

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 número 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.

```
#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\t%sops[%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\t sops[%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;

}

}

}

```

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.

```

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

```

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 inicio 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

```
1  #include <stdio.h>           //LIBRERIAS BASICAS PARA C
2  #include <stdlib.h>
3  #include <sys/types.h>       //LIBRERIAS PARA LA CREACION DE PROCESOS
4  #include <sys/wait.h>
5  #include <sys/ipc.h>         //LIBRERIAS PARA LA CREACION DE MEMORIA COMPARTIDA
6  #include <sys/shm.h>
7  #include <unistd.h>          //LIBRERIAS VARIAS PARA EL FUNCIONAMIENTO DEL PROGRAMA
8  #include <errno.h>
9  #include <sys/sem.h>
10
11  #define TAM_MEM 400
12
13  void obtencion ();
14  void lock ();
15  void unlock ();
16  void enviaMatrizA ();
17  void recibMatrizM ();
18  void recibMatrizS ();
19  void inversa();
20
21  // OBTENER SEMAFORO P
22  key_t key;
23  int semid;
24  struct sembuf sb = {0, -1, 0};
25
26  // ENVIO PARA LAS MATRICES A Y B
27
28  int shmIdAB;
29  key_t llaveAB = 5678;
30  char *shmAB, *sAB;
31
32  // RECIBO PARA LA MATRIZ M
33
34  int shmIdM;
35  key_t llaveM = 5688;
36  char *shmM, *sM;
37
38  // RECIBO PARA LA MATRIZ S
39
40  int shmIdS;
41  key_t llaveS = 5684;
42  char *shmS, *sS;
43
44  // MATRICES
45
46  int matrizA[10][10] = {
47
48      {1, 3, 5, 2, 4, 1, 2, 5, 5, 3},
49      {1, 3, 2, 4, 5, 1, 2, 3, 4, 1},
50      {3, 2, 1, 1, 3, 3, 1, 4, 2, 4},
51      {1, 1, 2, 4, 2, 5, 2, 1, 2, 4},
52      {5, 5, 1, 1, 3, 1, 2, 4, 2, 3},
53      {1, 4, 2, 3, 4, 1, 3, 1, 5, 2},
54      {1, 2, 5, 5, 2, 4, 1, 3, 3, 2},
55      {5, 4, 1, 4, 1, 4, 1, 3, 2, 5},
56      {1, 5, 3, 5, 1, 2, 1, 3, 1, 1},
57      {5, 1, 3, 1, 4, 2, 2, 4, 3, 1},
58  };
59
60  int matrizB[10][10] = {
61
62      {2, 1, 3, 1, 4, 2, 5, 1, 2, 1},
63      {1, 5, 1, 5, 3, 1, 2, 4, 1, 3},
64      {3, 2, 3, 5, 1, 2, 1, 2, 4, 5},
65      {2, 1, 4, 1, 2, 3, 5, 4, 4, 2},
66      {1, 2, 2, 3, 3, 4, 1, 1, 5, 4},
67      {3, 4, 2, 1, 5, 1, 4, 1, 4, 5},
68      {2, 5, 4, 2, 3, 4, 1, 1, 2, 1},
69      {1, 2, 4, 1, 1, 2, 3, 3, 1, 1},
70      {3, 1, 4, 4, 1, 4, 1, 1, 2, 1},
71      {3, 4, 1, 2, 1, 5, 2, 2, 1, 3},
72  };
73
```

```

74 double identidadM[10][10] = {
75
76     {1, 0, 0, 0, 0, 0, 0, 0, 0, 0},
77     {0, 1, 0, 0, 0, 0, 0, 0, 0, 0},
78     {0, 0, 1, 0, 0, 0, 0, 0, 0, 0},
79     {0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
80     {0, 0, 0, 0, 1, 0, 0, 0, 0, 0},
81     {0, 0, 0, 0, 0, 1, 0, 0, 0, 0},
82     {0, 0, 0, 0, 0, 0, 1, 0, 0, 0},
83     {0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
84     {0, 0, 0, 0, 0, 0, 0, 0, 1, 0},
85     {0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
86 };
87
88 double identidadS[10][10] = {
89     };
90
91 int matrizM[10][10];
92 int matrizS[10][10];
93
94 // VARIABLES VARIAS
95
96 int i = 0, j = 0, k = 0, aux1 = 0, aux2 = 0, aux3 = 0, aux4 = 0;
97
98 double auxS = 0, pivoteS = 0, saveS[10][10];
99 double auxM = 0, pivoteM = 0, saveM[10][10];
100
101 int main(){
102     //CODIGO PARA LA CREACION DE PROCESOS
103
104     pid_t hijo;
105     char *argv[2];
106
107     argv[0]="/home/jomiantc/Escritorio/HIJO.exe";
108     argv[1]="/home/jomiantc/Escritorio/NIETO.exe";
109     argv[2]=NULL;
110
111     //CODIGO QUE SE EJECUTA ANTES DE CREAR EL HIJO
112
113     obtencion();
114     enviaMatrizAB();
115     lock();
116
117     if((hijo=fork())==-1)
118         printf("Error al crear el proceso hijo\n");
119
120     if(hijo==0){
121         execv(argv[0],argv);
122     }
123     else{
124         wait(0);
125         unlock();
126         reciboMatrizM();
127         reciboMatrizS();
128         lock();
129         inversa();
130         unlock();
131         exit(0);
132     }
133 }

