESBLOQUEO DE SEMAFORO
176
177      sb.sem_op = 1;
178
179      if ((semop(semid, &sb, 1)) == -1){
180          perror("semopP");
181          exit(1);
182      }
183  }
184
185  void enviaMatrizAB (){
186  // CODIGO QUE ESTABLECE CONEXION PARA LAS MATRICES A Y B
187
188      if ((shmidAB = shmget(llaveAB, TAM_MEM, IPC_CREAT | 0666)) < 0) {
189          perror("Error al obtener memoria compartida P: shmget");
190          exit(-1);
191      }
192
193      if ((shmAB = shmat(shmidAB, NULL, 0)) == (char *) -1) {
194          perror("Error al enlazar la memoria compartida P: shmat");
195          exit(-1);
196      }
197
198  // CODIGO PARA ENVIAR MATRICES A Y B
199
200      sAB = shmAB;
201
202      for (i = 0; i < 10; i++){
203
204          for (j = 0; j < 10; j++){
205
206              *sAB++ = matrizA[i][j];
207          }
208      }
209
210      for (i = 0; i < 10; i++){
211
212          for (j = 0; j < 10; j++){
213
214              *sAB++ = matrizB[i][j];
215          }
216      }
217
218      *sAB++ = '\0';
219  }
```

```
221  void reciboMatrizM (){
222  // CODIGO QUE ESTABLECE LA CONEXION PARA LA MATRIZ M
223
224      if ((shmidM = shmget(llaveM, TAM_MEM, 0666)) < 0) {
225
226          perror("Error al obtener memoria compartida P: shmget");
227          exit(-1);
228      }
229
230      if ((shmM = shmat(shmidM, NULL, 0)) == (char *) -1) {
231
232          perror("Error al enlazar la memoria compartida P: shmat");
233          exit(-1);
234      }
235
236  //CODIGO QUE GUARDA LA MATRIZ M
237
238      for (sM = shmM; *sM != '\0'; sM++){
239
240          matrizM[aux1][aux2] = *sM;
241          aux2++;
242
243          if (aux2 == 10){
244
245              aux2 = 0;
246              aux1++;
247
248          }
249      }
250
251      *shmM = '*';
252  }
253
254  void reciboMatrizS (){
255  // CODIGO QUE ESTABLECE LA CONEXION PARA LA MATRIZ S
256
257      if ((shmidS = shmget(llaveS, TAM_MEM, 0666)) < 0) {
258
259          perror("Error al obtener memoria compartida P: shmget");
260          exit(-1);
261      }
262
263      if ((shmS = shmat(shmidS, NULL, 0)) == (char *) -1) {
264
265          perror("Error al enlazar la memoria compartida P: shmat");
266          exit(-1);
267      }
268
269  //CODIGO QUE GUARDA LA MATRIZ S
270
271      for (sS = shmS; *sS != '\0'; sS++){
272
273          matrizS[aux3][aux4] = *sS;
274          aux4++;
275
276          if (aux4 == 10){
277
278              aux4 = 0;
279              aux3++;
280          }
281      }
282
283      *shmS = '*';
284  }
```

```c
void inversa (){
// CODIGO QUE CALCULA LA INVERSA DE LAS MATRICES M Y S
    for (i = 0; i < 10; i++){

        for (j = 0; j < 10; j++){
            saveM[i][j] = matrizM[i][j];
            saveS[i][j] = matrizS[i][j];
        }
    }

    for (i = 0; i < 10; i++){
        pivoteM = saveM[i][i];
        pivoteS = saveS[i][i];

        for (k = 0; k < 10; k++){
            saveM[i][k] = saveM[i][k]/pivoteM;
            identidadM[i][k] = identidadM[i][k]/pivoteM;
            saveS[i][k] = saveS[i][k]/pivoteS;
            identidadS[i][k] = identidadS[i][k]/pivoteS;

        }

        for (j = 0; j < 10; j++){

            if (i != j) {

                auxM = saveM[j][i];
                auxS = saveS[j][i];

                for (k = 0; k < 10; k++){

                    saveM[j][k] = saveM[j][k] - auxM*saveM[i][k];
                    identidadM[j][k] = identidadM[j][k]- auxM*identidadM[i][k];
                    saveS[j][k] = saveS[j][k] - auxS*saveS[i][k];
                    identidadS[j][k] = identidadS[j][k]- auxS*identidadS[i][k];
                }
            }
        }
    }

    FILE *fichero;

    fichero = fopen("INVERSAS.txt","w");
    fprintf(fichero, "Matriz inversa de la multiplicacion: \n");
    printf("\nMatriz inversa de la multiplicacion\n");

    for (i = 0; i < 10; i++){

        for (j = 0; j < 10; j++){

            printf("%0.2lf, ", identidadM[i][j]);
            fprintf(fichero, "%0.2lf, ", identidadM[i][j]);
        }
        printf("\n");
        fprintf(fichero, "\n");
    }

    fprintf(fichero, "\nMatriz inversa de la suma: \n");
    printf("\nMatriz inversa de la Suma\n");

    for (i = 0; i < 10; i++){

        for (j = 0; j < 10; j++){
            printf("%0.2lf, ", identidadS[i][j]);
            fprintf(fichero, "%0.2lf, ", identidadS[i][j]);
        }
        printf("\n");
        fprintf(fichero, "\n");
    }

