INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

SISTEMAS OPERATIVOS

Práctica #6. Mecanismos de sincronización de procesos en Linux y Windows(semáforos)

INTEGRANTES DEL EQUIPO:

COLIN RAMIRO JOEL
HERNÁNDEZ REYES JULIO CÉSAR
MALDONADO CERÓN CARLOS
MENDOZA GARCÍA ELIÚ EDUARDO

Grupo: 4CM1

PROFESOR: CORTÉS GALICIA JORGE

05 Diciembre, 2021

I.INTRODUCCIÓN TEÓRICA

Un proceso, se puede representar como una abstracción que hace referencia a cada caso de ejecución de un programa. Un proceso no tiene porqué estar siempre en ejecución y la vida de un proceso pasa por varias fases, incluyendo la de su ejecución.

En muchos casos, los procesos se reúnen para realizar tareas en conjunto, a este tipo de relación se le llama procesos cooperativos. Para lograr la comunicación, los procesos deben sincronizarse, de no ser así pueden ocurrir problemas no deseados. La sincronización es la transmisión y recepción de señales que tiene por objeto llevar a cabo el trabajo de un grupo de procesos cooperativos.

Ahora. ya teniendo una idea de lo que es un proceso, se define lo que es la sincronización, y es que la sincronización no es nada más que el funcionamiento coordinado de una tarea encomendada. En otras palabras, es la coordinación y cooperación de un conjunto de procesos para asegurar la comparación de recursos de cómputo.

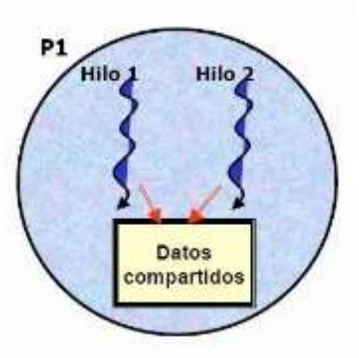
Esta acción es necesaria para prevenir y corregir errores de sincronización debido al acceso concurrente a recursos compartidos, tal como las estructuras de datos o dispositivos de E/S de procesos. La sincronización entre procesos permite intercambiar señales de tiempo.

Para que estos procesos puedan sincronizarse es necesario disponer de servicios que permitan bloquear o suspender bajo determinadas circunstancias la ejecución de un proceso.

Los distintos hilos de ejecución comparten una serie de recursos tales como:

- Espacio de Memoria
- Archivos Abiertos
- Situación de Autenticación

Esta técnica permite simplificar el diseño de una aplicación que debe de llevar a cabo distintas funciones simultáneamente. Un hilo modifica un dato en la memoria, los otros hilos restantes, acceden a este dato modificado de manera inmediata.



Los diferentes S.O. ofrecen una gran variedad de mecanismos de sincronización dentro de los cuales destacan:

- Señales
- Tuberías
- Semáforos
- Mutex y Variables Condicionales
- Paso de Mensajes

II.DESARROLLO EXPERIMENTAL

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

semget()

NOMBRE

semget - obtiene el identificador de un conjunto de semáforos

DESCRIPCIÓN

Esta función devuelve el identificador del conjunto de semáforos asociado con el argumento key. Un nuevo conjunto de nsems semáforos se crea si key tiene el valor IPC_PRIVATE, o si no hay un conjunto de semáforos asociado a key y el bit IPC_CREAT vale 1 en semflg (p.ej. semflg & IPC_CREAT es distinto de cero).
La presencia en semflg de los campos IPC_CREAT e IPC_EXCL tiene el mismo papel, con respecto a la existencia del conjunto de semáforos, que la presencia de O_CREAT y O_EXCL en el argumento mode de la llamada del sistema open(2): p.ej., la función semget falla si semflg tiene a 1 tanto IPC_CREAT como IPC_EXCL y ya existe un conjunto de semáforos para key.

Acerca de la creación, los 9 bits bajos del argumento semflg definen los permisos de acceso (para el propietario, grupo y otros) para el conjunto de semáforos. Estos bits tienen el mismo formato, y el mismo significado que el argumento de modo en las llamadas al sistema open(2) o creat(2) (aunque los permisos de ejecución no son significativos para los semáforos, y los permisos de escritura significan permisos para alterar los valores del semáforo).

Cuando se crea un nuevo conjunto de semáforos, semget inicializa la estructura de datos semid_ds asociada al conjunto de semáforos como sigue:

Se pone el ID de usuario efectivo del proceso que realiza la llamada en sem_perm.cuid y sem_perm.uid
Se pone el ID de grupo efectivo del proceso que realiza la llamada en sem_perm.cgid y sem_perm.gid
Los 9 bits más bajos de sem_perm.mode se ponen como los 9 bits más bajos de semflg.
Se pone el valor de nsems en sem_nsems.
sem_otime se pone a 0.
Se pone la hora actual en sem_ctime.

El argumento nsems puede ser 0 (un comodín o valor sin importancia) cuando no se está creando un conjunto de semáforos. En otro caso nsems debe ser mayor que 0 y menor o igual que el número máximo de semáforos por conjunto de semáforos, (SEMMSL).
Si el conjunto de semáforos ya existe, los permisos de acceso son verificados.

VALOR DEVUELTO

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

semop()

NOMBRE

semop - operaciones con semáforos

DESCRIPCIÓN

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 */
```

Banderas reconocidas en sem_flg son IPC_NOWAIT y SEM_UNDO. Si una operación ejecuta SEM_UNDO, sera deshecha cuando el proceso finalice.

El conjunto de operaciones contenido en sops se realiza de forma atómica, es decir, las operaciones son llevadas a cabo al mismo tiempo, y sólo si pueden ser realizadas simultáneamente. El comportamiento de la llamada al sistema en caso de que no todas las operaciones puedan realizarse inmediatamente depende de la presencia de la bandera IPC_NOWAIT en los campos sem_flg individuales, como se ve más abajo.

Cada operación es ejecutada en el semáforo numero sem_num donde el primer semáforo del conjunto es el semáforo 0. Hay tres tipos de operación, que se distinguen por el valor de sem_op.

Si sem_op es un entero positivo, la operación añade este valor al valor del semáforo (semval). Además, si SEM_UNDO es invocado para esta operación, el sistema actualiza el contador del proceso para operaciones "undo" (semadj) para este semáforo. La operación siempre puede ejecutarse - nunca fuerza a un proceso a esperar. El proceso invocador debe tener permisos de modificación sobre el conjunto de semáforos.

Si sem_op es cero, el proceso debe tener permiso de lectura en el semáforo. Esta es una operación "espera-por-cero": si semval es cero, la operación puede ejecutarse inmediatamente. Por otra parte, si IPC_NOWAIT es invocado en sem_flg, la llamada al sistema falla con la variable errno fijada a EAGAIN . (y ninguna de las operaciones sops se realiza.) En otro caso semzcnt (el número de procesos esperando hasta que el valor del semáforo sea cero) es incremetada en uno y el proceso duerme hasta que algo de lo siguiente ocurra:

- semval es 0, instante en el que el valor de semzcnt es decrementado.
- El semáforo es eliminado: la llamada al sistema falla con error fijada a EIDRM.
- El proceso que lo invoca captura una señal: el valor de semzcnt es decrementado y la llamada al sistema falla con error fijada a EINTR.

Si sem_op es menor que cero, el proceso debe tener los permisos de modificación sobre el semáforo. Si semval es mayor que o igual que el valor absoluto de sem_op, la operación puede ejecutarse inmediatamente: el valor absoluto de sem_op es restado a semval. y, si SEM_UNDO es invocado para esta operación, el sistema actualiza el contador "undo" del proceso (semadj) para este semáforo. Si el valor absoluto de sem_op es mayor que semval, y IPC_NOWAIT está presente en sem_flg, la llamada al sistema falla con errno fijado a EAGAIN. (y ninguna de las operaciones sops se

realiza.) En otro caso semncnt (el número de procesos esperando a que se incremente el valor de este semáforo) es incrementado en uno y el proceso duerme hasta que ocurra:

- semval sea mayor o igual que el valor absoluto de sem_op, en cuyo instante el valor de semncnt es decrementado, el valor absoluto de sem_op es restado de semval y, si SEM_UNDO es invocado para esta operación, el sistema actualiza el contador "undo" del proceso (semadj) para este semáforo.
- El semáforo es es eliminado del sistema: la llamada al sistema falla con errno fijado a EIDRM.
- El proceso invocador captura una señal: el valor de semncnt es decrementado y la llamada al sistema falla con errno fijado a EINTR.

En caso de que haya funcionado, el valor sempid para cada semáforo especificado en el array apuntado por sops es fijado al identificador del proceso del proceso invocador. Además sem_otime es puesto a la hora actual.

VALOR DEVUELTO

Si todo ha sido correcto la llamada al sistema devuelve 0, en otro caso devuelve -1 con errno indicando el error.

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

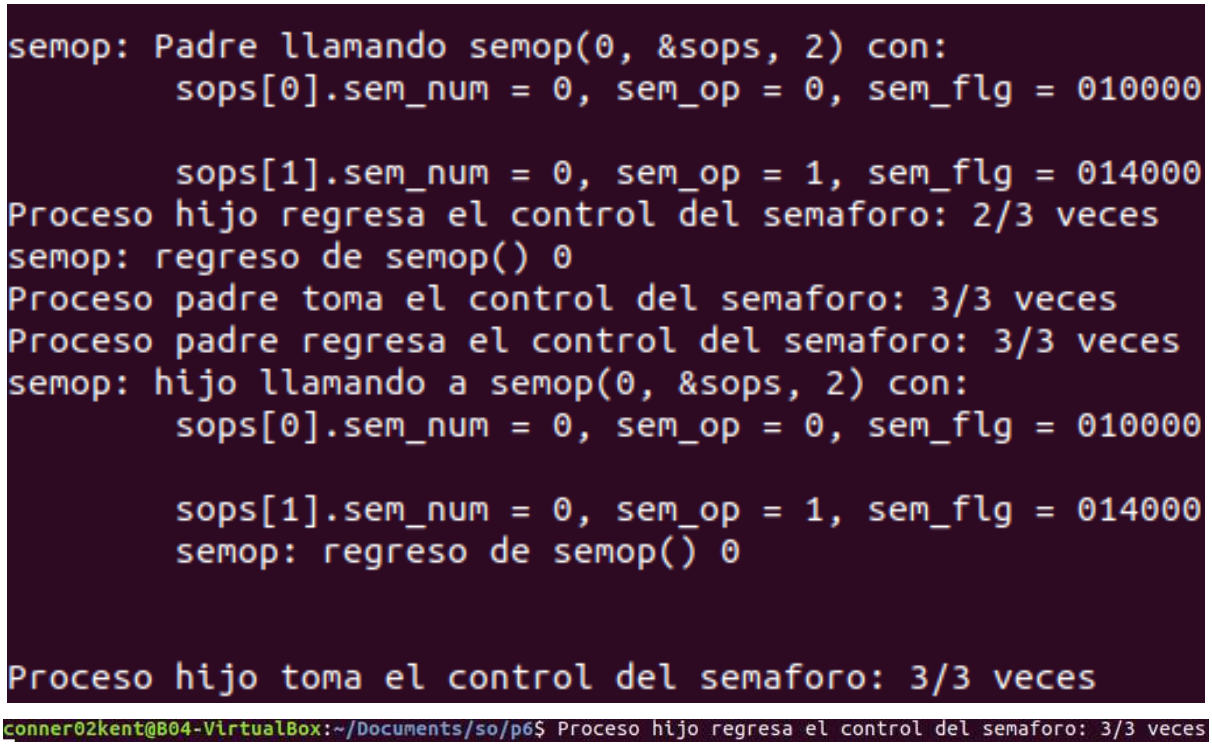
CAPTURA DEL CÓDIGO

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdlib.h>
#include <unistd.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;
        }
    }
}
```