```



```

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 ((shmAB = 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 = '\0';
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
284     *shmS = '\0';

```

```

286 void inversa (){
287 // CODIGO QUE CALCULA LA INVERSA DE LAS MATRICES M Y S
288     for (i = 0; i < 10; i++){
289         for (j = 0; j < 10; j++){
290             saveM[i][j] = matrizM[i][j];
291             saveS[i][j] = matrizS[i][j];
292         }
293     }
294 }
295
296     for (i = 0; i < 10; i++){
297         pivoteM = saveM[i][i];
298         pivoteS = saveS[i][i];
299
300         for (k = 0; k < 10; k++){
301             saveM[i][k] = saveM[i][k]/pivoteM;
302             identidadM[i][k] = identidadM[i][k]/pivoteM;
303             saveS[i][k] = saveS[i][k]/pivoteS;
304             identidadS[i][k] = identidadS[i][k]/pivoteS;
305         }
306     }
307
308     for (j = 0; j < 10; j++){
309         if (i != j) {
310             auxM = saveM[j][i];
311             auxS = saveS[j][i];
312
313             for (k = 0; k < 10; k++){
314                 saveM[j][k] = saveM[j][k] - auxM*saveM[i][k];
315                 identidadM[j][k] = identidadM[j][k] - auxM*identidadM[i][k];
316                 saveS[j][k] = saveS[j][k] - auxS*saveS[i][k];
317                 identidadS[j][k] = identidadS[j][k] - auxS*identidadS[i][k];
318             }
319         }
320     }
321 }
322 }
323 }
324 }
325
326 FILE *fichero;
327
328 fichero = fopen("INVERSAS.txt","w");
329 fprintf(fichero, "Matriz inversa de la multiplicacion: \n");
330 printf("\nMatriz inversa de la multiplicacion\n");
331
332     for (i = 0; i < 10; i++){
333         for (j = 0; j < 10; j++){
334             printf("%0.2lf, ", identidadM[i][j]);
335             fprintf(fichero, "%0.2lf, ", identidadM[i][j]);
336         }
337         printf("\n");
338         fprintf(fichero, "\n");
339     }
340
341     fprintf(fichero, "\nMatriz inversa de la suma: \n");
342     printf("\nMatriz inversa de la Suma\n");
343
344     for (i = 0; i < 10; i++){
345         for (j = 0; j < 10; j++){
346             printf("%0.2lf, ", identidadS[i][j]);
347             fprintf(fichero, "%0.2lf, ", identidadS[i][j]);
348         }
349         printf("\n");
350         fprintf(fichero, "\n");
351     }
352
353     fclose(fichero);
354 }
355 }
356 }
357 }