    fclose(fichero);
}
```

Hijo.c

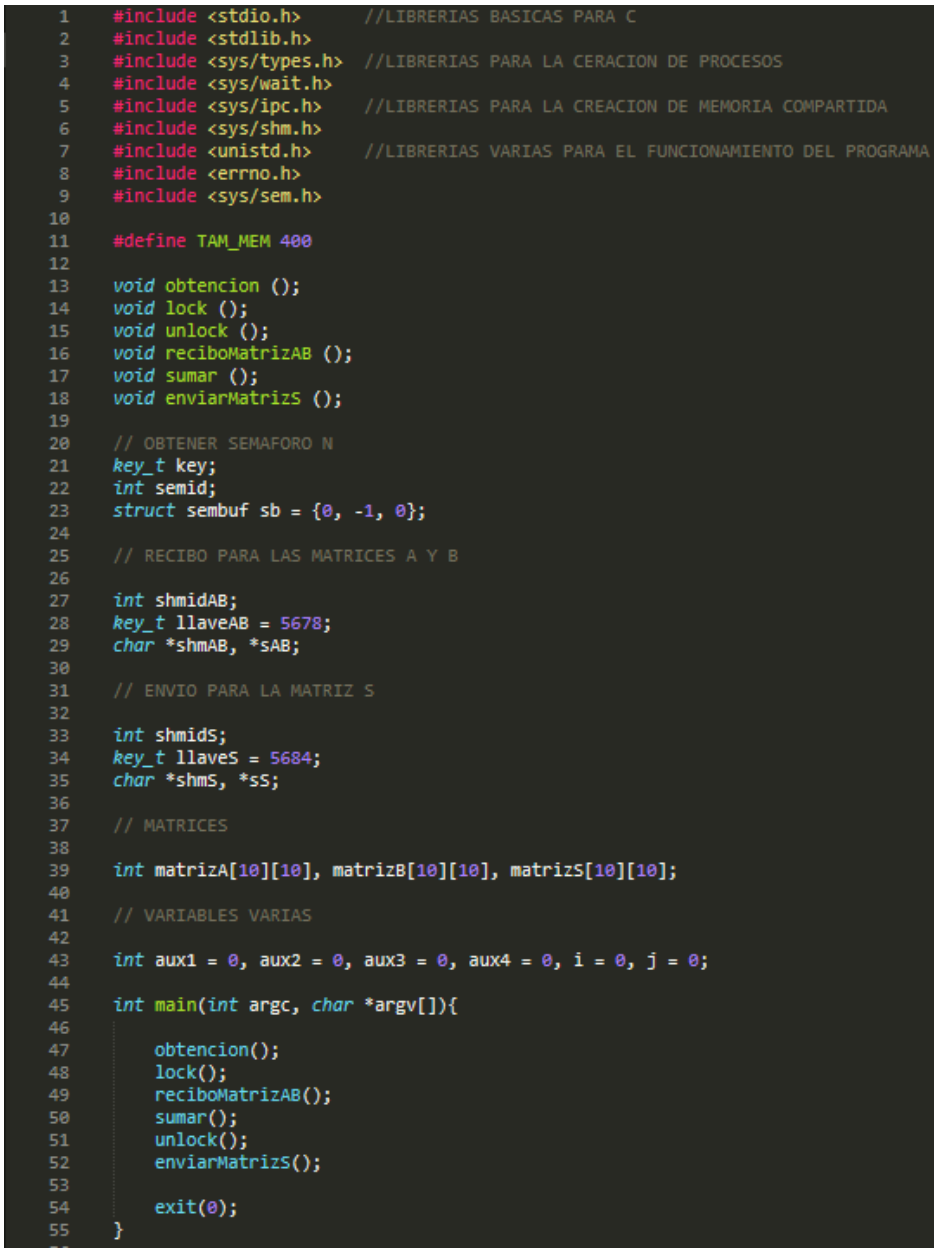
Algunas de las funciones fueron omitidas debido a que son significativamente similares a las que se encuentran en Padre.c por lo que resulta redundante volver

```c
#include <stdio.h>      //LIBRERIAS BASICAS PARA C
#include <stdlib.h>
#include <sys/types.h>  //LIBRERIAS PARA LA CERACION DE PROCESOS
#include <sys/wait.h>
#include <sys/ipc.h>    //LIBRERIAS PARA LA CREACION DE MEMORIA COMPARTIDA
#include <sys/shm.h>
#include <unistd.h>     //LIBRERIAS VARIAS PARA EL FUNCIONAMIENTO DEL PROGRAMA
#include <errno.h>
#include <sys/sem.h>

#define TAM_MEM 400

void obtencion ();
void lock ();
void unlock ();
void reciboMatrizAB ();
void multiplicar ();
void enviarMatrizM ();

// OBTENER SEMAFORO H
key_t key;
int semid;
struct sembuf sb = {0, -1, 0};

// RECIBO PARA LAS MATRICES A Y B

int shmidAB;
key_t llaveAB = 5678;
char *shmAB, *sAB;

// ENVIO PARA LA MATRIZ M

int shmidM;
key_t llaveM = 5680;
char *shmM, *sM;

// MATRICES

int matrizA[10][10], matrizB[10][10], matrizM[10][10];

// VARIABLES VARIAS

int aux1 = 0, aux2 = 0, aux3 = 0, aux4 = 0, suma = 0, i = 0, j = 0, a = 0;

int main(int argc, char *argv[]){
//CODIGO PARA LA CREACION DE PROCESOS

    pid_t nieto;

//CODIGO QUE SE EJECUTA ANTES DE CREAR EL NIETO
    obtencion();
    lock();
    reciboMatrizAB();
    multiplicar();
    unlock();
    enviarMatrizM();
    lock();

    if((nieto=fork())==-1)
        printf("Error al crear el proceso nieto\n");