COMPILACIÓN Y EJECUCIÓN

```
conner02kent@B04-VirtualBox:~/Documents/so/p6$ gcc p6_2.c
conner02kent@B04-VirtualBox:~/Documents/so/p6$ ./a.out
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

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: 1/3 veces
semop: regreso de semop() 0
Proceso padre toma el control del semaforo: 2/3 veces
Proceso padre regresa 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
        semop: regreso de semop() 0


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



Para que el programa compilara sin advertencias, fue necesario agregar las librerías unistd.h y stdlib.h. El programa que se nos proporcionó, resulta sencillo de entender desde analizar el código y al ser ejecutado. Podemos observar la creación del semáforo, y cómo en la ejecución, el proceso hijo toma el control de este, y entre el proceso padre e hijo se alterna el control del semáforo, 3 veces cada uno. Es interesante observar cómo para que un proceso tome el control del semáforo, este deber ser regresado por el proceso anterior que lo estaba usando.

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

Programa del Proceso Padre

padre.c hijo.c

```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);
}
```

Programa del Proceso Hijo

padre.c hijo.c

```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++;
 }
}
```

Se compilaron y ejecutaron los archivos y el resultado fue el siguiente:

Tenemos un semáforo donde el proceso padre va a entrar y liberar al semáforo 3 veces cada cosa lo mismo para el proceso hijo eso como el total pero su proceso va intercalado van entrando y liberando hasta que al final los dos procesos se hayan liberado 3 de 3.

```
C:\Users\Carlos\Desktop>padre.exe hijo.exe
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 hijo entrando 2 de 3 veces al semaforo
Soy el padre liberando 2 de 3 veces al semaforo
Soy el padre entrando 3 de 3 veces al semaforo
Soy el hijo liberando 2 de 3 veces al semaforo
Soy el hijo entrando 3 de 3 veces al semaforo
Soy el padre liberando 3 de 3 veces al semaforo
Soy el hijo liberando 3 de 3 veces al semaforo

C:\Users\Carlos\Desktop>
```

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

VERSIÓN EN LINUX:

punto4.c

```
/*
    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 como para la lectura de las memorias
    compartidas. Úselos en los lugares donde haya necesidad de sincronizar el acceso a memoria compartida.


    -> 7.- PRACTICA 5
    Programe nuevamente la aplicación del punto cuatro utilizando en esta ocasión memoria compartida en
    lugar de tuberías (utilice tantas memorias compartidas como requiera). Programe esta aplicación para
    Linux como para Windows utilizando la memoria compartida de cada sistema operativo.

    -> 4.- PRACTICA 5
    Programe una aplicación que cree un proceso hijo a partir de un proceso padre, el proceso padre enviará
    al proceso hijo, a travéz de una tuberia, dos matrices de 10 x 10 a multiplicar por parte del hijo,
    mientras tanto el proceso hijo creará un hijo de él, al cual enviará dos matrices de 10 x 10 a sumar en
    el proceso hijo creado, nuevamente el envio de estos valores será a través de una tubería. Una vez
    calculando el resultado de la suma, el proceso hijo del hijo devolverá la matriz resultante a su abuelo
    (via tuberiá). A su vez, el proceso hijo devolverá la matriz resultante de la multiplicación que realizó
    a su padre. Finalmente, el proceso padre obtendra la matriz inversa de cada una de las matrices recibidas
    y el resultado lo guardará en un archivo para cada matriz inversa obtenida.

    RESUMEN
    Proceso PADRE:
        -crear un proceso HIJO
        -enviar al HIJO dos matrices 10x10 para que se multipliquen
                -Recibir la matriz resultante de la multiplicación
                -Recibir la matriz resultante de la suma
                    -Obtener la matriz inversa de la matriz recibida del NIETO
                        -Guardar el resultado en un archivo de texto
                    -Obtener la matriz inversa de la matriz recibida del HIJO
                        -Guardar el resultado en un archivo de texto
    Proceso HIJO:
        -Recibe dos matrices 10x10 de su PADRE y las multiplica
        -Crear un proceso HIJO
            -Enviar al HIJO dos matrices 10x10 para que se sumen
                -Regresar la matriz resulatante de la multiplicacion a su PADRE
    Proceso HIJO DEL HIJO:
            -Recibe dos matrices 10x10 de su PADRE y las suma
                -Regresar la matriz resultante e la suma al ABUELO
*/
```

```
//pid == 0  <- Proceso Hijo
//pid < 0  <- Proceso Padre
//pid == -1 <-Error>

#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#include <time.h>
#include <stdlib.h>
#include <sys/shm.h>
#include <unistd.h>
#include <string.h>


const int FILAS = 10;
const int COLUMNAS = 10;
const int TAM_NMEM = 120*sizeof(int);
```

```
62
63  //memoria compartida para matrices
64  void llenarmatriz(int matriz[][COLUMNAS]);
65  void imprimirmatriz(int matriz[][COLUMNAS]);
66  void matrizatexto(int matriz[][COLUMNAS], char* cad);
67  void cadenaAentero(char* cad,int matriz[][COLUMNAS]);
68  void concatenar(char* cad, char caracter);
69  //operaciones con matrices
70  void imprimirmatrizres(int m[][COLUMNAS]);
71  void imprimirmatriz_float(float m[][COLUMNAS]);
72  void imprimirmatrizres_float(float m[][COLUMNAS]);
73  void multiplicacion(int m1[][COLUMNAS],int m2[][COLUMNAS],int mr[][COLUMNAS]);
74  void suma(int m1[][COLUMNAS],int m2[][COLUMNAS],int mr[][COLUMNAS]);
75  void copiarmatriz(int m1[][COLUMNAS],float mr[][COLUMNAS]);
76  void inversa(float matrix[][COLUMNAS]);
77  void guardarmulti(int matrix[][COLUMNAS]);
78  void guardarmultiinversa(float matrix[][COLUMNAS]);
79  void guardarsuma(int matrix[][COLUMNAS]);
80  void guardarsumainversa(float matrix[][COLUMNAS]);
81  void leermultiplicacion(int matrix[][COLUMNAS]);
82  //void leersuma(int matrix[][COLUMNAS]);
83  void cadenaAmatriz(char* cad,int matriz[][COLUMNAS]);
84  void leersuma(int matrix[][COLUMNAS]);
85
```

```
86  int main(void){
87      srand(time(NULL));
88
89      int i,j;
90      pid_t pid;
91      int semid; //Se usa para guardar el identificador del sistema operativo va a devolver cuando se crea el semaf
92      key_t llave = 1234; //Similar a memoria compartida, para un semaforo se requiere una variable que identifique
93      int semban = IPC_CREAT | 0666; //Esta variable va a almacenar las banderas para la creación del semaforo
94      int nsems = 1; //Esta variable nos indica cuantos semaforos vamos a crear en el grupo
95      int nsops; //Nos indica el numero de estructuras que vamos a utilizar con una operación del semaforo
96      struct sembuf *sops = (struct sembuf *) malloc(2*sizeof(struct sembuf)); //sops  es del tipo struct sembuf, p
97      //Comienza la ejecución del programa
98      printf("Iniciando semaforo...\n");
99      //semget(llave del semaforo, numero de semaforos, banderas para la creación del semaforo)
100     if ((semid = semget(llave, nsems, semban)) == -1) { //usando semget se crea el grupo de semaforos a usar
101         perror("semget: error al iniciar semaforo");
102         exit(1);
103     }
104     else
105         printf("Semaforo iniciado...\n");
106     if ((pid = fork()) < 0) {
107         perror("fork: error al crear proceso\n");
108         exit(1);
109     }
110     if (pid == 0) {
111         //PROCESO HIJO
112         i = 0;
113         while (i < 1) { //Las 3 veces que el proceso va a entrar a la seccion critica
114             nsops = 2; //numero de estructuras de tipo sembuf con las que se trabajaran las operaciones con el se
115             // la estructura sops tiene tres mienbros
116             // sen_num, sem_op, sem_flg
117             sops[0].sem_num = 0; //numero del semaforo dentro del grupo sobre el cual se va a aplicar esta estruc
118             sops[0].sem_op = 0; //nos indica como vamos a trabajar la operación del semaforo que se manipulara co
119             sops[0].sem_flg = SEM_UNDO; //las banderas para la estructura en las operaciones
120             //SEM_UNDO se utiliza para indicar que la estructura dentro de la operacion se debe de llevar de mane
121             //regresando al estado estable del semaforo)
122             sops[1].sem_num = 0;
123             sops[1].sem_op = 1; //es para cambiar el valor del semaforo cuando el semaforo esta ocupado
124             sops[1].sem_flg = SEM_UNDO | IPC_NOWAIT;
125
126             if ((i = semop(semid, sops, nsops)) == -1) {
```

```
127                 perror("semop: error en operacion del semaforo\n");
128             }
129             else {
130                 printf("\n\nProceso hijo toma el control del semaforo.\n");
131                 printf("Soy el hijo(%d, hijo de %d)\n",getpid(), getppid());
132                 printf("El hijo esta en la sección critica\n");
133                 {
134                     //Matriz 1
135                     int matriz1[FILAS][COLUMNAS];
136
137                     int shmidM1;   //Para almacenar identificador de la memoria compartida
138                     key_t llaveM1; //Valor numerico para identificar a la memoria compartida
```