```

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

```
1  #include <stdio.h>           //LIBRERIAS BASICAS PARA C
2  #include <stdlib.h>
3  #include <sys/types.h>       //LIBRERIAS PARA LA CERCACION DE PROCESOS
4  #include <sys/wait.h>
5  #include <sys/ipc.h>         //LIBRERIAS PARA LA CREACION DE MEMORIA COMPARTIDA
6  #include <sys/shm.h>
7  #include <unistd.h>          //LIBRERIAS VARIAS PARA EL FUNCIONAMIENTO DEL PROGRAMA
8  #include <errno.h>
9  #include <sys/sem.h>
10
11 #define TAM_MEM 400
12
13 void obtencion ();
14 void lock ();
15 void unlock ();
16 void reciboMatrizAB ();
17 void multiplicar ();
18 void enviarMatrizM ();
19
20 // OBTENER SEMAFORO H
21 key_t key;
22 int semid;
23 struct sembuf sb = {0, -1, 0};
24
25 // RECIBO PARA LAS MATRICES A Y B
26
27 int shmidAB;
28 key_t llaveAB = 5678;
29 char *shmAB, *sAB;
30
31 // ENVIO PARA LA MATRIZ M
32
33 int shmidM;
34 key_t llaveM = 5680;
35 char *shmM, *sM;
36
37 // MATRICES
38
39 int matrizA[10][10], matrizB[10][10], matrizM[10][10];
40
41 // VARIABLES VARIAS
42
43 int aux1 = 0, aux2 = 0, aux3 = 0, aux4 = 0, suma = 0, i = 0, j = 0, a = 0;
44
45 int main(int argc, char *argv[]){
46     //CODIGO PARA LA CREACION DE PROCESOS
47     pid_t nieto;
48
49     //CODIGO QUE SE EJECUTA ANTES DE CREAR EL NIETO
50     obtencion();
51     lock();
52     reciboMatrizAB();
53     multiplicar();
54     unlock();
55     enviarMatrizM();
56     lock();
57
58     if((nieto=fork())==-1)
59         printf("Error al crear el proceso nieto\n");
60
61     if(nieto==0){
62         execv(argv[1],argv);
63     }
64     else{
65         unlock();
66         wait(0);
67         exit(0);
68     }
69 }
70
71
72
73
```

```

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 ***
114
115 //CODIGO QUE GUARDA LAS MATRICES A Y B ***
116 }
117
118 void multiplicar (){
119 //CODIGO QUE MULTIPLICA LAS MATRICES A Y B
120
121     for (a = 0; a < 10; a++) {
122
123         for (i = 0; i < 10; i++) {
124
125             suma = 0;
126
127             for (j = 0; j < 10; j++) {
128
129                 suma += matrizA[i][j] * matrizB[j][a];
130             }
131
132             matrizM[i][a] = suma+a;
133         }
134     }
135 }
136
137 void enviarMatrizM (){
138 // CODIGO QUE ESTABLECE CONEXION PARA LA MATRIZ M ***
139
140 // CODIGO PARA ENVIAR LA MATRIZ M ***
141 }
142
143