    if(nieto==0){

        execv(argv[1],argv);
    }
    else{

        unlock();
        wait(0);
        exit(0);
    }
}
```

```
74  void obtencion (){
75  // OBTENCION E INICIALIZACION PARA SEMAFORO PADRE
76
77      if ((key = ftok("Hijo.c", 'H')) == -1){
78
79          perror("ftokH");
80          exit(1);
81      }
82
83      if ((semid = semget(key, 1, 0)) == -1){
84
85          perror("semgetH");
86          exit(1);
87      }
88  }
89
90  void lock (){
91  // BLOQUEO DE SEMAFORO
92      if ((semop(semid, &sb, 1)) == -1){
93
94          perror("semopH");
95          exit(1);
96      }
97  }
98
99  void unlock (){
100 // DESBLOQUEO DE SEMAFORO
101
102     sb.sem_op = 1;
103
104     if ((semop(semid, &sb, 1)) == -1){
105
106         perror("semopH");
107         exit(1);
108     }
109 }
110
111
112 void reciboMatrizAB (){
113 // CODIGO QUE ESTABLECE LA CONEXION PARA LAS MATRICES A Y B ...
126
127 //CODIGO QUE GUARDA LAS MATRICES A Y B ...
158 }
159
160 void multiplicar (){
161 //CODIGO QUE MULTIPLICA LAS MATRICES A Y B
162
163     for (a = 0; a < 10; a++) {
164
165         for (i = 0; i < 10; i++) {
166
167             suma = 0;
168
169             for (j = 0; j < 10; j++) {
170
171                 suma += matrizA[i][j] * matrizB[j][a];
172             }
173
174             matrizM[i][a] = suma+a;
175         }
176     }
177 }
178
179 void enviarMatrizM (){
180 // CODIGO QUE ESTABLECE CONEXION PARA LA MATRIZ M ...
193
194 // CODIGO PARA ENVIAR LA MATRIZ M ...
207 }
208
```

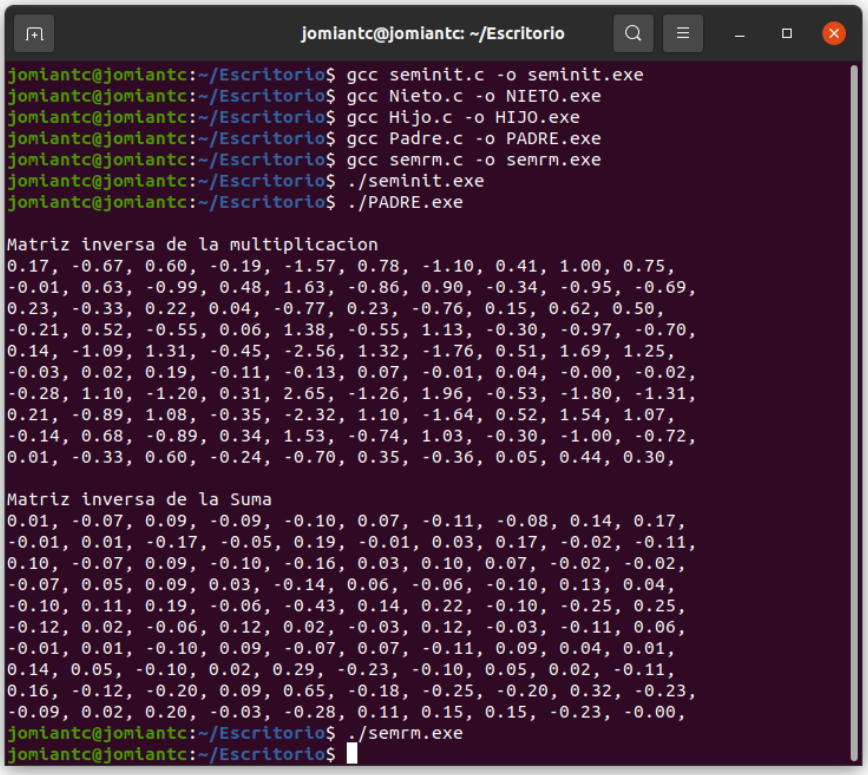
Nieto.c

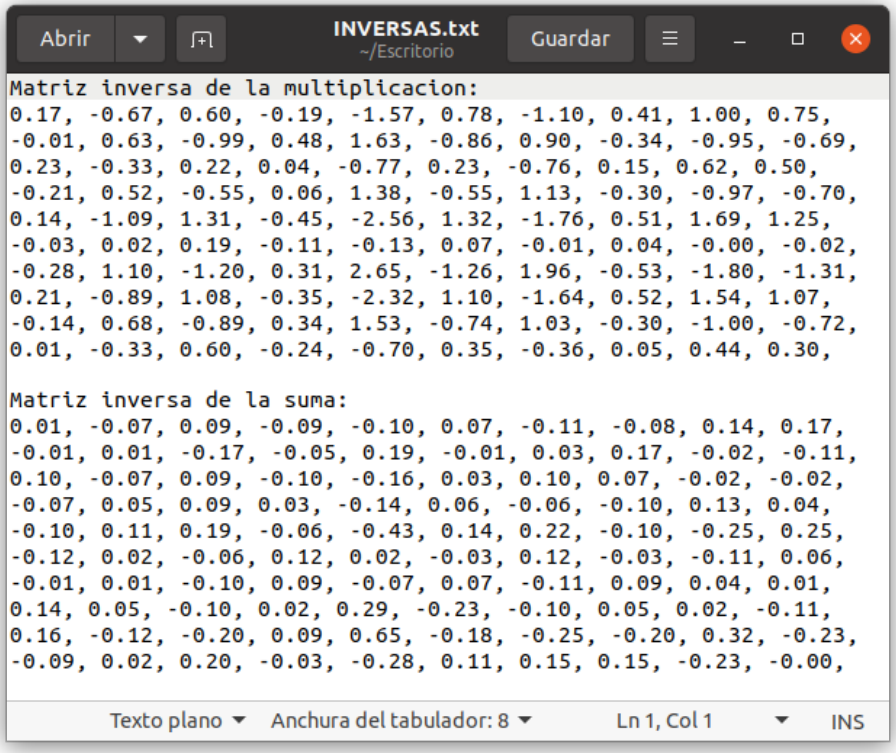
Algunas de las funciones fueron omitidas debido a que son significativamente similares a las que se encuentran en Padre.c por lo que resulta redundante volver a colocarlas



```c
#include <stdio.h>      //LIBRERIAS BASICAS PARA C
#include <stdlib.h>
#include <sys/types.h>  //LIBRERIAS PARA LA CERACION DE PROCESOS
#include <sys/wait.h>
#include <sys/ipc.h>    //LIBRERIAS PARA LA CREACION DE MEMORIA COMPARTIDA
#include <sys/shm.h>
#include <unistd.h>     //LIBRERIAS VARIAS PARA EL FUNCIONAMIENTO DEL PROGRAMA
#include <errno.h>
#include <sys/sem.h>

#define TAM_MEM 400

void obtencion ();
void lock ();
void unlock ();
void reciboMatrizAB ();
void sumar ();
void enviarMatrizS ();

// OBTENER SEMAFORO N
key_t key;
int semid;
struct sembuf sb = {0, -1, 0};

// RECIBO PARA LAS MATRICES A Y B

int shmidAB;
key_t llaveAB = 5678;
char *shmAB, *sAB;

// ENVIO PARA LA MATRIZ S

int shmidS;
key_t llaveS = 5684;
char *shmS, *sS;

// MATRICES

int matrizA[10][10], matrizB[10][10], matrizS[10][10];

// VARIABLES VARIAS

int aux1 = 0, aux2 = 0, aux3 = 0, aux4 = 0, i = 0, j = 0;

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

    obtencion();
    lock();
    reciboMatrizAB();
    sumar();
    unlock();
    enviarMatrizS();

    exit(0);
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int main () {

    key_t key;
    int semid;
    union semun{
        int val;
        struct semid_ds *buf;
        unsigned short *array;
    } arg;

// CIERRE DEL SEMAFORO PARA PADRE

    if ((key = ftok("Padre.c", 'P')) == -1){
        perror("ftok");
        exit(1);
    }
    if ((semid = semget(key, 1, 0)) == -1) {
        perror("semget");
        exit(1);
    }

    arg.val = 1;
    if (semctl(semid, 0, IPC_RMID, arg) == -1) {
        perror("semctl");
        exit(1);
    }

// CIERRE DEL SEMAFORO PARA HIJO

    if ((key = ftok("Hijo.c", 'H')) == -1){
        perror("ftok");
        exit(1);
    }
    if ((semid = semget(key, 1, 0)) == -1) {
        perror("semget");
        exit(1);
    }

    arg.val = 1;
    if (semctl(semid, 0, IPC_RMID, arg) == -1) {
        perror("semctl");
        exit(1);
    }