```
139                 char *shmM1, *sM1;
140                 llaveM1 = 100; // Valor entero
141                 if((shmidM1 = shmget(llaveM1,TAM_NMEM,0666)) < 0){
142                     perror("Error al obtener memoria compartida: shmget");
143                     exit(-1);
144                 }
145                 if((shmM1 = shmat(shmidM1,NULL,0)) == (char*)-1){
146                     perror("Error al enlazar la memoria compartida; shmat");
147                     exit(-1);
148                 }
149 
150                 char cadena1[250] = "";
151                 sM1 = shmM1;
152                 strcpy(cadena1,sM1);
153                 //printf("String: \n%s\n\n",sM1);
154                 //printf("Cadena: \n%s\n\n",cadena);
155                 //for(sM1 = shmM1; *sM1 != '\0';sM1++)
156                 //  putchar(*sM1);
157 
158                 cadenaAentero(cadena1,matriz1);
159                 printf("Matriz 1 recibida:\n");
160                 imprimirmatriz(matriz1);
161                 //*shmM1 = '*';
162 
163                 //Matriz 2
164                 int matriz2[FILAS][COLUMNAS];
165 
166                 int shmidM2;   //Para almacenar identificador de la memoria compartida
167                 key_t llaveM2; //Valor numerico para identificar a la memoria compartida
168                 char *shmM2, *sM2;
169                 llaveM2 = 200; // Valor entero
170                 if((shmidM2 = shmget(llaveM2,TAM_NMEM,IPC_CREAT|0666)) < 0){
171                     perror("Error al obtener memoria compartida: shmget");
172                     exit(-1);
173                 }
174                 if((shmM2 = shmat(shmidM2,NULL,0)) == (char*)-1){
175                     perror("Error al enlazar la memoria compartida; shmat");
176                     exit(-1);
177                 }
178                 char cadena2[250] = "";
179                 sM2 = shmM2;
180                 strcpy(cadena2,sM2);
181                 //printf("String2: \n%s\n\n",sM2);
182 
183                 cadenaAentero(cadena2,matriz2);
184                 printf("Matriz 2 recibida:\n");
185                 imprimirmatriz(matriz2);
186 
187                 int mr[FILAS][COLUMNAS];
188                 multiplicacion(matriz1,matriz2,mr); //Se hace la multiplicacion
189                 guardarmulti(mr);
190                 printf("Multiplicación:\n");
191                 imprimirmatriz(mr);
192                 //*shmM2 = '*';
193                 //exit(0);
194             }
195              // Aqui es para simular lo que el proceso hijo estaria ejecutando como parte de su seccion criti
196             //un recurso compartido, etc.
197             //Antes de salir el proceso debe de regresar al semaforo a su estado de desocupado:
198             nsops = 1;
199             sops[0].sem_num = 0;
200             sops[0].sem_op = -1;
201             sops[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
202             if ((j = semop(semid, sops, nsops)) == -1) {
203                 perror("semop: error en operacion del semaforo\n");
204             }
205             else
206                 printf("Proceso hijo regresa el control del semaforo.\n");
207                 printf("\n");
208                 printf("\n");
209                 {
210                     pid_t pid2;
211                     if((pid2 = fork()) == 0){
212                         //PROCESO NIETO
213                         printf("Soy el nieto(%d, hijo de %d)\n",getpid(), getppid());
214                         //Matriz 1
215                         int matriz1[FILAS][COLUMNAS];
```

```
216
217                         int shmidM1;   //Para almacenar identificador de la memoria compartida
218                         key_t llaveM1; //Valor numerico para identificar a la memoria compartida
219                         char *shmM1, *sM1;
220                         llaveM1 = 100; // Valor entero
221                         if((shmidM1 = shmget(llaveM1,TAM_NMEM,0666)) < 0){
222                             perror("Error al obtener memoria compartida: shmget");
223                             exit(-1);
224                         }
225                         if((shmM1 = shmat(shmidM1,NULL,0)) == (char*)-1){
226                             perror("Error al enlazar la memoria compartida; shmat");
227                             exit(-1);
228                         }
229
230                         char cadena1[250] = "";
231                         sM1 = shmM1;
232                         strcpy(cadena1,sM1);
233                         //printf("String: \n%s\n\n",sM1);
234                         //printf("Cadena: \n%s\n\n",cadena);
235                         //for(sM1 = shmM1; *sM1 != '\0';sM1++)
236                         //  putchar(*sM1);
237
238                         cadenaAentero(cadena1,matriz1);
239                         printf("Matriz 1 recibida:\n");
240                         imprimirmatriz(matriz1);
241                         //*shmM1 = '*';
242
243                         //Matriz 2
244                         int matriz2[FILAS][COLUMNAS];
245
246                         int shmidM2;   //Para almacenar identificador de la memoria compartida
247                         key_t llaveM2; //Valor numerico para identificar a la memoria compartida
248                         char *shmM2, *sM2;
249                         llaveM2 = 200; // Valor entero
250                         if((shmidM2 = shmget(llaveM2,TAM_NMEM,IPC_CREAT|0666)) < 0){
251                             perror("Error al obtener memoria compartida: shmget");
252                             exit(-1);
253                         }
254                         if((shmM2 = shmat(shmidM2,NULL,0)) == (char*)-1){
255                             perror("Error al enlazar la memoria compartida; shmat");
256                             exit(-1);
257                         }
258                         char cadena2[250] = "";
259                         sM2 = shmM2;
260                         strcpy(cadena2,sM2);
261                         //printf("String2: \n%s\n\n",sM2);
262
263                         cadenaAentero(cadena2,matriz2);
264                         printf("Matriz 2 recibida:\n");
265                         imprimirmatriz(matriz2);
266
267                         int mr[FILAS][COLUMNAS];
268                         suma(matriz1,matriz2,mr); //Se hace la suma
269                         guardarsuma(mr);
270                         printf("Suma:\n");
271                         imprimirmatriz(mr);
272                         printf("\n");
273                         printf("\n");
274
275                     }
276
277                 }
278                 sleep(5);
279             }
280             ++i;
281         }
282     }
283     //PROCESO PADRE
284     else {
285         i = 0;
286         while (i < 1) {
287             nsops = 2;
288             sops[0].sem_num = 0;
289             sops[0].sem_op = 0;
290             sops[0].sem_flg = SEM_UNDO;
291
292             sops[1].sem_num = 0;
293             sops[1].sem_op = 1;
```

```
294             sops[1].sem_flg = SEM_UNDO | IPC_NOWAIT;
295
296             if ((j = semop(semid, sops, nsops)) == -1) {
297                 perror("semop: error en operacion del semaforo\n");
298             }else {
299                 printf("Proceso padre toma el control del semaforo.\n");
300                 printf("Soy el padre(%d, hijo de %d)\n",getpid(), getppid());
301                 printf("El padre esta en la sección critica\n");
302                 {
303                     //Matriz 1
304
305                     int matriz1[FILAS][COLUMNAS];
306                     llenarmatriz(matriz1);
307                     printf("Matriz 1 generada:\n");
308                     imprimirmatriz(matriz1);
309
310                     int shmidM1;   //Para almacenar identificador de la memoria compartida
311                     key_t llaveM1; //Valor numerico para identificar a la memoria compartida
312                     char *shmM1, *sM1;
313                     llaveM1 = 100; // Valor entero
314                     if((shmidM1 = shmget(llaveM1,TAM_NMEM,IPC_CREAT|0666)) < 0){
315                         perror("Error al obtener memoria compartida: shmget");
316                         exit(-1);
317                     }
318                     if((shmM1 = shmat(shmidM1,NULL,0)) == (char*)-1){
319                         perror("Error al enlazar la memoria compartida; shmat");
320                         exit(-1);
321                     }
322
323                     sM1 = shmM1;
324                     matrizatexto(matriz1,sM1);
325                     //for(sM1 = shmM1; *sM1 != '\0';sM1++)
326                     //  putchar(*sM1);
327                     //putchar('\n');
328
329                     //while(*shmM1!='*')
330                         //sleep(1); //El cliente es el que escribe el a
331                     //exit(0);
332
333                     //Matriz 2
```

```
334
335                     int matriz2[FILAS][COLUMNAS];
336                     llenarmatriz(matriz2);
337                     printf("Matriz 2 generada:\n");
338                     imprimirmatriz(matriz2);
339
340                     int shmidM2;   //Para almacenar identificador de la memoria compartida
341                     key_t llaveM2; //Valor numerico para identificar a la memoria compartida
342                     char *shmM2, *sM2;
343                     llaveM2 = 200; // Valor entero
344                     if((shmidM2 = shmget(llaveM2,TAM_NMEM,IPC_CREAT|0666)) < 0){
345                         perror("Error al obtener memoria compartida: shmget");
346                         exit(-1);
347                     }
348                     if((shmM2 = shmat(shmidM2,NULL,0)) == (char*)-1){
349                         perror("Error al enlazar la memoria compartida; shmat");
350                         exit(-1);
351                     }
352
353                     sM2 = shmM2;
354                     matrizatexto(matriz2,sM2);
355
356                     //for(sM2 = shmM2; *sM2 != '\0';sM2++)
357                     //  putchar(*sM2);
358                     //putchar('\n');
359
360                     //while(*shmM2!='*')
361                     //  sleep(1); //El cliente es el que escribe el a
362                     //exit(0);
363                 }
364                 nsops = 1;
365                 sops[0].sem_num = 0;
366                 sops[0].sem_op = -1;
367                 sops[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
368                 if ((j = semop(semid, sops, nsops)) == -1) {
369                     perror("semop: error en semop()\n");
370                 }
371                 else
```

```
                    printf("Proceso padre regresa el control del semaforo.\n");
                sleep(5);
            }
            ++i;
        }
        printf("PROCESO PADRE\n");
        printf("Matriz resultante de la Multiplicación:\n");
        int m_multiplicacion[FILAS][COLUMNAS];
        leermultiplicacion(m_multiplicacion);
        imprimirmatriz(m_multiplicacion);
        printf("\n");
        printf("Matriz resultante de la Suma:\n");
        int m_suma[FILAS][COLUMNAS];
        leersuma(m_suma);
        imprimirmatriz(m_suma);

        //Sacar las inversas de estas matrices resultado
        float m_inversa_multi[FILAS][COLUMNAS];
        copiarmatriz(m_multiplicacion,m_inversa_multi);
        inversa(m_inversa_multi);

        float m_inversa_suma[FILAS][COLUMNAS];
        copiarmatriz(m_suma,m_inversa_suma);
        inversa(m_inversa_suma);

        printf("\n");
        printf("Inversa de la multiplicacion:\n");
        imprimirmatrizres_float(m_inversa_multi);
        printf("\n");
        printf("Inversa de la suma:\n");
        imprimirmatrizres_float(m_inversa_suma);

        guardarmultiinversa(m_inversa_multi);
        guardarsumainversa(m_inversa_suma);

        printf("Archivos creados.\nFin del programa.\n");

    }
    return 0;
}
```

```
void llenarmatriz(int matriz[][COLUMNAS]){
    for (int i = 0; i < FILAS; i++){
        for(int j = 0; j < COLUMNAS; j++){
            matriz[i][j] = rand() % 10;
        }
    }
}
void imprimirmatriz(int matriz[][COLUMNAS]){
    for (int i = 0; i < FILAS; i++){
        for(int j = 0; j < COLUMNAS; j++){
            if(j == 0){
                //printf("|   ");
            }
            printf("%d ",matriz[i][j]);
            if(j == 9){
                printf("  \n");
            }
        }
    }
}
void matrizatexto(int matriz[][COLUMNAS], char* cad){
    int numero;
    char charnumero[12];
    char c;
    int i, j, k;
    for(i = 0; i < FILAS; i++){
        *cad++ = '<';
        for(j = 0; j < COLUMNAS; j++){
            numero = matriz[i][j];
            sprintf(charnumero,"%d", numero);
            k = 0;
            for(k = 0; k < strLen(charnumero);k++){
                c = charnumero[k];
                *cad++ = c;
                //printf("%c",charnumero[k]);
            }
            *cad++ = '|';
```