```

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

```
1  #include <stdio.h>          //LIBRERIAS BASICAS PARA C
2  #include <stdlib.h>
3  #include <sys/types.h>      //LIBRERIAS PARA LA CREACION DE PROCESOS
4  #include <sys/wait.h>
5  #include <sys/ipc.h>        //LIBRERIAS PARA LA CREACION DE MEMORIA COMPARTIDA
6  #include <sys/shm.h>
7  #include <unistd.h>         //LIBRERIAS VARIAS PARA EL FUNCIONAMIENTO DEL PROGRAMA
8  #include <errno.h>
9  #include <sys/sem.h>
10
11 #define TAM_MEM 400
12
13 void obtencion ();
14 void lock ();
15 void unlock ();
16 void reciboMatrizAB ();
17 void sumar ();
18 void enviarMatrizS ();
19
20 // OBTENER SEMAFORO N
21 key_t key;
22 int semid;
23 struct sembuf sb = {0, -1, 0};
24
25 // RECIBO PARA LAS MATRICES A Y B
26
27 int shmidAB;
28 key_t llaveAB = 5678;
29 char *shmAB, *sAB;
30
31 // ENVIO PARA LA MATRIZ S
32
33 int shmidS;
34 key_t llaveS = 5684;
35 char *shmS, *sS;
36
37 // MATRICES
38
39 int matrizA[10][10], matrizB[10][10], matrizS[10][10];
40
41 // VARIABLES VARIAS
42
43 int aux1 = 0, aux2 = 0, aux3 = 0, aux4 = 0, i = 0, j = 0;
44
45 int main(int argc, char *argv[]){
46
47     obtencion();
48     lock();
49     reciboMatrizAB();
50     sumar();
51     unlock();
52     enviarMatrizS();
53
54     exit(0);
55 }
```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <errno.h>
4  #include <sys/types.h>
5  #include <sys/ipc.h>
6  #include <sys/sem.h>
7
8  int main () {
9
10     key_t key;
11     int semid;
12     union semun{
13         int val;
14         struct semid_ds *buf;
15         unsigned short *array;
16     } arg;
17
18     // CIERRE DEL SEMAFORO PARA PADRE
19
20     if ((key = ftok("Padre.c", 'P')) == -1){
21         perror("ftok");
22         exit(1);
23     }
24     if ((semid = semget(key, 1, 0)) == -1) {
25         perror("semget");
26         exit(1);
27     }
28
29     arg.val = 1;
30     if (semctl(semid, 0, IPC_RMID, arg) == -1) {
31         perror("semctl");
32         exit(1);
33     }
34
35     // CIERRE DEL SEMAFORO PARA HIJO
36
37     if ((key = ftok("Hijo.c", 'H')) == -1){
38         perror("ftok");
39         exit(1);
40     }
41     if ((semid = semget(key, 1, 0)) == -1) {
42         perror("semget");
43         exit(1);
44     }
45
46     arg.val = 1;
47     if (semctl(semid, 0, IPC_RMID, arg) == -1) {
48         perror("semctl");
49         exit(1);
50     }
51
52     // CIERRE DEL SEMAFORO PARA NIETO
53
54     if ((key = ftok("Nieto.c", 'N')) == -1){
55         perror("ftok");
56         exit(1);
57     }
58     if ((semid = semget(key, 1, 0)) == -1) {
59         perror("semget");
60         exit(1);
61     }
62
63     arg.val = 1;
64     if (semctl(semid, 0, IPC_RMID, arg) == -1) {
65
66         perror("semctl");
67         exit(1);
68     }
69
70     return 0;
71 }

```