// CIERRE DEL SEMAFORO PARA NIETO

    if ((key = ftok("Nieto.c", 'N')) == -1){
        perror("ftok");
        exit(1);
    }
    if ((semid = semget(key, 1, 0)) == -1) {
        perror("semget");
        exit(1);
    }

    arg.val = 1;
    if (semctl(semid, 0, IPC_RMID, arg) == -1) {

        perror("semctl");
        exit(1);
    }

    return 0;
}
```


jomiantc@jomiantc: ~/Escritorio
jomiantc@jomiantc:~/Escritorio$ gcc seminit.c -o seminit.exe
jomiantc@jomiantc:~/Escritorio$ gcc Nieto.c -o NIETO.exe
jomiantc@jomiantc:~/Escritorio$ gcc Hijo.c -o HIJO.exe
jomiantc@jomiantc:~/Escritorio$ gcc Padre.c -o PADRE.exe
jomiantc@jomiantc:~/Escritorio$ gcc semrm.c -o semrm.exe
jomiantc@jomiantc:~/Escritorio$ ./seminit.exe
jomiantc@jomiantc:~/Escritorio$ ./PADRE.exe
Matriz inversa de la multiplicacion
0.17, -0.67, 0.60, -0.19, -1.57, 0.78, -1.10, 0.41, 1.00, 0.75,
-0.01, 0.63, -0.99, 0.48, 1.63, -0.86, 0.90, -0.34, -0.95, -0.69,
0.23, -0.33, 0.22, 0.04, -0.77, 0.23, -0.76, 0.15, 0.62, 0.50,
-0.21, 0.52, -0.55, 0.06, 1.38, -0.55, 1.13, -0.30, -0.97, -0.70,
0.14, -1.09, 1.31, -0.45, -2.56, 1.32, -1.76, 0.51, 1.69, 1.25,
-0.03, 0.02, 0.19, -0.11, -0.13, 0.07, -0.01, 0.04, -0.00, -0.02,
-0.28, 1.10, -1.20, 0.31, 2.65, -1.26, 1.96, -0.53, -1.80, -1.31,
0.21, -0.89, 1.08, -0.35, -2.32, 1.10, -1.64, 0.52, 1.54, 1.07,
-0.14, 0.68, -0.89, 0.34, 1.53, -0.74, 1.03, -0.30, -1.00, -0.72,
0.01, -0.33, 0.60, -0.24, -0.70, 0.35, -0.36, 0.05, 0.44, 0.30,
Matriz inversa de la Suma
0.01, -0.07, 0.09, -0.09, -0.10, 0.07, -0.11, -0.08, 0.14, 0.17,
-0.01, 0.01, -0.17, -0.05, 0.19, -0.01, 0.03, 0.17, -0.02, -0.11,
0.10, -0.07, 0.09, -0.10, -0.16, 0.03, 0.10, 0.07, -0.02, -0.02,
-0.07, 0.05, 0.09, 0.03, -0.14, 0.06, -0.06, -0.10, 0.13, 0.04,
-0.10, 0.11, 0.19, -0.06, -0.43, 0.14, 0.22, -0.10, -0.25, 0.25,
-0.12, 0.02, -0.06, 0.12, 0.02, -0.03, 0.12, -0.03, -0.11, 0.06,
-0.01, 0.01, -0.10, 0.09, -0.07, 0.07, -0.11, 0.09, 0.04, 0.01,
0.14, 0.05, -0.10, 0.02, 0.29, -0.23, -0.10, 0.05, 0.02, -0.11,
0.16, -0.12, -0.20, 0.09, 0.65, -0.18, -0.25, -0.20, 0.32, -0.23,
-0.09, 0.02, 0.20, -0.03, -0.28, 0.11, 0.15, 0.15, -0.23, -0.00,
jomiantc@jomiantc:~/Escritorio$ ./semrm.exe
jomiantc@jomiantc:~/Escritorio$


Abrir
INVERSAS.txt
~/Escritorio
Guardar
Matriz inversa de la multiplicacion:
0.17, -0.67, 0.60, -0.19, -1.57, 0.78, -1.10, 0.41, 1.00, 0.75,
-0.01, 0.63, -0.99, 0.48, 1.63, -0.86, 0.90, -0.34, -0.95, -0.69,
0.23, -0.33, 0.22, 0.04, -0.77, 0.23, -0.76, 0.15, 0.62, 0.50,
-0.21, 0.52, -0.55, 0.06, 1.38, -0.55, 1.13, -0.30, -0.97, -0.70,
0.14, -1.09, 1.31, -0.45, -2.56, 1.32, -1.76, 0.51, 1.69, 1.25,
-0.03, 0.02, 0.19, -0.11, -0.13, 0.07, -0.01, 0.04, -0.00, -0.02,
-0.28, 1.10, -1.20, 0.31, 2.65, -1.26, 1.96, -0.53, -1.80, -1.31,
0.21, -0.89, 1.08, -0.35, -2.32, 1.10, -1.64, 0.52, 1.54, 1.07,
-0.14, 0.68, -0.89, 0.34, 1.53, -0.74, 1.03, -0.30, -1.00, -0.72,
0.01, -0.33, 0.60, -0.24, -0.70, 0.35, -0.36, 0.05, 0.44, 0.30,
Matriz inversa de la suma:
0.01, -0.07, 0.09, -0.09, -0.10, 0.07, -0.11, -0.08, 0.14, 0.17,
-0.01, 0.01, -0.17, -0.05, 0.19, -0.01, 0.03, 0.17, -0.02, -0.11,
0.10, -0.07, 0.09, -0.10, -0.16, 0.03, 0.10, 0.07, -0.02, -0.02,
-0.07, 0.05, 0.09, 0.03, -0.14, 0.06, -0.06, -0.10, 0.13, 0.04,
-0.10, 0.11, 0.19, -0.06, -0.43, 0.14, 0.22, -0.10, -0.25, 0.25,
-0.12, 0.02, -0.06, 0.12, 0.02, -0.03, 0.12, -0.03, -0.11, 0.06,
-0.01, 0.01, -0.10, 0.09, -0.07, 0.07, -0.11, 0.09, 0.04, 0.01,
0.14, 0.05, -0.10, 0.02, 0.29, -0.23, -0.10, 0.05, 0.02, -0.11,
0.16, -0.12, -0.20, 0.09, 0.65, -0.18, -0.25, -0.20, 0.32, -0.23,
-0.09, 0.02, 0.20, -0.03, -0.28, 0.11, 0.15, 0.15, -0.23, -0.00,
Texto plano
Anchura del tabulador: 8
Ln 1, Col 1
INS

PROGRAMACION WINDOWS

Padre.c

```c
//Program that uses process inter communication to process two matrices with semaphores
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <stdbool.h>
#include <windows.h>