```
        }
        *cad++ = '>';
    }
    *cad = '\0';
}
void cadenaAentero(char* cad,int matriz[][COLUMNAS]){
    char numero[12] = "";
    int number;
    int i,j,k;
    int fila = 0;
    int columna = 0;
    for(i = 0; i < strlen(cad); i++){
        concatenar(numero,cad[i]);
        if(cad[i] == '<'){
            columna = 0;
            memset(numero,'\0',strlen(numero));
        }
        if(cad[i] == '|'){
            number = atoi(numero);
            matriz[fila][columna] = number ;
            memset(numero,'\0',strlen(numero));
            columna++;
        }
        if(cad[i] == '>'){
            fila++;
        }
    }
}
void concatenar(char* cad, char caracter){
    char cadTemp[2];
    cadTemp[0] = caracter;
    cadTemp[1] = '\0';
    strcat(cad,cadTemp);
}
//imprimir matrices resultado
void imprimirmatrizres(int matriz[][COLUMNAS]){
    for (int i = 0; i < FILAS; i++){
        for(int j = 0; j < COLUMNAS; j++){
            if(j == 0){
                //printf("|   ");
            }
            printf("%d  ",matriz[i][j]);
            if(j == 9){
                printf("  \n");
            }
        }
    }
}
void imprimirmatriz_float(float matriz[][COLUMNAS]){
    for (int i = 0; i < FILAS; i++){
        for(int j = 0; j < COLUMNAS; j++){
            printf("%0.1f ",matriz[i][j]);
            if(j == 9){
                printf("  \n");
            }
        }
    }
}
//imprimir matrices resultado
void imprimirmatrizres_float(float matriz[][COLUMNAS]){
    for (int i = 0; i < FILAS; i++){
        for(int j = 0; j < COLUMNAS; j++){
            printf("%0.4f   ",matriz[i][j]);
            if(j == 9){
                printf("  \n");
            }
        }
    }
}
void multiplicacion(int m1[][COLUMNAS],int m2[][COLUMNAS],int mr[][COLUMNAS]){
    for(int a = 0;a < FILAS;a++){
        for(int i = 0; i < FILAS; i++){
            int suma = 0;
            for(int j = 0;j < FILAS; j++){
                suma +=(m1[i][j] * m2[j][a]);
            }
            mr[i][a] = suma;
```

```
        }
    }
}
void suma(int m1[][COLUMNAS],int m2[][COLUMNAS],int mr[][COLUMNAS]){
    for(int i = 0; i < FILAS; i++){
        for(int j = 0;j < FILAS; j++){
            mr[i][j] = m1[i][j] + m2[i][j];
        }
    }
}

void copiarmatriz(int m1[][COLUMNAS],float mr[][COLUMNAS]){
    for (int i = 0; i < FILAS; i++){
        for(int j = 0; j < COLUMNAS; j++){
            int temp = m1[i][j];
            mr[i][j] = (float)temp;
        }
    }
}
void inversa(float matrix[][COLUMNAS]){
    float **A,**I,temp;
    int i,j,k,matsize;

    matsize = FILAS;

    A=(float **)malloc(matsize*sizeof(float *));            //allocate memory dynamically for matrix A(matsize X

    for(i=0;i<matsize;i++)
        A[i]=(float *)malloc(matsize*sizeof(float));

    I=(float **)malloc(matsize*sizeof(float *));            //memory allocation for indentity matrix I(matsize X

    for(i=0;i<matsize;i++)
        I[i]=(float *)malloc(matsize*sizeof(float));

    // se cargan los elementos de la matriz a A
    for(i=0;i<matsize;i++)
        for(j=0;j<matsize;j++)
            A[i][j] = matrix[i][j];
```

```


    for(i=0;i<matsize;i++)                                  //automatically initialize the unit matrix, e.g.
        for(j=0;j<matsize;j++)                              //  -       -
            if(i==j)                                        // | 1  0  0 |
                I[i][j]=1;                                  // | 0  1  0 |
            else                                            // | 0  0  1 |
                I[i][j]=0;                                  //  -       -
/*---------------LoGiC starts here-----------------*/      //procedure // to make the matrix A to unit matrix

    for(k=0;k<matsize;k++)                                  //by some row operations,and the same row operations
    {                                                       //Unit mat. I gives the inverse of matrix A
        temp=A[k][k];                                       //'temp'
                                                            // stores the A[k][k] value so that A[k][k]  will not
        for(j=0;j<matsize;j++)                              //during the operation //A[i] //[j]/=A[k][k]  when i=
        {
            A[k][j]/=temp;                                  //it performs // the following row operations to make
            I[k][j]/=temp;                                  //R0=R0/A[0][0],similarly for I also R0=R0/A[0][0]
        }                                                   //R1=R1-R0*A[1][0] similarly for I
        for(i=0;i<matsize;i++)                              //R2=R2-R0*A[2][0]      ,,
        {
            temp=A[i][k];                       //R1=R1/A[1][1]
            for(j=0;j<matsize;j++)             //R0=R0-R1*A[0][1]
            {                                   //R2=R2-R1*A[2][1]
                if(i==k)
                    break;                      //R2=R2/A[2][2]
                A[i][j]-=A[k][j]*temp;          //R0=R0-R2*A[0][2]
                I[i][j]-=I[k][j]*temp;          //R1=R1-R2*A[1][2]
            }
        }
    }
/*---------------LoGiC ends here-------------------*/
    //printf("The inverse of the matrix is: ");              //Print the //matrix I that now contains the invers
    for(i=0;i<matsize;i++)
    {
        for(j=0;j<matsize;j++)
            matrix[i][j] = I[i][j];
```

```
603     }
604
605 }
606 void guardarmulti(int matrix[][COLUMNAS]){
607     FILE *m1 = NULL;
608     m1 = fopen("Multiplicación.txt","w+");
609     if(m1 == NULL){
610         printf("No fue posible abrir el archivo\n");
611     }
612     for (int i = 0; i < FILAS; i++){
613         fprintf(m1,"<");
614         for(int j = 0; j < COLUMNAS; j++){
615             fprintf(m1,"%d|",matrix[i][j]);
616         }
617         fprintf(m1,">\n");
618     }
619     fclose(m1);//Se cierra el archivo
620 }
621 void guardarsuma(int matrix[][COLUMNAS]){
622     FILE *m = NULL;
623     m = fopen("Suma.txt","w+");
624     if(m == NULL){
625         printf("No fue posible abrir el archivo\n");
626     }
627     for (int i = 0; i < FILAS; i++){
628         fprintf(m,"<");
629         for(int j = 0; j < COLUMNAS; j++){
630             fprintf(m,"%d|",matrix[i][j]);
631         }
632         fprintf(m,">\n");
633     }
634     fclose(m);//Se cierra el archivo
635 }
636 void guardarmultiinversa(float matrix[][COLUMNAS]){
637     FILE *m1 = NULL;
638     m1 = fopen("MatrizInversaMultiplicación.txt","w+");
639     if(m1 == NULL){
640         printf("No fue posible abrir el archivo\n");
641     }
642     fprintf(m1,"Matriz inversa de la Multiplicación:\n");
643     for (int i = 0; i < FILAS; i++){
644         for(int j = 0; j < COLUMNAS; j++){
645             fprintf(m1,"%0.4f    ",matrix[i][j]);
646             if(j == 9){
647                 fprintf(m1,"    \n");
648             }
649         }
650     }
651     fclose(m1);//Se cierra el archivo
652 }
653 void guardarsumainversa(float matrix[][COLUMNAS]){
654     FILE *m = NULL;
655     m = fopen("MatrizInversaSuma.txt","w+");
656     if(m == NULL){
657         printf("No fue posible abrir el archivo\n");
658     }
659     fprintf(m,"Matriz inversa de la Suma:\n");
660     for (int i = 0; i < FILAS; i++){
661         for(int j = 0; j < COLUMNAS; j++){
662             fprintf(m,"%0.4f    ",matrix[i][j]);
663             if(j == 9){
664                 fprintf(m," \n");
665             }
666         }
667     }
668     fclose(m);//Se cierra el archivo
669 }
670 void leermultiplicacion(int matrix[][COLUMNAS]){
671     const char* archivo = "Multiplicación.txt";
672
673     FILE* input_file = fopen(archivo, "r");
674     if(!input_file)
675         exit(EXIT_FAILURE);
676
677     char *linea = NULL;
678     size_t len = 0;
```

```c
    char buffer[500];
    while(getline(&linea, &len, input_file) != -1){
        //printf("%s", linea);
        strcat(strcpy(buffer,buffer), linea);
    }
    //printf("\n\n%s\n",buffer);
    cadenaAmatriz(buffer,matrix);

    fclose(input_file);
    free(linea);
}
void leersuma(int matrix[][COLUMNAS]){
    const char* archivo = "Suma.txt";

    FILE* input_file = fopen(archivo, "r");
    if(!input_file)
        exit(EXIT_FAILURE);

    char *linea = NULL;
    size_t len = 0;

    char buffer[500];
    while(getline(&linea, &len, input_file) != -1){
        //printf("%s", linea);
        strcat(strcpy(buffer,buffer), linea);
    }
    //printf("\n\n%s\n",buffer);
    cadenaAmatriz(buffer,matrix);

    fclose(input_file);
    free(linea);
}
void cadenaAmatriz(char* cad,int matriz[][COLUMNAS]){
    char numero[12] = "";
    int number;
    int i,j,k;
    int fila = 0;
    int columna = 0;
    for(i = 0; i < strlen(cad); i++){
        concatenar(numero,cad[i]);
        if(cad[i] == '<'){
            columna = 0;
            memset(numero,'\0',strlen(numero));
        }
        if(cad[i] == '|'){
            number = atoi(numero);
            matriz[fila][columna] = number ;
            memset(numero,'\0',strlen(numero));
            columna++;
        }
        if(cad[i] == '>'){
            fila++;
        }
    }
}
```

COMPILACIÓN Y EJECUCIÓN

```
cesar@cesar-HP-Notebook: ~/Documentos/SO/Practica6
cesar@cesar-HP-Notebook:~/Documentos/SO/Practica6$ gcc punto4.c -o punto4
cesar@cesar-HP-Notebook:~/Documentos/SO/Practica6$ ./punto4
Iniciando semaforo...
Semaforo iniciado...
Proceso padre toma el control del semaforo.
Soy el padre(19537, hijo de 4513)
El padre esta en la sección critica
Matriz 1 generada:
5 5 4 2 2 0 2 2 3 7
6 9 1 2 3 9 9 4 2 0
7 6 6 9 8 6 3 5 8 3
6 4 0 2 6 2 2 0 6 7
9 2 7 0 4 0 1 6 4 4
8 4 2 6 3 2 3 7 7 3
2 4 7 4 6 5 8 9 6 5
6 5 7 3 8 4 6 1 2 0
5 0 6 0 6 2 2 1 1 0
5 3 4 4 9 2 2 7 1 8
Matriz 2 generada:
```

```
Matriz 2 generada:
2 0 5 2 5 5 8 1 7 0
4 4 2 2 4 0 4 9 4 5
9 1 0 5 5 9 7 9 9 1
9 1 3 5 5 8 2 3 2 9
5 6 4 9 0 0 0 5 9 4
0 0 7 3 5 4 4 5 4 3
8 3 7 1 0 2 1 3 8 3
4 5 1 0 5 2 1 7 9 2
1 1 3 8 4 0 4 9 5 8
4 3 4 1 6 4 6 8 9 6
Proceso padre regresa el control del semaforo.