```
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$
```

```
INVERSAS.txt
~/Escritorio
Abrir 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

```
1 //Program that uses process inter communication to process two matrices with semaphores
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 #include <time.h>
6 #include <stdbool.h>
7 #include <windows.h>
8
9 #define SIZE 10
10 #define MEM_SIZE 4096
11
12 bool inverse(int A[SIZE][SIZE], float inverse[SIZE][SIZE]);
13 void adjoint(int A[SIZE][SIZE], int adj[SIZE][SIZE]);
14 void getCofactor(int A[SIZE][SIZE], int temp[SIZE][SIZE], int p, int q, int n);
15 int determinant(int A[SIZE][SIZE], int n);
16 int Isprime(int *a);
17 void LlenaMatriz(int **matriz);
18 void MatrizSM(int **matriz, int *b);
19
20
21 int main(void)
22 {
23     int i, j, k;
24     int *buffer, *shm;
25     int **matrix1, **matrix2, result_matrix[SIZE][SIZE];
26     float inv[SIZE][SIZE];
27     FILE *write_fp;
28     char *shmid = "SharedMemory";
29     HANDLE hMapFile;
30     STARTUPINFO si;
31     PROCESS_INFORMATION pi;
32     HANDLE hSem1, hSem2;
33
34     ZeroMemory(&si, sizeof(si));
35     si.cb = sizeof(si);
36     ZeroMemory(&pi, sizeof(pi));
37
38     srand((unsigned) time(NULL));
39
40     //Asignacion de espacio de memoria a las matrices
41     matrix1 = malloc(SIZE * sizeof(int *));
42     matrix2 = malloc(SIZE * sizeof(int *));
43     for(i = 0; i < SIZE; i++)
44     {
45         matrix1[i] = malloc(SIZE * sizeof(int));
46         matrix2[i] = malloc(SIZE * sizeof(int));
47     }
48
49     //llenando matrices con valores aleatorios
50     LlenaMatriz(matrix1);
51     LlenaMatriz(matrix2);
52     //Map shared memory
53     if((hMapFile = CreateFileMapping(INVALID_HANDLE_VALUE, NULL, PAGE_READWRITE, 0, MEM_SIZE, shmid)) == NULL)
54     {
55         printf("Failed to map shared memory: (%i)\n", GetLastError());
56         exit(EXIT_FAILURE);
57     }
58     //Create shared memory
59     if((shm = (int *)MapViewOfFile(hMapFile, FILE_MAP_ALL_ACCESS, 0, 0, MEM_SIZE)) == NULL)
60     {
61         printf("Failed to create shared memory: (%i)\n", GetLastError());
62         CloseHandle(hMapFile);
63         exit(EXIT_FAILURE);
64     }
65     //Creando un apuntador a la memoria compartida llamada Buffer
66     buffer = shm;
67
68     //Escribiendo matrices en la memoria compartida
69     MatrizSM(matrix1, buffer);
70     MatrizSM(matrix2, buffer);
71
72     //Creando el primer semaforo
73     if(hSem1 = CreateSemaphore(NULL, 1, 1, "Semaphore1") == NULL)
```

```

72 //Creando el primer semaforo
73 if((hSem1 = CreateSemaphore(NULL, 1, 1, "Semaphore1")) == NULL)
74 {
75     printf("Failed to invoke CreateSemaphore: %d\n", GetLastError());
76     return -1;
77 }
78 //Creando Semaforo 2
79 if((hSem2 = CreateSemaphore(NULL, 1, 1, "Semaphore2")) == NULL)
80 {
81     printf("Failed to invoke CreateSemaphore: %d\n", GetLastError());
82     return -1;
83 }
84
85 //creando Proceso Hijo
86 if(!CreateProcess(NULL, "MatrixSemWChild", NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
87 {
88     printf("Failed to invoke CreateProcess: %d\n", GetLastError());
89     return -1;
90 }
91 else //PADRE
92 {
93     //Halt in order to leave the Child take the control of both semaphores
94     Sleep(1000);
95
96     //Prueba del semaforo
97     WaitForSingleObject(hSem1, INFINITE);
98
99     //Imprimiendo las matrices en consola
100     Imprime(buffer);
101
102     //PADRE VUELVE A TOMAR CONTROL
103     //leyendo la matriz resultante de la multiplicacion, la cual se encuentra almacenada en la memoria compartida
104     //Por lo que almacenamos dichos valores en "result_matrix"
105     k = 2 * SIZE * SIZE;
106     for(i = 0; i < SIZE; i++)
107         for(j = 0; j < SIZE; j++)
108             result_matrix[i][j] = buffer[k++];
109
110     //Calculando la inversa de la multiplicacion y escribiendola en archivo.
111     write_fp = fopen("MultInverse.txt", "w");
112     printf("Escribiendo la Inversa en un archivo\n");
113     if(inverse(result_matrix, inv))
114     {
115         for(i = 0; i < SIZE; i++)
116         {
117             for(j = 0; j < SIZE; j++)
118                 fprintf(write_fp, "%6.2f ", inv[i][j]);
119             fprintf(write_fp, "\n");
120         }
121     }
122     fclose(write_fp);
123
124     //Metiendo en "result_matrix" los valores obtenidos de la suma los cuales estan en la memoria compartida
125     k = 3 * SIZE * SIZE;
126     for(i = 0; i < SIZE; i++)
127         for(j = 0; j < SIZE; j++)
128             result_matrix[i][j] = buffer[k++];
129
130     //Calculando la inversa y Escribiendola en el archivo correspondiente
131     write_fp = fopen("SumaInverse.txt", "w");
132     printf("Enviando la Inversa de la Suma al archivo\n");
133     if(inverse(result_matrix, inv))
134     {
135         for(i = 0; i < SIZE; i++)
136         {
137             for(j = 0; j < SIZE; j++)
138                 fprintf(write_fp, "%6.2f ", inv[i][j]);
139             fprintf(write_fp, "\n");
140         }
141     }
142     fclose(write_fp);
143 }