#define SIZE 10
#define MEM_SIZE 400

bool inverse(int A[SIZE][SIZE], float inverse[SIZE][SIZE]);
void adjoint(int A[SIZE][SIZE], int adj[SIZE][SIZE]);
void getCofactor(int A[SIZE][SIZE], int temp[SIZE][SIZE], int p, int q, int n);
int determinant(int A[SIZE][SIZE], int n);
int Imprime(int *a);
void LlenaMatriz(int **matriz);
void MatrizSM(int **matriz, int *b);


int main(void)
{
    int i, j, k;
    int *buffer, *shm;
    int **matrix1, **matrix2, result_matrix[SIZE][SIZE];
    float inv[SIZE][SIZE];
    FILE *write_fp;
    char *shmid = "SharedMemory";
    HANDLE hMapFile;
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    HANDLE hSem1, hSem2;

    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    srand((unsigned) time(NULL));

    //Asigancion de espacio de memoria a las matrices
    matrix1 = malloc(SIZE * sizeof(int *));
    matrix2 = malloc(SIZE * sizeof(int *));
    for(i = 0; i < SIZE; i++)
    {
        matrix1[i] = malloc(SIZE * sizeof(int));
        matrix2[i] = malloc(SIZE * sizeof(int));
    }

    //Llenando  matrices con valores aleatorios
    LlenaMatriz(matrix1);
    LlenaMatriz(matrix2);
    //Map shared memory
    if((hMapFile = CreateFileMapping(INVALID_HANDLE_VALUE, NULL, PAGE_READWRITE, 0, MEM_SIZE, shmid)) == NULL)
    {
        printf("Failed to map shared memory: (%i)\n", GetLastError());
        exit(EXIT_FAILURE);
    }
    //Create shared memory
    if((shm = (int *)MapViewOfFile(hMapFile, FILE_MAP_ALL_ACCESS, 0, 0, MEM_SIZE)) == NULL)
    {
        printf("Failed to create shared memory: (%i)\n", GetLastError());
        CloseHandle(hMapFile);
        exit(EXIT_FAILURE);
    }
    //MCreando un apuntador a la memoria compartida llamada Buffer
    buffer = shm;

    //Escribiendo matrices en la memoria compartida
    MatrizSM(matrix1,buffer);
    MatrizSM(matrix2,buffer);

    //Creando el primer semaforo
    if((hSem1 = CreateSemaphore(NULL, 1, 1, "Semaphore1")) == NULL)
```

```
    //Creando el primer semaforo
    if((hSem1 = CreateSemaphore(NULL, 1, 1, "Semaphore1")) == NULL)
    {
        printf("Failed to invoke CreateSemaphore: %d\n", GetLastError());
        return -1;
    }
    //Creando Semaforo 2
    if((hSem2 = CreateSemaphore(NULL, 1, 1, "Semaphore2")) == NULL)
    {
        printf("Failed to invoke CreateSemaphore: %d\n", GetLastError());
        return -1;
    }

    //creando Proceso Hijo
    if(!CreateProcess(NULL, "MatrixSemWChild", NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
    {
        printf("Failed to invoke CreateProcess: %d\n", GetLastError());
        return -1;
    }
    else //PADRE
    {
        //Halt in order to leave the Child take the control of both semaphores
        Sleep(1000);

        //Prueba del semáforo
        WaitForSingleObject(hSem1, INFINITE);

        //Imprimiendo las matrices en consola
        Imprime(buffer);

        //PADRE VUELVE A TOMAR CONTROL
        //leyendo la matriz resultante de la multiplicacion, la cual se encuentra almacenada en la memoria compartida
        //Por lo que almacenamos dichos valores en "result_matrix"
        k = 2 * SIZE * SIZE;
        for(i = 0; i < SIZE; i++)
            for(j = 0; j < SIZE; j++)
                result_matrix[i][j] = buffer[k++];

        //Calculando la inversa de la multiplicacion y escribiendola en archivo.
        write_fp = fopen("MultInverse.txt", "w");
        printf("Escribiendo la Inversa en un archivo\n");
        if(inverse(result_matrix, inv))
        {
            for(i = 0; i < SIZE; i++)
            {
                for(j = 0; j < SIZE; j++)
                    fprintf(write_fp, "%6.2f ", inv[i][j]);
                fprintf(write_fp, "\n");
            }
        }
        fclose(write_fp);

        //Metiendo en "result_matrix" los valores obtenidos de la suma los cuales estan en la memoria compartida
        k = 3 * SIZE * SIZE;
        for(i = 0; i < SIZE; i++)
            for(j = 0; j < SIZE; j++)
                result_matrix[i][j] = buffer[k++];

        //Calculando la inversa y Escribiendola en el archivo correspondiente
        write_fp = fopen("SumaInverse.txt", "w");
        printf("Enviando la Inversa de la Suma al archivo\n");
        if(inverse(result_matrix, inv))
        {
            for(i = 0; i < SIZE; i++)
            {
                for(j = 0; j < SIZE; j++)
                    fprintf(write_fp, "%6.2f ", inv[i][j]);
                fprintf(write_fp, "\n");
            }
        }
        fclose(write_fp);
    }
```

```
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
    UnmapViewOfFile(shm);
    CloseHandle(hMapFile);
}
//Funcion para calcular y almacenar la inversa, regresa valor falso en caso de que
//se trate de una matriz singular
bool inverse(int A[SIZE][SIZE], float inverse[SIZE][SIZE])
{
    // Encontrando el Determinante de A[][]
    int det = determinant(A, SIZE);
    if (det == 0)
    {
        printf("Matriz singular, no posee inversa\n");
        return false;
    }

    // Buscando la Adjunta
    int adj[SIZE][SIZE];
    adjoint(A, adj);

    // Inversa de la matriz = "inverse(A) = adj(A)/det(A)"
    for (int i=0; i<SIZE; i++)
        for (int j=0; j<SIZE; j++)
            inverse[i][j] = adj[i][j]/(float) det;

    return true;
}
// Funcion para obtener la ADJUNTA de A[N][N] en adj[N][N].
void adjoint(int A[SIZE][SIZE], int adj[SIZE][SIZE])
{
    if (SIZE == 1)
    {
        adj[0][0] = 1;
        return;
    }

    // temp almacena los cofactores de A[][]
    int sign = 1, temp[SIZE][SIZE];

    for (int i=0; i<SIZE; i++)
    {
        for (int j=0; j<SIZE; j++)
        {
            // Obteniendo cofactor de A[i][j]
            getCofactor(A, temp, i, j, SIZE);

            // sign of adj[j][i] positive if sum of row
            // and column indexes is even.
            sign = ((i+j)%2==0)? 1: -1;

            // Interchanging rows and columns to get the
            // transpose of the cofactor matrix
            adj[j][i] = (sign)*(determinant(temp, SIZE-1));
        }
    }
}