Proceso hijo toma el control del semaforo.
```

```
Soy el hijo(19538, hijo de 19537)
El hijo esta en la sección critica
Matriz 1 recibida:
5 5 4 2 2 0 2 2 3 7
6 9 1 2 3 9 9 4 2 0
7 6 6 9 8 6 3 5 8 3
6 4 0 2 6 2 2 0 6 7
9 2 7 0 4 0 1 6 4 4
8 4 2 6 3 2 3 7 7 3
2 4 7 4 6 5 8 9 6 5
6 5 7 3 8 4 6 1 2 0
5 0 6 0 6 2 2 1 1 0
5 3 4 4 9 2 2 7 1 8
Matriz 2 recibida:
2 0 5 2 5 5 8 1 7 0
4 4 2 2 4 0 4 9 4 5
9 1 0 5 5 9 7 9 9 1
9 1 3 5 5 8 2 3 2 9
5 6 4 9 0 0 0 5 9 4
0 0 7 3 5 4 4 5 4 3
8 3 7 1 0 2 1 3 8 3
4 5 1 0 5 2 1 7 9 2
1 1 3 8 4 0 4 9 5 8
4 3 4 1 6 4 6 8 9 6
Multiplicación:
149 78 102 101 139 113 150 205 225 131
180 106 202 124 154 117 152 235 272 154
277 138 210 261 239 213 222 352 377 268
126 87 142 147 132 86 144 204 233 164
161 88 106 130 158 138 176 223 288 104
190 102 153 159 187 146 170 250 286 191
275 153 192 197 205 179 180 349 395 222
216 103 158 178 133 147 154 226 286 139
115 54 81 110 74 93 97 121 181 52
216 140 149 161 174 151 157 263 336 173
Proceso hijo regresa el control del semaforo.
```

```
Soy el nieto(19539, hijo de 19538)
Matriz 1 recibida:
5 5 4 2 2 0 2 2 3 7
6 9 1 2 3 9 9 4 2 0
7 6 6 9 8 6 3 5 8 3
6 4 0 2 6 2 2 0 6 7
9 2 7 0 4 0 1 6 4 4
8 4 2 6 3 2 3 7 7 3
2 4 7 4 6 5 8 9 6 5
6 5 7 3 8 4 6 1 2 0
5 0 6 0 6 2 2 1 1 0
5 3 4 4 9 2 2 7 1 8
Matriz 2 recibida:
2 0 5 2 5 5 8 1 7 0
4 4 2 2 4 0 4 9 4 5
9 1 0 5 5 9 7 9 9 1
9 1 3 5 5 8 2 3 2 9
5 6 4 9 0 0 0 5 9 4
0 0 7 3 5 4 4 5 4 3
8 3 7 1 0 2 1 3 8 3
4 5 1 0 5 2 1 7 9 2
1 1 3 8 4 0 4 9 5 8
4 3 4 1 6 4 6 8 9 6
Suma:
7 5 9 4 7 5 10 3 10 7
10 13 3 4 7 9 13 13 6 5
16 7 6 14 13 15 10 14 17 4
15 5 3 7 11 10 4 3 8 16
14 8 11 9 4 0 1 11 13 8
8 4 9 9 8 6 7 12 11 6
10 7 14 5 6 7 9 12 14 8
10 10 8 3 13 6 7 8 11 2
6 1 9 8 10 2 6 10 6 8
9 6 8 5 15 6 8 15 10 14


PROCESO PADRE
```

```
PROCESO PADRE
Matriz resultante de la Multiplicación:
149 78 102 101 139 113 150 205 225 131
180 106 202 124 154 117 152 235 272 154
277 138 210 261 239 213 222 352 377 268
126 87 142 147 132 86 144 204 233 164
161 88 106 130 158 138 176 223 288 104
190 102 153 159 187 146 170 250 286 191
275 153 192 197 205 179 180 349 395 222
216 103 158 178 133 147 154 226 286 139
115 54 81 110 74 93 97 121 181 52
216 140 149 161 174 151 157 263 336 173

Matriz resultante de la Suma:
7 5 9 4 7 5 10 3 10 7
10 13 3 4 7 9 13 13 6 5
16 7 6 14 13 15 10 14 17 4
15 5 3 7 11 10 4 3 8 16
14 8 11 9 4 0 1 11 13 8
8 4 9 9 8 6 7 12 11 6
10 7 14 5 6 7 9 12 14 8
10 10 8 3 13 6 7 8 11 2
6 1 9 8 10 2 6 10 6 8
9 6 8 5 15 6 8 15 10 14
```

```
Inversa de la multiplicacion:
-0.0403 -0.0145 -0.0215 -0.0032 0.0419  0.0285  0.0015  0.0809  -0.0920 -0.0161
-0.0031 0.0134  0.0333  -0.0013 0.0224  -0.0652 -0.0340 0.0109  -0.0491 0.0482
0.0025  0.0152  0.0039  -0.0027 -0.0106 -0.0028 -0.0016 -0.0145 0.0207  -0.0019
-0.0423 -0.0054 0.0106  0.0074  0.0387  -0.0100 -0.0051 0.0328  -0.0518 -0.0030
-0.0875 -0.0027 -0.0018 -0.0065 0.0835  0.0263  -0.0082 0.0673  -0.1223 -0.0084
0.0837  0.0189  0.0280  -0.0188 -0.0853 -0.0274 -0.0049 -0.1078 0.1494  0.0238
0.0206  -0.0022 -0.0013 0.0075  0.0035  -0.0134 -0.0169 0.0200  -0.0172 0.0048
0.0052  0.0089  0.0226  0.0005  0.0123  -0.0432 0.0114  -0.0163 -0.0048 -0.0070
0.0155  -0.0095 -0.0286 0.0058  -0.0351 0.0411  0.0185  -0.0290 0.0654  -0.0089
0.0483  -0.0077 -0.0135 0.0047  -0.0606 0.0246  0.0064  -0.0386 0.0735  0.0026

Inversa de la suma:
-0.0058 0.0101  0.2441  -0.0704 0.0864  -0.7492 0.1773  -0.0874 0.2790  0.0335
-0.0282 0.0534  -0.2449 0.1014  -0.0253 0.6143  -0.1632 0.1356  -0.2226 -0.0986
-0.0970 -0.0047 -0.1127 0.0722  -0.0629 0.1605  0.0985  0.0923  0.0669  -0.1406
0.0073  0.0243  -0.1437 0.0746  -0.0155 0.5052  -0.1785 0.0436  -0.0856 -0.1195
-0.0055 -0.0267 -0.0097 0.0053  -0.0240 0.0167  -0.0533 0.0777  0.0295  0.0180
-0.1326 0.0008  -0.0995 0.1152  -0.1196 0.3158  0.0685  0.0701  -0.0851 -0.1047
0.1429  0.0380  0.1345  -0.0980 0.0325  -0.3652 0.0199  -0.1078 0.1253  0.0588
-0.0661 0.0143  0.0660  -0.0545 0.0165  -0.1560 0.0681  -0.0496 0.0474  0.0699
0.1304  -0.0647 0.0686  -0.0769 0.0558  -0.0101 -0.0637 -0.0515 -0.1623 0.1352
0.0274  0.0022  -0.0797 0.0481  -0.0024 0.1760  -0.0477 -0.0320 -0.0932 0.0358
Archivos creados.
Fin del programa.
cesar@cesar-MP-Notebook:~/Documentos/SO/Practica6$
```

ARCHIVOS CREADOS

MatrizInversaMultiplicación.txt

```
Matriz inversa de la Multiplicación:
-0.0403 -0.0145 -0.0215 -0.0032 0.0419  0.0285  0.0015  0.0809  -0.0920 -0.0161
-0.0031 0.0134  0.0333  -0.0013 0.0224  -0.0652 -0.0340 0.0109  -0.0491 0.0482
0.0025  0.0152  0.0039  -0.0027 -0.0106 -0.0028 -0.0016 -0.0145 0.0207  -0.0019
-0.0423 -0.0054 0.0106  0.0074  0.0387  -0.0100 -0.0051 0.0328  -0.0518 -0.0030
-0.0875 -0.0027 -0.0018 -0.0065 0.0835  0.0263  -0.0082 0.0673  -0.1223 -0.0084
0.0837  0.0189  0.0280  -0.0188 -0.0853 -0.0274 -0.0049 -0.1078 0.1494  0.0238
0.0206  -0.0022 -0.0013 0.0075  0.0035  -0.0134 -0.0169 0.0200  -0.0172 0.0048
0.0052  0.0089  0.0226  0.0005  0.0123  -0.0432 0.0114  -0.0163 -0.0048 -0.0070
0.0155  -0.0095 -0.0286 0.0058  -0.0351 0.0411  0.0185  -0.0290 0.0654  -0.0089
0.0483  -0.0077 -0.0135 0.0047  -0.0606 0.0246  0.0064  -0.0386 0.0735  0.0026
```

MatrizInversaSuma.txt