```

```

146     CloseHandle(pi.hProcess);
147     CloseHandle(pi.hThread);
148     UnmapViewOfFile(shm);
149     CloseHandle(hMapFile);
150 }
151 //Funcion para calcular y almacenar la inversa, regresa valor falso en caso de que
152 //se trate de una matriz singular
153 bool inverse(int A[SIZE][SIZE], float inverse[SIZE][SIZE])
154 {
155     // Encontrando el Determinante de A[][]
156     int det = determinant(A, SIZE);
157     if (det == 0)
158     {
159         printf("Matriz singular, no posee inversa\n");
160         return false;
161     }
162
163     // Buscando la Adjunta
164     int adj[SIZE][SIZE];
165     adjoint(A, adj);
166
167     // Inversa de la matriz = "inversa(A) = adj(A)/det(A)"
168     for (int i=0; i<SIZE; i++)
169         for (int j=0; j<SIZE; j++)
170             inverse[i][j] = adj[i][j]/(float) det;
171
172     return true;
173 }
174 // Funcion para obtener la ADJUNTA de A[N][N] en adj[N][N].
175 void adjoint(int A[SIZE][SIZE], int adj[SIZE][SIZE])
176 {
177     if (SIZE == 1)
178     {
179         adj[0][0] = 1;
180         return;
181     }
182
183     // temp almacena los cofactores de A[][]
184     int sign = 1, temp[SIZE][SIZE];
185
186     for (int i=0; i<SIZE; i++)
187     {
188         for (int j=0; j<SIZE; j++)
189         {
190             // Obteniendo cofactor de A[i][j]
191             getCofactor(A, temp, i, j, SIZE);
192
193             // sign of adj[j][i] positive if sum of row
194             // and column indexes is even.
195             sign = ((i+j)%2==0)? 1: -1;
196
197             // Interchanging rows and columns to get the
198             // transpose of the cofactor matrix
199             adj[j][i] = (sign)*(determinant(temp, SIZE-1));
200         }
201     }
202 }
203
204 // Function to get cofactor of A[p][q] in temp[][]. n is current
205 // dimension of A[][]
206 void getCofactor(int A[SIZE][SIZE], int temp[SIZE][SIZE], int p, int q, int n)
207 {
208     int i = 0, j = 0;
209
210     // Looping for each element of the matrix
211     for (int row = 0; row < n; row++)
212     {
213         for (int col = 0; col < n; col++)
214         {
215             // Copying into temporary matrix only those element
216             // which are not in given row and column
217             if (row != p && col != q)

```

```

ProgramaFinalWindows > C:\MatrizSemW.c > getCofactor(int [SIZE][SIZE], int [SIZE][SIZE], int, int, int)
262 //Funcion que permite realizar la impresion de las matrices escritas en m
263
264 int Imprime(int *a){
265     int i;
266     printf("\nMatriz numero 1\n\n");
267     for(i = 0; i < SIZE * SIZE; i++)
268     {
269         if(i % 10 == 0 && i != 0)
270             printf("\n");
271         printf("%d ", a[i]);
272     }
273     printf("\n");
274
275
276     printf("\nMatriz numero 2\n\n");
277
278     for(i = 0; i < 2 * SIZE * SIZE; i++)
279     {
280         if(i % 10 == 0 && i != 0)
281             printf("\n");
282         printf("%d ", a[i]);
283     }
284     printf("\n");
285
286     printf("\nMatriz Resultante de Multiplicacion\n\n");
287
288     for(i = 0; i < 3 * SIZE * SIZE; i++)
289     {
290         if(i % 10 == 0 && i != 0)
291             printf("\n");
292         printf("%d ", a[i]);
293     }
294     printf("\n");
295
296     printf("\nMatriz Resultante de Suma\n\n");
297
298     for(i = 0; i < 4 * SIZE * SIZE; i++)
299     {
300         if(i % 10 == 0 && i != 0)
301             printf("\n");
302         printf("%d ", a[i]);
303     }
304     printf("\n");
305 }
306
307 //Funcion para llenar de valores aleatorios la matriz
308
309 void LlenaMatriz(int **matriz){
310     int i,j;
311     for(i = 0; i < SIZE; i++)
312     {
313         for(j = 0; j < SIZE; j++)
314         {
315             matriz[i][j] = rand() % 5;
316         }
317     }
318 }
319 // Funcion para escribir los valores en la memoria compartida
320
321 void MatrizSM(int **matriz, int *b){
322
323     int i,j,k;
324     k = 0;
325     for(i = 0; i < SIZE; i++)
326     {
327         for(j = 0; j < SIZE; j++)
328             b[k++] = matriz[i][j];
329
330         for(i = 0; i < SIZE; i++)
331         {
332             for(j = 0; j < SIZE; j++)
333                 b[k++] = matriz[i][j];
334         }
335     }
336 }