// Function to get cofactor of A[p][q] in temp[][]. n is current
// dimension of A[][]
void getCofactor(int A[SIZE][SIZE], int temp[SIZE][SIZE], int p, int q, int n)
{
    int i = 0, j = 0;

    // Looping for each element of the matrix
    for (int row = 0; row < n; row++)
    {
        for (int col = 0; col < n; col++)
        {
            //  Copying into temporary matrix only those element
            //  which are not in given row and column
            if (row != p && col != q)
```

ProgramaFinalWindows > C MatrizSemW.c > getCofactor(int [SIZE][SIZE], int [SIZE][SIZE], int, int, int)

```c
//Funcion que permite realizar la Impresion de las matrices escritas en m

int Imprime(int *a){
    int i;
    printf("\nMatriz numero 1\n\n");
    for(i = 0; i < SIZE * SIZE; i++)
        {
            if(i % 10 == 0 && i != 0)
                printf("\n");
            printf("%d ", a[i]);
        }
        printf("\n");


    printf("\nMatriz numero 2\n");

        for(; i < 2 * SIZE * SIZE; i++)
        {
            if(i % 10 == 0 && i != 0)
                printf("\n");
            printf("%d ", a[i]);
        }
        printf("\n");

    printf("\nMatriz Resultante de Multiplicacion\n");

        for(; i < 3 * SIZE * SIZE; i++)
        {
            if(i % 10 == 0 && i != 0)
                printf("\n");
            printf("%d ", a[i]);
        }
        printf("\n");

    printf("\nMatriz Resultante de Suma\n");

        for(; i < 4 * SIZE * SIZE; i++)
        {
            if(i % 10 == 0 && i != 0)
                printf("\n");
            printf("%d ", a[i]);
        }
        printf("\n");
}

//Funcion para llenar de valores aleatorios la matriz

void LlenaMatriz(int **matriz){
    int i,j;
    for(i = 0; i < SIZE; i++)
        for(j = 0; j < SIZE; j++)
        {
            matriz[i][j] = rand() % 5;
        }


}
// Funcion para escribir los valores en la memoria compartida

void MatrizSM(int **matriz, int *b){

        int i,j,k;
        k = 0;
    for(i = 0; i < SIZE; i++)
        for(j = 0; j < SIZE; j++)
            b[k++] = matriz[i][j];

    for(i = 0; i < SIZE; i++)
        for(j = 0; j < SIZE; j++)
            b[k++] = matriz[i][j];
}
```

Hijo.c

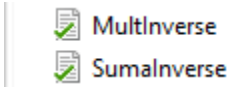
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <stdbool.h>
#include <windows.h>

#define SIZE 10
#define MEM_SIZE 400

int main(void)
{
    int matrix1[SIZE][SIZE], matrix2[SIZE][SIZE], result_matrix[SIZE][SIZE], i, j, k, accumulator;
    int *buffer, *shm;
    char *shmid = "SharedMemory";
    HANDLE hMapFile, hSem1, hSem2;
    PROCESS_INFORMATION piChild;
    STARTUPINFO siChild;

    //Abriendo Semaforo 1
    if((hSem1 = OpenSemaphore(SEMAPHORE_ALL_ACCESS, FALSE, "Semaphore1")) == NULL)
    {
        printf("Failed to invoke OpenSemaphore1: %d\n", GetLastError());
        return -1;
    }
    //Abriendo Semaforo 2
    if((hSem2 = OpenSemaphore(SEMAPHORE_ALL_ACCESS, FALSE, "Semaphore2")) == NULL)
    {
        printf("Failed to invoke OpenSemaphore2: %d\n", GetLastError());
        return -1;
    }
    ZeroMemory(&siChild, sizeof(siChild));
    siChild.cb = sizeof(siChild);
    ZeroMemory(&piChild, sizeof(piChild));

    //Open file mapping
    if((hMapFile = OpenFileMapping(FILE_MAP_ALL_ACCESS, FALSE, shmid)) == NULL)
    {
        printf("Failed to open file of mapping from shared memory: (%i)\n", GetLastError());
        exit(EXIT_FAILURE);
    }
    //Access/Attach to shared memory
    if((shm = (int *)MapViewOfFile(hMapFile, FILE_MAP_ALL_ACCESS, 0, 0, MEM_SIZE)) == NULL)
    {
        printf("Failed to access to shared memory: (%i)\n", GetLastError());
        CloseHandle(hMapFile);
        exit(EXIT_FAILURE);
    }
    //Apuntador a memoria compartida
    buffer = shm;

    //Creacion de Proceso Hijo
    if(!CreateProcess(NULL, "MatrixSemWGrandChild", NULL, NULL, FALSE, 0, NULL, NULL, &siChild, &piChild))
    {
        printf("Failed to create child process\n");
        exit(EXIT_FAILURE);
    }
    //Test control of semaphore 1
    WaitForSingleObject(hSem1, INFINITE);
    //Test control of semaphore 2
    WaitForSingleObject(hSem2, INFINITE);
    //CRITIC SECTION
    //Copiando en las matrices los valores almacenados en la memoria compartida
    k = 0;
    for(i = 0; i < SIZE; i++)
        for(j = 0; j < SIZE; j++)
            matrix1[i][j] = buffer[k++];

    for(i = 0; i < SIZE; i++)
        for(j = 0; j < SIZE; j++)
            matrix2[i][j] = buffer[k++];
```