```
Matriz inversa de la Suma:
-0.0058 0.0101  0.2441  -0.0704 0.0864  -0.7492 0.1773  -0.0874 0.2790  0.0335
-0.0282 0.0534  -0.2449 0.1014  -0.0253 0.6143  -0.1632 0.1356  -0.2226 -0.0986
-0.0970 -0.0047 -0.1127 0.0722  -0.0629 0.1605  0.0985  0.0923  0.0669  -0.1406
0.0073  0.0243  -0.1437 0.0746  -0.0155 0.5052  -0.1785 0.0436  -0.0856 -0.1195
-0.0055 -0.0267 -0.0097 0.0053  -0.0240 0.0167  -0.0533 0.0777  0.0295  0.0180
-0.1326 0.0008  -0.0995 0.1152  -0.1196 0.3158  0.0685  0.0701  -0.0851 -0.1047
0.1429  0.0380  0.1345  -0.0980 0.0325  -0.3652 0.0199  -0.1078 0.1253  0.0588
-0.0661 0.0143  0.0660  -0.0545 0.0165  -0.1560 0.0681  -0.0496 0.0474  0.0699
0.1304  -0.0647 0.0686  -0.0769 0.0558  -0.0101 -0.0637 -0.0515 -0.1623 0.1352
0.0274  0.0022  -0.0797 0.0481  -0.0024 0.1760  -0.0477 -0.0320 -0.0932 0.0358
```

EXPLICACIÓN DEL CÓDIGO

El objetivo de esta práctica era usar los semáforos para administrar los tiempos de ejecución de los procesos, al intentar acceder a una memoria compartida y que no ocurriera ningún error. Teniendo como base el código presentado en el formato de la práctica, las lecturas de apoyo y lo que se puede encontrar en internet, el código creado funciona de la siguiente forma:

Se empieza creando un semáforo, con todo lo necesario para su manipulación(un variable id, una variable llave, una variable para las banderas, una variable para la cantidad de semáforos) y se inicia el semáforo.

En este caso solo creamos un solo semáforo para toda la práctica pues según nosotros solo se necesitaba en el primer proceso padre e hijo, pues en el de nieto se podía hacer sin administrar el tiempo de la memoria compartida, pues al momento de que empiece el proceso nieto , el proceso hijo y el padre ya hicieron lo que tenían que hacer con la memoria compartida, por lo que quedaba libre para el proceso nieto.

Después de empezar el semáforo, se llamaba al fork() para crear el proceso padre con un proceso hijo, y haciendo uso del semáforo hicimos que el padre siempre entrará primero a las memorias compartidas, pues el padre creaba dos memorias compartidas, una para la matriz 1 y otra para la matriz 2, que eran con las que se trabajan en todo el resto del programa. Una vez creadas las matrices de 10 x 10, se quedaban guardadas en sus respectivas memorias. Por lo que después de que el proceso padre liberará el espacio, el proceso hijo puede acceder a las memorias compartidas en las que se encuentran las matrices antes creadas por el padre, entonces el hijo las multiplicaba y las regresaba.

Después el proceso hijo dejaba de usar la memoria y entonces llamaba otro fork() para crear al proceso nieto, el cual ya no necesitaba a ningún semáforo pues tenía las memorias para el solo. Cuando accedía a las memorias con las matrices el proceso nieto las sumaba y regresaba el resultado.

Al final regresamos al proceso padre y este sacaba la inversa de cada matriz resultado, la de la multiplicación y la de la suma.

El proceso de pasar las matrices por las memorias compartidas fue igual que en el punto 7 de la práctica 5. Nosotros lo hicimos con conversión de las matrices a una cadena usando separadores entre los valores de la matriz para después poder reconstruir la matriz cuando se requiera. La cadena quedaba como: <numero|numero|numero|...><numero|numero|numero|...>... pues después cuando se requería se obtenía otra vez la cadena de la memoria compartida para reconstruir la matriz para poder trabajar con ella.

FUNCIONES USADAS:

//memoria compartida para matrices

void llenarmatriz(int matriz[][COLUMNAS]);

Para llenar una matriz con números enteros aleatorios.

void imprimirmatriz(int matriz[][COLUMNAS]);

Imprime una matriz de enteros, en pantalla/consola.

void matrizatexto(int matriz[][COLUMNAS], char cad);*

Convierte una matriz a una cadena con los diferenciadores (<|>)

void cadenaAentero(char cad,int matriz[][COLUMNAS]);*

Convierte una cadena a matriz.

void concatenar(char cad, char caracter);*

Concatena una cadena con un carácter.

//operaciones con matrices

void imprimirmatrizres(int m[][COLUMNAS]);

Imprime una matriz resultado.

void imprimirmatriz_float(float m[][COLUMNAS]);

Imprime una matriz de flotantes.

void imprimirmatrizres_float(float m[][COLUMNAS]);

Imprime una matriz resultado de flotantes

void multiplicacion(int m1[][COLUMNAS],int m2[][COLUMNAS],int mr[][COLUMNAS]);

Hace la multiplicación de dos matrices y guarda el resultado en una tercera matriz.

void suma(int m1[][COLUMNAS],int m2[][COLUMNAS],int mr[][COLUMNAS]);

Hace la suma de dos matrices y guarda el resultado en una tercera matriz

void copiarmatriz(int m1[][COLUMNAS],float mr[][COLUMNAS]);

Copia una matriz entera a otra matriz pero de flotantes

void inversa(float matrix[][COLUMNAS]);

Obtiene la matriz inversa de la matriz solicitada.

void guardarmulti(int matrix[][COLUMNAS]);

Guarda la multiplicación

void guardarmultiinversa(float matrix[][COLUMNAS]);

Guarda la inversa de la multiplicación

void guardarsuma(int matrix[][COLUMNAS]);

Guarda la suma

void guardarsumainversa(float matrix[][COLUMNAS]);

Guarda la inversa de la suma

void leermultiplicacion(int matrix[][COLUMNAS]);

Lee la matriz resultado de la multiplicación

void cadenaAmatriz(char cad,int matriz[][COLUMNAS]);*

Convierte una cadena en una matriz

void leersuma(int matrix[][COLUMNAS]);

Lee la matriz resultado de la suma

NOTA: Se trabajó con archivos de texto pues se nos complicó hacer una tercera matriz resultado pues los valores no logramos guardarlos una vez que se realizaba una nueva operación. Con los archivos de texto, se guardaban las matrices de resultado por lo que se nos hizo más fácil trabajar con ellos.

VERSIÓN EN WINDOWS:

padre.c

padre.c | hijo.c | nieto.c

```c
#include <windows.h> /*Programa padre*/
#include <stdio.h>

#include <time.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

const int FILAS = 10;
const int COLUMNAS = 10;

#define TAM_MEM 400
int i, j;

void guardarmatriz(int matrix[][COLUMNAS], char* nombre);
void llenarmatriz(int matriz[][COLUMNAS]);
void imprimirmatriz(int matriz[][COLUMNAS]);
void leermatriz(int matrix[][COLUMNAS], char* nombre);
void cadenaAmatriz(char* cad,int matriz[][COLUMNAS]);
size_t getline(char **lineptr, size_t *n, FILE *stream);
void copiarmatriz(int m1[][COLUMNAS],float mr[][COLUMNAS]);
void inversa(float matrix[][COLUMNAS]);
void imprimirmatrizres_float(float matriz[][COLUMNAS]);
void guardarmulti(float matrix[][COLUMNAS]);
void guardarsuma(float matrix[][COLUMNAS]);
void concatenar(char* cad, char caracter);

int main(int argc, char *argv[])
{
    remove("matriz1.txt");
    remove("matriz2.txt");
    remove("Multiplicacion.txt");
    remove("Suma.txt");
    remove("MatrizInversaSuma.txt");
    remove("MatrizInversaMultiplicacion.txt");
    srand(time(NULL));
    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));
    //Se verifica que se ejecute bien el programa
    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
```

```
58      if(!CreateProcess(NULL, argv[1], NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
59      {
60          printf("Falla al invocar CreateProcess: %d\n", GetLastError() );
61          return -1;
62      }
63      while(i<2)
64      {
65          // Prueba del semáforo
66          WaitForSingleObject(hSemaforo, INFINITE);
67          //Sección crítica
68          printf("Soy el padre entrando al semaforo\n",i);
69          //Sleep(5000);
70          {//Codigo de creacion de memoria compartida
71              printf("\n  PROCESO PADRE\n");
72              int matriz1[FILAS][COLUMNAS];
73              int matriz2[FILAS][COLUMNAS];
74              llenarmatriz(matriz1);
75              llenarmatriz(matriz2);
76              guardarmatriz(matriz1,"matriz1.txt");
77              guardarmatriz(matriz2,"matriz2.txt");
78              printf("    Matriz 1:\n");
79              imprimirmatriz(matriz1);
80              printf("    Matriz 2:\n");
81              imprimirmatriz(matriz2);
82              printf("    Matrices enviadas para multiplicar\n");
83          }
84          //Liberación el semáforo
85          if (!ReleaseSemaphore(hSemaforo, 1, NULL) )
```

```
86          {
87              printf("Falla al invocar ReleaseSemaphore: %d\n", GetLastError());
88          }
89          printf("Soy el padre liberando al semaforo\n",i);
90          Sleep(1000);
91          i++;
92      }
93
94      // Terminación controlada del proceso e hilo asociado de ejecución
95      CloseHandle(pi.hProcess);
96      CloseHandle(pi.hThread);
97      {
98          printf("\n  PROCESO PADRE\n");
99          int m_multiplicacion[FILAS][COLUMNAS];
100         leermatriz(m_multiplicacion,"Multiplicacion.txt");
101         printf("Matriz resultado de la Multiplicacion\n");
102         imprimirmatriz(m_multiplicacion);
103
104         int m_suma[FILAS][COLUMNAS];
105         leermatriz(m_suma,"Suma.txt");
106         printf("Matriz resultado de la Suma\n");
107         imprimirmatriz(m_suma);
108
109         //Sacar las inversas de estas matrices resultado
110         float m_inversa_multi[FILAS][COLUMNAS];
111         copiarmatriz(m_multiplicacion,m_inversa_multi);
112         inversa(m_inversa_multi);
113
114         float m_inversa_suma[FILAS][COLUMNAS];
115         copiarmatriz(m_suma,m_inversa_suma);
116         inversa(m_inversa_suma);
117
```

```
        printf("\n");
        printf("Inversa de la multiplicacion:\n");
        imprimirmatrizres_float(m_inversa_multi);
        printf("\n");
        printf("Inversa de la suma:\n");
        imprimirmatrizres_float(m_inversa_suma);

        guardarmulti(m_inversa_multi);
        guardarsuma(m_inversa_suma);

        printf("Archivos creados.\nFin del programa.\n");
    }
}
void llenarmatriz(int matriz[][COLUMNAS]){
    int i, j;
    for (i = 0; i < FILAS; i++){
        for(j = 0; j < COLUMNAS; j++){
            matriz[i][j] = rand() % 10;
        }
    }
}
void guardarmatriz(int matrix[][COLUMNAS], char* nombre){
    FILE *archivo = NULL;
    archivo = fopen(nombre,"w+");
    if(archivo == NULL){
        printf("No fue posible abrir el archivo\n");
    }
    for (i = 0; i < FILAS; i++){
        fprintf(archivo,"<");
        for(j = 0; j < COLUMNAS; j++){
            fprintf(archivo,"%d|",matrix[i][j]);
        }
        fprintf(archivo,">\n");
    }
    fclose(archivo);//Se cierra el archivo
}
void imprimirmatriz(int matriz[][COLUMNAS]){
    for (i = 0; i < FILAS; i++){
        for(j = 0; j < COLUMNAS; j++){
            if(j == 0){
                printf("    ");
            }
            printf("%d ",matriz[i][j]);
            if(j == 9){
                printf("\n");
            }
        }
    }
}
void leermatriz(int matrix[][COLUMNAS], char* nombre){
    FILE* input_file = fopen(nombre, "r");
    if(!input_file)
        exit(EXIT_FAILURE);

    char *linea = NULL;
    size_t len = 0;

    char buffer[500];
```

```
176     while(getline(&linea, &len, input_file) != -1){
177         //printf("%s", linea);
178         strcat(strcpy(buffer,buffer), linea);
179     }
180     //printf("\n\n%s\n",buffer);
181     cadenaAmatriz(buffer,matrix);
182 
183     fclose(input_file);
184     free(linea);
185 }
186 void cadenaAmatriz(char* cad,int matriz[][COLUMNAS]){
187     char numero[12] = "";
188     int number;
189     int i,j,k;
190     int fila = 0;
191     int columna = 0;
192     for(i = 0; i < strlen(cad); i++){
193         concatenar(numero,cad[i]);
194         if(cad[i] == '<'){
195             columna = 0;
196             memset(numero,'\0',strlen(numero));
197         }
198         if(cad[i] == '|'){
199             number = atoi(numero);
200             matriz[fila][columna] = number ;
201             memset(numero,'\0',strlen(numero));
202             columna++;
203         }
204         if(cad[i] == '>'){
205             fila++;
206         }
207     }
208 }
209 size_t getline(char **lineptr, size_t *n, FILE *stream){
210     char *bufptr = NULL;
211     char *p = bufptr;
212     size_t size;
213     int c;
```

```
214 
215     if (lineptr == NULL) {
216         return -1;
217     }
218     if (stream == NULL) {
219         return -1;
220     }
221     if (n == NULL) {
222         return -1;
223     }
224     bufptr = *lineptr;
225     size = *n;
226 
227     c = fgetc(stream);
228     if (c == EOF) {
229         return -1;
230     }
231     if (bufptr == NULL) {
232         bufptr = malloc(128);
233         if (bufptr == NULL) {
234             return -1;
235         }
```