```

Hijo.c

```
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <time.h>
6  #include <stdbool.h>
7  #include <windows.h>
8
9  #define SIZE 10
10 #define MEM_SIZE 400
11
12 int main(void)
13 {
14     int matrix1[SIZE][SIZE], matrix2[SIZE][SIZE], result_matrix[SIZE][SIZE], i, j, k, accumulator;
15     int *buffer, *shm;
16     char *shmid = "SharedMemory";
17     HANDLE hMapFile, hSem1, hSem2;
18     PROCESS_INFORMATION piChild;
19     STARTUPINFO siChild;
20
21     //Abriendo Semaforo 1
22     if((hSem1 = OpenSemaphore(SEMAPHORE_ALL_ACCESS, FALSE, "Semaphore1")) == NULL)
23     {
24         printf("Failed to invoke OpenSemaphore1: %d\n", GetLastError());
25         return -1;
26     }
27     //Abriendo Semaforo 2
28     if((hSem2 = OpenSemaphore(SEMAPHORE_ALL_ACCESS, FALSE, "Semaphore2")) == NULL)
29     {
30         printf("Failed to invoke OpenSemaphore2: %d\n", GetLastError());
31         return -1;
32     }
33     ZeroMemory(&siChild, sizeof(siChild));
34     siChild.cb = sizeof(siChild);
35     ZeroMemory(&piChild, sizeof(piChild));
36
37     //Open file mapping
38     if((hMapFile = OpenFileMapping(FILE_MAP_ALL_ACCESS, FALSE, shmid)) == NULL)
39     {
40         printf("Failed to open file of mapping from shared memory: (%i)\n", GetLastError());
41         exit(EXIT_FAILURE);
42     }
43     //Access/Attach to shared memory
44     if((shm = (int *)MapViewOfFile(hMapFile, FILE_MAP_ALL_ACCESS, 0, 0, MEM_SIZE)) == NULL)
45     {
46         printf("Failed to access to shared memory: (%i)\n", GetLastError());
47         CloseHandle(hMapFile);
48         exit(EXIT_FAILURE);
49     }
50     //Apuntador a memoria compartida
51     buffer = shm;
52
53     //Creacion de Proceso Hijo
54     if(!CreateProcess(NULL, "MatrixSemWGrandChild", NULL, NULL, FALSE, 0, NULL, NULL, &siChild, &piChild))
55     {
56         printf("Failed to create child process\n");
57         exit(EXIT_FAILURE);
58     }
59     //Test control of semaphore 1
60     WaitForSingleObject(hSem1, INFINITE);
61     //Test control of semaphore 2
62     WaitForSingleObject(hSem2, INFINITE);
63     //CRITIC SECTION
64     //Copiando en las matrices los valores almacenados en la memoria compartida
65     k = 0;
66     for(i = 0; i < SIZE; i++)
67         for(j = 0; j < SIZE; j++)
68             matrix1[i][j] = buffer[k++];
69
70     for(i = 0; i < SIZE; i++)
71         for(j = 0; j < SIZE; j++)
72             matrix2[i][j] = buffer[k++];
73 }
```

```

50 //Apuntador a memoria compartida
51 buffer = shm;
52
53 //Creacion de Proceso Hijo
54 if(!CreateProcess(NULL, "MatrixSemWGrandChild", NULL, NULL, FALSE, 0, NULL, NULL, &siChild, &piChild))
55 {
56     printf("Failed to create child process\n");
57     exit(EXIT_FAILURE);
58 }
59 //Test control of semaphore 1
60 WaitForSingleObject(hSem1, INFINITE);
61 //Test control of semaphore 2
62 WaitForSingleObject(hSem2, INFINITE);
63 //CRITIC SECTION
64 //Copiando en las matrices los valores almacenados en la memoria compartida
65 k = 0;
66 for(i = 0; i < SIZE; i++)
67     for(j = 0; j < SIZE; j++)
68         matrix1[i][j] = buffer[k++];
69
70 for(i = 0; i < SIZE; i++)
71     for(j = 0; j < SIZE; j++)
72         matrix2[i][j] = buffer[k++];
73
74 //Haciendo la multiplicacion
75 for(j = 0; j < SIZE; j++)
76 {
77     for(k = 0; k < SIZE; k++)
78     {
79         accumulator = 0;
80         for(int l = 0, m = 0; l < SIZE && m < SIZE; l++, m++)
81             accumulator += matrix1[j][l] * matrix2[m][k];
82         result_matrix[j][k] = accumulator;
83     }
84 }
85 //Escribiendo resultado de la multiplicacion en la memoria compartida
86 k = 2 * SIZE * SIZE;
87 for(i = 0; i < SIZE; i++)
88     for(j = 0; j < SIZE; j++)
89         buffer[k++] = result_matrix[i][j];
90
91 //Liberando Senaforo 2
92 if(!ReleaseSemaphore(hSem2, 1, NULL))
93 {
94     printf("Failed to invoke ReleaseSemaphore2: %d\n", GetLastError());
95 }
96
97 //Esperando la finalizacion del nieto, es necesario para no dejar al semaforo 1 abandonado
98 WaitForSingleObject(piChild.hProcess, INFINITE);
99
100 //Liberando Semaforo 1
101 if(!ReleaseSemaphore(hSem1, 1, NULL))
102 {
103     printf("Failed to invoke ReleaseSemaphore1: %d\n", GetLastError());
104 }
105 CloseHandle(piChild.hThread);
106 CloseHandle(piChild.hProcess);
107 UnmapViewOfFile(shm);
108 CloseHandle(hMapFile);
109
110 return 0;
111 }