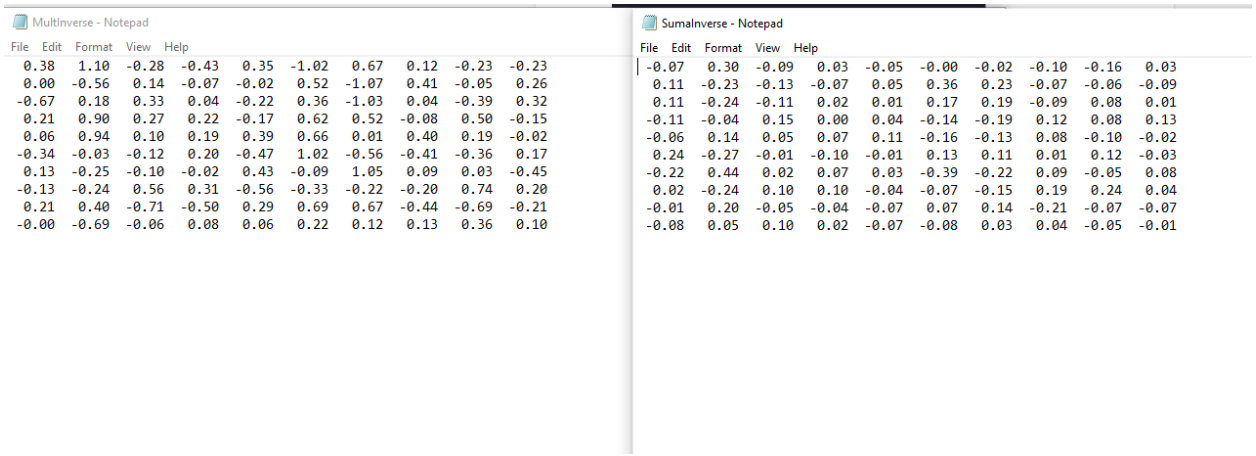
```
    //Apuntador a memoria compartida
    buffer = shm;

    //Creacion de Proceso Hijo
    if(!CreateProcess(NULL, "MatrixSemWGrandChild", NULL, NULL, FALSE, 0, NULL, NULL, &siChild, &piChild))
    {
        printf("Failed to create child process\n");
        exit(EXIT_FAILURE);
    }
    //Test control of semaphore 1
    WaitForSingleObject(hSem1, INFINITE);
    //Test control of semaphore 2
    WaitForSingleObject(hSem2, INFINITE);
    //CRITIC SECTION
    //Copiando en las matrices los valores almacenados en la memoria compartida
    k = 0;
    for(i = 0; i < SIZE; i++)
        for(j = 0; j < SIZE; j++)
            matrix1[i][j] = buffer[k++];

    for(i = 0; i < SIZE; i++)
        for(j = 0; j < SIZE; j++)
            matrix2[i][j] = buffer[k++];

    //Haciendo la multiplicacion
    for(j = 0; j < SIZE; j++)
    {
        for(k = 0; k < SIZE; k++)
        {
            accumulator = 0;
            for(int l = 0, m = 0; l < SIZE && m < SIZE; l++, m++)
                accumulator += matrix1[j][l] * matrix2[m][k];
            result_matrix[j][k] = accumulator;
        }
    }
    //Escribiendo resultado de la multiplicacion en la memoria compartida
    k = 2 * SIZE * SIZE;
    for(i = 0; i < SIZE; i++)
        for(j = 0; j < SIZE; j++)
            buffer[k++] = result_matrix[i][j];

    //FLiberando Semaforo 2
    if(!ReleaseSemaphore(hSem2, 1, NULL))
    {
        printf("Failed to invoke ReleaseSemaphore2: %d\n", GetLastError());
    }

    //Esperando la finalizacion del nieto, es necesario para no dejar al semaforo 1 abandonado
    WaitForSingleObject(piChild.hProcess, INFINITE);

    //Liberando Semaforo 1
    if(!ReleaseSemaphore(hSem1, 1, NULL))
    {
        printf("Failed to invoke ReleaseSemaphore1: %d\n", GetLastError());
    }
    CloseHandle(piChild.hThread);
    CloseHandle(piChild.hProcess);
    UnmapViewOfFile(shm);
    CloseHandle(hMapFile);

    return 0;
}
```

Nieto.c

```c
//Program that uses process inter communication to process two matrices
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <stdbool.h>
#include <windows.h>

#define SIZE 10
#define MEM_SIZE 400

int main(void)
{
    int matrix1[SIZE][SIZE], matrix2[SIZE][SIZE], result_matrix[SIZE][SIZE], i, j, k;
    int *buffer, *shm;
    char *shmid = "SharedMemory";
    HANDLE hMapFile, hSem2;

    //Abre Semaforo 2
    if((hSem2 = OpenSemaphore(SEMAPHORE_ALL_ACCESS, FALSE, "Semaphore2")) == NULL)
    {
        printf("Failed to invoke OpenSemaphore2: %d\n", GetLastError());
        return -1;
    }
    //Open file mapping
    if((hMapFile = OpenFileMapping(FILE_MAP_ALL_ACCESS, FALSE, shmid)) == NULL)
    {
        printf("Failed to open file of mapping from shared memory: (%i)\n", GetLastError());
        exit(EXIT_FAILURE);
    }
    //Access/Attach to shared memory
    if((shm = (int *)MapViewOfFile(hMapFile, FILE_MAP_ALL_ACCESS, 0, 0, MEM_SIZE)) == NULL)
    {
        printf("Failed to access to shared memory: (%i)\n", GetLastError());
        CloseHandle(hMapFile);
        exit(EXIT_FAILURE);
    }
    //Apuntador a Memoria Compartida
    buffer = shm;

    //Copiando en las matries los valores almacenados en la mmemoria compartida
    k = 0;
    for(i = 0; i < SIZE; i++)
        for(j = 0; j < SIZE; j++)
            matrix1[i][j] = buffer[k++];

    for(i = 0; i < SIZE; i++)
        for(j = 0; j < SIZE; j++)
            matrix2[i][j] = buffer[k++];

    //Realizando la suma
    for(i = 0; i < SIZE; i++)
        for(j = 0; j < SIZE; j++)
            result_matrix[i][j] = matrix1[i][j] + matrix2[i][j];

    //Detenido hasta que la multiplicacion esta hecha
    Sleep(1000);

    //Test control of semaphore 2
    WaitForSingleObject(hSem2, INFINITE);
    //Escribiendo la suma dentro de la memoria compartida
    k = 3 * SIZE * SIZE;
    for(i = 0; i < SIZE; i++)
        for(j = 0; j < SIZE; j++)
            buffer[k++] = result_matrix[i][j];

    //FLiberando al Semaforo 2
    if(!ReleaseSemaphore(hSem2, 1, NULL))
    {
        printf("Falla al invocar ReleaseSemaphore: %d\n", GetLastError());
    }
    UnmapViewOfFile(shm);
    CloseHandle(hMapFile);
```

```
Matriz numero 1

3 0 1 1 3 4 0 1 1 1
3 4 1 3 3 4 3 3 4 2
2 2 1 4 4 3 0 2 4 3
3 1 3 1 4 1 0 3 2 2
0 3 2 3 4 2 2 1 3 0
3 4 0 3 2 2 1 3 3 1
1 3 3 2 2 3 3 1 2 4
2 4 0 1 1 3 3 2 0 3
0 3 1 2 1 2 3 3 4 0
4 1 3 4 1 3 1 0 0 2

Matriz numero 2

3 0 1 1 3 4 0 1 1 1
3 4 1 3 3 4 3 3 4 2
2 2 1 4 4 3 0 2 4 3
3 1 3 1 4 1 0 3 2 2
0 3 2 3 4 2 2 1 3 0
3 4 0 3 2 2 1 3 3 1
1 3 3 2 2 3 3 1 2 4
2 4 0 1 1 3 3 2 0 3
0 3 1 2 1 2 3 3 4 0
4 1 3 4 1 3 1 0 0 2

Matriz Resultante de Multiplicacion

32 36 17 36 40 38 17 28 34 17
61 81 42 68 72 80 54 62 72 49
51 61 38 59 63 60 38 51 59 32
38 47 26 49 53 55 29 34 44 30
32 58 31 49 57 48 37 45 60 31
47 58 29 45 54 60 40 48 51 34
54 61 40 66 59 65 38 44 58 46
49 52 29 47 43 57 35 34 38 39
32 60 25 41 42 49 43 45 51 36
51 34 31 45 57 50 13 36 42 34

Matriz Resultante de Suma

6 0 2 2 6 8 0 2 2 2
6 8 2 6 6 8 6 6 8 4
4 4 2 8 8 6 0 4 8 6
6 2 6 2 8 2 0 6 4 4
0 6 4 6 8 4 4 2 6 0
6 8 0 6 4 4 2 6 6 2
2 6 6 4 4 6 6 2 4 8
4 8 0 2 2 6 6 4 0 6
0 6 2 4 2 4 6 6 8 0
8 2 6 8 2 6 2 0 0 4
Escribiendo la Inversa en un archivo
Enviando la Inversa de la Suma al archivo
PS C:\Users\TONY AYALA\Google Drive\Escolar\College\CuartoSemestre\Sistemas Operativos\Practic
```