```
236         size = 128;
237     }
238     p = bufptr;
239     while(c != EOF) {
240         if ((p - bufptr) > (size - 1)) {
241             size = size + 128;
242             bufptr = realloc(bufptr, size);
243             if (bufptr == NULL) {
244                 return -1;
245             }
246         }
247         *p++ = c;
248         if (c == '\n') {
249             break;
250         }
251         c = fgetc(stream);
252     }
253
254     *p++ = '\0';
255     *lineptr = bufptr;
256     *n = size;
257
258     return p - bufptr - 1;
259 }
260 void copiarmatriz(int m1[][COLUMNAS],float mr[][COLUMNAS]){
261     int i, j;
262     for (i = 0; i < FILAS; i++){
263         for(j = 0; j < COLUMNAS; j++){
264             int temp = m1[i][j];
265             mr[i][j] = (float)temp;
266         }
267     }
268 }
269 void inversa(float matrix[][COLUMNAS]){
270     float **A,**I,temp;
271     int i,j,k,matsize;
272
273     matsize = FILAS;
274
275     A=(float **)malloc(matsize*sizeof(float *));
276
277     for(i=0;i<matsize;i++)
278         A[i]=(float *)malloc(matsize*sizeof(float));
279
280     I=(float **)malloc(matsize*sizeof(float *));
281
282     for(i=0;i<matsize;i++)
283         I[i]=(float *)malloc(matsize*sizeof(float));
284
285     // se cargan los elementos de la matriz a A
286     for(i=0;i<matsize;i++)
287         for(j=0;j<matsize;j++)
288             A[i][j] = matrix[i][j];
289
290
291     for(i=0;i<matsize;i++)
292         for(j=0;j<matsize;j++)
293             if(i==j)
294                 I[i][j]=1;
295             else
```

```
                I[i][j]=0;
/*---------------LoGiC starts here------------------*/

    for(k=0;k<matsize;k++)
    {
        temp=A[k][k];

        for(j=0;j<matsize;j++)
        {
            A[k][j]/=temp;
            I[k][j]/=temp;
        }
        for(i=0;i<matsize;i++)
        {
            temp=A[i][k];
            for(j=0;j<matsize;j++)
            {
                if(i==k)
                    break;
                A[i][j]-=A[k][j]*temp;
                I[i][j]-=I[k][j]*temp;
            }
        }
    }
/*---------------LoGiC ends here--------------------*/
    //printf("The inverse of the matrix is: ");
    for(i=0;i<matsize;i++)
    {
        for(j=0;j<matsize;j++)
            matrix[i][j] = I[i][j];
    }

}
void imprimirmatrizres_float(float matriz[][COLUMNAS]){
    int i, j;
    for (i = 0; i < FILAS; i++){
        printf("    ");
        for(j = 0; j < COLUMNAS; j++){
            printf("%0.4f   ",matriz[i][j]);
            if(j == 9){
                printf("  \n");
            }
        }
    }
}
void guardarmulti(float matrix[][COLUMNAS]){
    FILE *m1 = NULL;
    m1 = fopen("MatrizInversaMultiplicacion.txt","w+");
    if(m1 == NULL){
        printf("No fue posible abrir el archivo\n");
    }
    fprintf(m1,"Matriz inversa de la Multiplicación:\n");
    int i, j;
    for (i = 0; i < FILAS; i++){
        for(j = 0; j < COLUMNAS; j++){
            fprintf(m1,"%0.4f   ",matrix[i][j]);
            if(j == 9){
                fprintf(m1,"    \n");
```

```
354             }
355         }
356     }
357     fclose(m1);//Se cierra el archivo
358 }
359 void guardarsuma(float matrix[][COLUMNAS]){
360     FILE *m = NULL;
361     m = fopen("MatrizInversaSuma.txt","w+");
362     if(m == NULL){
363         printf("No fue posible abrir el archivo\n");
364     }
365     fprintf(m,"Matriz inversa de la Suma:\n");
366     int i, j;
367     for (i = 0; i < FILAS; i++){
368         for(j = 0; j < COLUMNAS; j++){
369             fprintf(m,"%0.4f    ",matrix[i][j]);
370             if(j == 9){
371                 fprintf(m," \n");
372             }
373         }
374     }
375     fclose(m);//Se cierra el archivo
376 }
377 void concatenar(char* cad, char caracter){
378     char cadTemp[2];
379     cadTemp[0] = caracter;
380     cadTemp[1] = '\0';
381     strcat(cad,cadTemp);
```

```
381     strcat(cad,cadTemp);
382 }
```

hijo.c

padre.c | hijo.c | nieto.c

```
1  #include <windows.h> /*Programa hijo*/
2  #include <stdio.h>
3
4  #include <time.h>
5  #include <stdlib.h>
6  #include <unistd.h>
7  #include <string.h>
8
9  const int FILAS = 10;
10 const int COLUMNAS = 10;
11
12 #define TAM_MEM 400
13 int i, j;
14
15 void leermatriz(int matrix[][COLUMNAS], char* nombre);
16 void cadenaAmatriz(char* cad,int matriz[][COLUMNAS]);
17 void imprimirmatriz(int matriz[][COLUMNAS]);
18 void concatenar(char* cad, char caracter);
19 void multiplicacion(int m1[][COLUMNAS],int m2[][COLUMNAS],int mr[][COLUMNAS]);
20 void guardarmatriz(int matrix[][COLUMNAS], char* nombre);
21 size_t getline(char **lineptr, size_t *n, FILE *stream);
22
23
24 int main()
25 {
```

```
26      HANDLE hSemaforo;
27      int i=1;
28      // Apertura del semáforo
29      if((hSemaforo = OpenSemaphore(SEMAPHORE_ALL_ACCESS, FALSE, "Semaforo")) ==NULL)
30      {
31          printf("Falla al invocar OpenSemaphore: %d\n", GetLastError());
32          return -1;
33      }
34      while(i<2)
35      {
36          // Prueba del semáforo
37          WaitForSingleObject(hSemaforo, INFINITE);
38          //Sección crítica
39          printf("Soy el hijo entrando al semaforo\n",i);
40          //Sleep(5000);
41          {//Codigo del acceso a la memoria compartida
42              printf("\n      PROCESO HIJO\n");
43              int matriz1[FILAS][COLUMNAS];
44              leermatriz(matriz1,"matriz1.txt");
45              printf("        Matriz 1:\n");
46              imprimirmatriz(matriz1);
47
48              int matrizdos[FILAS][COLUMNAS];
49              leermatriz(matrizdos,"matriz2.txt");
50              printf("        Matriz 2:\n");
51              imprimirmatriz(matrizdos);
52
53              printf("        Matriz resultado de la multiplicacion\n");
54              int matriz_res_multi[FILAS][COLUMNAS];
55              multiplicacion(matriz1,matrizdos,matriz_res_multi);
56              imprimirmatriz(matriz_res_multi);
57              guardarmatriz(matriz_res_multi,"Multiplicacion.txt");
58              printf("        Matriz resultado enviada al padre\n");
59
60
61          }
62          //Liberación el semáforo
63          if (!ReleaseSemaphore(hSemaforo, 1, NULL) )
64          {
65              printf("Falla al invocar ReleaseSemaphore: %d\n", GetLastError());
66          }
67          printf("Soy el hijo liberando al semaforo\n",i);
68          //Sleep(5000);
69          {//CREACION DEL PROCESO NIETO
70                  PROCESS_INFORMATION piHijoHN;
71                  STARTUPINFO siHijoHN;
72                  //Obtencion de información para la inicialización del proceso hijo
73                  GetStartupInfo(&siHijoHN);
74                  printf("        Proceso NIETO creado.\n");
75                  printf("        Matrices enviadas para sumar.\n");
76                  CreateProcess (NULL, "nieto.exe", NULL, NULL,
77                  TRUE, // Hereda el proceso hijo los manejadores del padre
78                  0, NULL, NULL, &siHijoHN, &piHijoHN);
79                  WaitForSingleObject (piHijoHN.hProcess, INFINITE);
80                  CloseHandle(piHijoHN.hThread);
81                  CloseHandle(piHijoHN.hProcess);
82          }
83          i++;
84      }
85  }
```

```c
void leermatriz(int matrix[][COLUMNAS], char* nombre){
    FILE* input_file = fopen(nombre, "r");
    if(!input_file)
        exit(EXIT_FAILURE);

    char *linea = NULL;
    size_t len = 0;

    char buffer[500];
    while(getline(&linea, &len, input_file) != -1){
        //printf("%s", linea);
        strcat(strcpy(buffer,buffer), linea);
    }
    //printf("\n\n%s\n",buffer);
    cadenaAmatriz(buffer,matrix);

    fclose(input_file);
    free(linea);
}
void cadenaAmatriz(char* cad,int matriz[][COLUMNAS]){
    char numero[12] = "";
    int number;
    int i,j,k;
    int fila = 0;
    int columna = 0;
    for(i = 0; i < strlen(cad); i++){
        concatenar(numero,cad[i]);
        if(cad[i] == '<'){
            columna = 0;
            memset(numero,'\0',strlen(numero));
        }
        if(cad[i] == '|'){
            number = atoi(numero);
            matriz[fila][columna] = number ;
            memset(numero,'\0',strlen(numero));
            columna++;
        }
        if(cad[i] == '>'){
            fila++;
        }
    }
}
```

```c
void imprimirmatriz(int matriz[][COLUMNAS]){
    for (i = 0; i < FILAS; i++){
        for(j = 0; j < COLUMNAS; j++){
            if(j == 0){
                printf("        ");
            }
            printf("%d ",matriz[i][j]);
            if(j == 9){
                printf("\n");
            }
        }
    }
}
void concatenar(char* cad, char caracter){
    char cadTemp[2];
    cadTemp[0] = caracter;
    cadTemp[1] = '\0';
    strcat(cad,cadTemp);
```

```
}
void multiplicacion(int m1[][COLUMNAS],int m2[][COLUMNAS],int mr[][COLUMNAS]){
    int a;
    for(a = 0;a < FILAS;a++){
        for(i = 0; i < FILAS; i++){
            int suma = 0;
            for(j = 0;j < FILAS; j++){
                suma +=(m1[i][j] * m2[j][a]);
            }
            mr[i][a] = suma;
        }
    }
}
void guardarmatriz(int matrix[][COLUMNAS], char* nombre){
    FILE *archivo = NULL;
    archivo = fopen(nombre,"w+");
    if(archivo == NULL){
        printf("No fue posible abrir el archivo\n");
    }
    for (i = 0; i < FILAS; i++){
        fprintf(archivo,"<");
        for(j = 0; j < COLUMNAS; j++){
            fprintf(archivo,"%d|",matrix[i][j]);
        }
        fprintf(archivo,">\n");
    }
    fclose(archivo);//Se cierra el archivo
}
size_t getline(char **lineptr, size_t *n, FILE *stream){
    char *bufptr = NULL;
    char *p = bufptr;
    size_t size;
    int c;

    if (lineptr == NULL) {
        return -1;
    }
    if (stream == NULL) {
        return -1;
    }
    if (n == NULL) {
        return -1;
    }
    bufptr = *lineptr;
    size = *n;

    c = fgetc(stream);
    if (c == EOF) {
        return -1;
    }
    if (bufptr == NULL) {
        bufptr = malloc(128);
        if (bufptr == NULL) {
            return -1;
        }
        size = 128;
    }
    p = bufptr;
    while(c != EOF) {
        if ((p - bufptr) > (size - 1)) {
```