```

Nieto.c

```
1  //Program that uses process inter communication to process two matrices
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <time.h>
6  #include <stdbool.h>
7  #include <windows.h>
8
9  #define SIZE 10
10 #define MEM_SIZE 400
11
12 int main(void)
13 {
14     int matrix1[SIZE][SIZE], matrix2[SIZE][SIZE], result_matrix[SIZE][SIZE], i, j, k;
15     int *buffer, *shm;
16     char *shmid = "SharedMemory";
17     HANDLE hMapFile, hSem2;
18
19     //Abre Semaforo 2
20     if((hSem2 = OpenSemaphore(SEMAPHORE_ALL_ACCESS, FALSE, "Semaphore2")) == NULL)
21     {
22         printf("Failed to invoke OpenSemaphore2: %d\n", GetLastError());
23         return -1;
24     }
25     //Open file mapping
26     if((hMapFile = OpenFileMapping(FILE_MAP_ALL_ACCESS, FALSE, shmid)) == NULL)
27     {
28         printf("Failed to open file of mapping from shared memory: (Xi)\n", GetLastError());
29         exit(EXIT_FAILURE);
30     }
31     //Access/Attach to shared memory
32     if((shm = (int *)MapViewOfFile(hMapFile, FILE_MAP_ALL_ACCESS, 0, 0, MEM_SIZE)) == NULL)
33     {
34         printf("Failed to access to shared memory: (Xi)\n", GetLastError());
35         CloseHandle(hMapFile);
36         exit(EXIT_FAILURE);
37     }
38     //Apuntador a Memoria Compartida
39     buffer = shm;
40
41     //Copiando en las matrices los valores almacenados en la memoria compartida
42     k = 0;
43     for(i = 0; i < SIZE; i++)
44         for(j = 0; j < SIZE; j++)
45             matrix1[i][j] = buffer[k++];
46
47     for(i = 0; i < SIZE; i++)
48         for(j = 0; j < SIZE; j++)
49             matrix2[i][j] = buffer[k++];
50
51     //Realizando la suma
52     for(i = 0; i < SIZE; i++)
53         for(j = 0; j < SIZE; j++)
54             result_matrix[i][j] = matrix1[i][j] + matrix2[i][j];
55
56     //Detenido hasta que la multiplicacion esta hecha
57     Sleep(1000);
58
59     //Test control of semaphore 2
60     WaitForSingleObject(hSem2, INFINITE);
61     //Escribiendo la suma dentro de la memoria compartida
62     k = 3 * SIZE * SIZE;
63     for(i = 0; i < SIZE; i++)
64         for(j = 0; j < SIZE; j++)
65             buffer[k++] = result_matrix[i][j];
66
67     //Liberando al Semaforo 2
68     if(!ReleaseSemaphore(hSem2, 1, NULL))
69     {
70         printf("Falla al invocar ReleaseSemaphore: %d\n", GetLastError());
71     }
72     UnmapViewOfFile(shm);
73     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

D:\C:\Users\TOMY\AVIA\Google Drive\Escuelas\College\CuantoSemestre\Sistemas Operativos\Practi



MultInverse



SumaInverse