MultInverse - Notepad

File Edit Format View Help

```
 0.38   1.10  -0.28  -0.43   0.35  -1.02   0.67   0.12  -0.23  -0.23
 0.00  -0.56   0.14  -0.07  -0.02   0.52  -1.07   0.41  -0.05   0.26
-0.67   0.18   0.33   0.04  -0.22   0.36  -1.03   0.04  -0.39   0.32
 0.21   0.90   0.27   0.22  -0.17   0.62   0.52  -0.08   0.50  -0.15
 0.06   0.94   0.10   0.19   0.39   0.66   0.01   0.40   0.19  -0.02
-0.34  -0.03  -0.12   0.20  -0.47   1.02  -0.56  -0.41  -0.36   0.17
 0.13  -0.25  -0.10  -0.02   0.43  -0.09   1.05   0.09   0.03  -0.45
-0.13  -0.24   0.56   0.31  -0.56  -0.33  -0.22  -0.20   0.74   0.20
 0.21   0.40  -0.71  -0.50   0.29   0.69   0.67  -0.44  -0.69  -0.21
-0.00  -0.69  -0.06   0.08   0.06   0.22   0.12   0.13   0.36   0.10
```

SumaInverse - Notepad

File Edit Format View Help

```
-0.07   0.30  -0.09   0.03  -0.05  -0.00  -0.02  -0.10  -0.16   0.03
 0.11  -0.23  -0.13  -0.07   0.05   0.36   0.23  -0.07  -0.06  -0.09
 0.11  -0.24  -0.11   0.02   0.01   0.17   0.19  -0.09   0.08   0.01
-0.11  -0.04   0.15   0.00   0.04  -0.14  -0.19   0.12   0.08   0.13
-0.06   0.14   0.05   0.07   0.11  -0.16  -0.13   0.08  -0.10  -0.02
 0.24  -0.27  -0.01  -0.10  -0.01   0.13   0.11   0.01   0.12  -0.03
-0.22   0.44   0.02   0.07   0.03  -0.39  -0.22   0.09  -0.05   0.08
 0.02  -0.24   0.10   0.10  -0.04  -0.07  -0.15   0.19   0.24   0.04
-0.01   0.20  -0.05  -0.04  -0.07   0.07   0.14  -0.21  -0.07  -0.07
-0.08   0.05   0.10   0.02  -0.07  -0.08   0.03   0.04  -0.05  -0.01
```

CONCLUSIONES

Como conclusión de esta práctica pude obtener un mayor conocimiento y manejo de diversos temas Como los apuntadores, estar más inmerso en tópicos básicos de programación y algebraicos como el manejo de punteros y de matrices, me ha permitido agilizar la forma en que programó, pues me permite desarrollar una programación mucho más estructurada, valiéndome del uso de funciones y tratando de hacer el código lo más dinámico posible.
Gracias a las lecturas y a la práctica que se tuvo que realizar dentro de este trabajo pude comprender de mejor manera el manejo de los semáforos y como se pueden ir implementando, asi como la importancia de las jerarquizaciones realizadas para que el programa funcione de forma correcta, pues la parte matemática de las matrices ya había sido trabajada y solo tuvo que ser adaptada

-Mora Ayala Jose Antonio

En esta práctica se pudieron utilizar los semáforos que contienen los sistemas operativos Windows y Linux, además de notar sus respectivas diferencias. Ambos sistemas operativos tienen maneras distintas para implementarlos, pero lógicamente funcionan de la misma forma. Antes de la materia, ignoraba o no había pensado en la gran importancia que se tiene al sincronizar procesos y llevarlos al mismo tiempo, realmente ayuda al rendimiento de nuestros dispositivos y los convierte en más efectivos. Con los semáforos se llega a comprender su funcionamiento pues es un proceso que podemos encontrar en nuestro mundo diario en cada esquina, donde los procesos van en orden y respetando las señales. Sin duda esta ha sido una de mis prácticas favoritas.

-Ramírez Cotonieto Luis Fernando

Esta práctica me resulto sencilla pero a la vez interesante, el código fue fácil de realizar pero no comprendía del todo la utilidad de los semáforos hasta revisar con detenimiento la documentación del profesor y la del propio sistema de Linux, me doy cuenta cada vez más de los pequeños sistemas con los que cuenta un sistema operativo y su funcionamiento, y como de complejas deben ser algunas tareas que nosotros como usuarios no notamos pero que están ahí implementadas, la práctica fue muy entretenida de desarrollar una vez entendido el funcionamiento de los semáforos y su utilidad

-Torres Carrillo Josehf Miguel Angel

Después de realizar esta practica conoci lo que son los mecanismos de sincronización entre procesos y también logre comprender su funcionalidad y utilidad como que estos mecanismos nos permiten forzar a un proceso a detener su ejecución hasta que ocurra otro en otro proceso, también entendí que esto es necesario para prevenir y/o corregir errores de los principales mecanismos de sincronización que ofrecen los sistemas operativos, también logré entender la implementación de los semáforos tanto en los sistemas de Windows y en Linux esto para poder comprender de un mejor modo cómo es que los procesos trabajan entre sí. También cabe resaltar que logré darme cuenta que en ambos sistemas operativos la implementación de las tuberías tienen una complejidad distinta, en lo personal se me facilitó más Linux, creo yo que al ser un sistema libre tienen un método más sencillo al de Windows, en lo general está práctica me ayudó a ver en qué diversas problemáticas puedo aplicar lo aprendido y en qué momentos resulta más eficiente emplear está gestión de procesos

-Tovar Jacuinde Rodrigo