```
206             size = size + 128;
207             bufptr = realloc(bufptr, size);
208             if (bufptr == NULL) {
209                 return -1;
210             }
211         }
212         *p++ = c;
```

```
213         if (c == '\n') {
214             break;
215         }
216         c = fgetc(stream);
217     }
218
219     *p++ = '\0';
220     *lineptr = bufptr;
221     *n = size;
222
223     return p - bufptr - 1;
224 }
```

nieto.c

padre.c | hijo.c | nieto.c

```
1  #include <windows.h> /*Programa hijo*/
2  #include <stdio.h>
3
4  #include <time.h>
5  #include <stdlib.h>
6  #include <unistd.h>
7  #include <string.h>
8
9  const int FILAS = 10;
10 const int COLUMNAS = 10;
11
12 #define TAM_MEM 400
13 int i, j;
14
15 void leermatriz(int matrix[][COLUMNAS], char* nombre);
16 void cadenaAmatriz(char* cad,int matriz[][COLUMNAS]);
17 size_t getline(char **lineptr, size_t *n, FILE *stream);
18 void imprimirmatriz(int matriz[][COLUMNAS]);
19 void concatenar(char* cad, char caracter);
20 void suma(int m1[][COLUMNAS],int m2[][COLUMNAS],int mr[][COLUMNAS]);
21 void guardarmatriz(int matrix[][COLUMNAS], char* nombre);
22
23 int main()
24 {
25     printf("\n          PROCESO NIETO\n");
26     int matriz1[FILAS][COLUMNAS];
27     leermatriz(matriz1,"matriz1.txt");
28     printf("           Matriz 1:\n");
29     imprimirmatriz(matriz1);
30
31     int matrizdos[FILAS][COLUMNAS];
32     leermatriz(matrizdos,"matriz2.txt");
33     printf("           Matriz 2:\n");
34     imprimirmatriz(matrizdos);
35
36     printf("           Matriz resultado de la suma:\n");
37     int mr[FILAS][COLUMNAS];
```

```
    suma(matriz1,matrizdos,mr);
    imprimirmatriz(mr);
    guardarmatriz(mr,"Suma.txt");

    printf("            Matriz resultado enviada al padre\n");
}
void leermatriz(int matrix[][COLUMNAS], char* nombre){
    FILE* input_file = fopen(nombre, "r");
    if(!input_file)
        exit(EXIT_FAILURE);

    char *linea = NULL;
    size_t len = 0;

    char buffer[500];
    while(getline(&linea, &len, input_file) != -1){
        //printf("%s", linea);
        strcat(strcpy(buffer,buffer), linea);
    }
    //printf("\n\n%s\n",buffer);
    cadenaAmatriz(buffer,matrix);

    fclose(input_file);
    free(linea);
}
void cadenaAmatriz(char* cad,int matriz[][COLUMNAS]){
    char numero[12] = "";
    int number;
    int i,j,k;
    int fila = 0;
    int columna = 0;
    for(i = 0; i < strlen(cad); i++){
        concatenar(numero,cad[i]);
        if(cad[i] == '<'){
            columna = 0;
            memset(numero,'\0',strlen(numero));
        }
        if(cad[i] == '|'){
            number = atoi(numero);
            matriz[fila][columna] = number ;
            memset(numero,'\0',strlen(numero));
            columna++;
        }
        if(cad[i] == '>'){
            fila++;
        }
    }
}
size_t getline(char **lineptr, size_t *n, FILE *stream){
    char *bufptr = NULL;
    char *p = bufptr;
    size_t size;
    int c;

    if (lineptr == NULL) {
        return -1;
    }
    if (stream == NULL) {
        return -1;
```

```
    }
    if (n == NULL) {
        return -1;
    }
    bufptr = *lineptr;
    size = *n;

    c = fgetc(stream);
    if (c == EOF) {
        return -1;
    }
    if (bufptr == NULL) {
        bufptr = malloc(128);
        if (bufptr == NULL) {
            return -1;
        }
        size = 128;
    }
    p = bufptr;
    while(c != EOF) {
        if ((p - bufptr) > (size - 1)) {
            size = size + 128;
            bufptr = realloc(bufptr, size);
            if (bufptr == NULL) {
                return -1;
            }
        }
        *p++ = c;
        if (c == '\n') {
            break;
        }
        c = fgetc(stream);
```

```
    }

    *p++ = '\0';
    *lineptr = bufptr;
    *n = size;

    return p - bufptr - 1;
}
void imprimirmatriz(int matriz[][COLUMNAS]){
    for (i = 0; i < FILAS; i++){
        for(j = 0; j < COLUMNAS; j++){
            if(j == 0){
                printf("            ");
            }
            printf("%d ",matriz[i][j]);
            if(j == 9){
                printf("\n");
            }
        }
    }
}
void concatenar(char* cad, char caracter){
    char cadTemp[2];
    cadTemp[0] = caracter;
    cadTemp[1] = '\0';
    strcat(cad,cadTemp);
}
void suma(int m1[][COLUMNAS],int m2[][COLUMNAS],int mr[][COLUMNAS]){
```

```
157     int i, j;
158     for(i = 0; i < FILAS; i++){
159         for(j = 0;j < FILAS; j++){
160             mr[i][j] = m1[i][j] + m2[i][j];
161         }
162     }
163 }
164 void guardarmatriz(int matrix[][COLUMNAS], char* nombre){
165     FILE *archivo = NULL;
166     archivo = fopen(nombre,"w+");
167     if(archivo == NULL){
168         printf("No fue posible abrir el archivo\n");
169     }
170     for (i = 0; i < FILAS; i++){
171         fprintf(archivo,"<");
172         for(j = 0; j < COLUMNAS; j++){
173             fprintf(archivo,"%d|",matrix[i][j]);
174         }
175         fprintf(archivo,">\n");
176     }
177     fclose(archivo);//Se cierra el archivo
178 }
```

COMPILACIÓN Y EJECUCIÓN

```
Símbolo del sistema

C:\Users\cesar\OneDrive\Documentos\Cuarto Semestre\SISTEMAS OPERATIVOS\Tercer Parcial\Practica6\Windows\Pruebas>
gcc padre.c -o padre

C:\Users\cesar\OneDrive\Documentos\Cuarto Semestre\SISTEMAS OPERATIVOS\Tercer Parcial\Practica6\Windows\Pruebas>
gcc hijo.c -o hijo

C:\Users\cesar\OneDrive\Documentos\Cuarto Semestre\SISTEMAS OPERATIVOS\Tercer Parcial\Practica6\Windows\Pruebas>
gcc nieto.c -o nieto

C:\Users\cesar\OneDrive\Documentos\Cuarto Semestre\SISTEMAS OPERATIVOS\Tercer Parcial\Practica6\Windows\Pruebas>
padre hijo
Soy el padre entrando al semaforo

        PROCESO PADRE
        Matriz 1:
        0 5 4 3 3 4 6 7 3 0
        0 0 2 3 5 6 4 5 9 0
        2 2 2 3 1 3 7 5 8 9
        3 6 6 0 1 7 4 5 5 3
        0 1 2 2 2 0 3 1 9 0
        2 5 0 2 0 2 4 5 2 5
        7 2 6 5 4 6 9 8 0 8
        6 7 6 7 8 0 9 8 1 3
        6 7 5 3 6 9 7 1 4 9
        6 4 9 6 3 3 3 1 7 7
        Matriz 2:
        5 5 3 2 6 6 7 3 2 7
        8 5 6 0 9 3 0 5 3 9
        0 2 0 7 2 7 3 6 4 9
        8 6 9 3 7 5 2 8 9 2
        0 6 5 0 4 7 1 5 4 3
        6 6 2 9 7 4 7 8 6 5
        1 2 3 4 5 6 7 7 4 0
        6 1 3 8 5 9 6 7 4 2
        6 0 9 6 5 8 1 0 4 4
        8 3 6 1 7 2 8 6 5 4
        Matrices enviadas para multiplicar
Soy el padre liberando al semaforo
Soy el hijo entrando al semaforo
```

```
Soy el padre liberando al semaforo
Soy el hijo entrando al semaforo

                PROCESO HIJO
                Matriz 1:
                0 5 4 3 3 4 6 7 3 0
                0 0 2 3 5 6 4 5 9 0
                2 2 2 3 1 3 7 5 8 9
                3 6 6 0 1 7 4 5 5 3
                0 1 2 2 2 0 3 1 9 0
                2 5 0 2 0 2 4 5 2 5
                7 2 6 5 4 6 9 8 0 8
                6 7 6 7 8 0 9 8 1 3
                6 7 5 3 6 9 7 1 4 9
                6 4 9 6 3 3 3 1 7 7
                Matriz 2:
                5 5 3 2 6 6 7 3 2 7
                8 5 6 0 9 3 0 5 3 9
                0 2 0 7 2 7 3 6 4 9
                8 6 9 3 7 5 2 8 9 2
                0 6 5 0 4 7 1 5 4 3
                6 6 2 9 7 4 7 8 6 5
                1 2 3 4 5 6 7 7 4 0
                6 1 3 8 5 9 6 7 4 2
                6 0 9 6 5 8 1 0 4 4
                8 3 6 1 7 2 8 6 5 4
                Matriz resultado de la multiplicacion
                154 112 146 171 194 218 136 211 158 142
                148 101 172 187 177 229 126 172 163 115
                225 112 218 179 243 235 207 219 192 152
                193 127 154 200 228 228 176 217 165 209
                87 40 127 94 100 140 48 71 89 75
                164 87 133 101 175 140 132 156 118 111
                248 197 209 233 306 309 298 342 249 221
                229 202 241 184 303 330 218 322 239 229
                273 225 249 206 340 286 263 319 251 273
                235 168 234 189 269 271 195 249 223 253
                Matriz resultado enviada al padre
Soy el hijo liberando al semaforo
```

```
Soy el hijo liberando al semaforo
                Proceso NIETO creado.
                Matrices enviadas para sumar.

                        PROCESO NIETO
                        Matriz 1:
                        0 5 4 3 3 4 6 7 3 0
                        0 0 2 3 5 6 4 5 9 0
                        2 2 2 3 1 3 7 5 8 9
                        3 6 6 0 1 7 4 5 5 3
                        0 1 2 2 2 0 3 1 9 0
                        2 5 0 2 0 2 4 5 2 5
                        7 2 6 5 4 6 9 8 0 8
                        6 7 6 7 8 0 9 8 1 3
                        6 7 5 3 6 9 7 1 4 9
                        6 4 9 6 3 3 3 1 7 7
                        Matriz 2:
                        5 5 3 2 6 6 7 3 2 7
                        8 5 6 0 9 3 0 5 3 9
                        0 2 0 7 2 7 3 6 4 9
                        8 6 9 3 7 5 2 8 9 2
                        0 6 5 0 4 7 1 5 4 3
                        6 6 2 9 7 4 7 8 6 5
                        1 2 3 4 5 6 7 7 4 0
                        6 1 3 8 5 9 6 7 4 2
                        6 0 9 6 5 8 1 0 4 4
                        8 3 6 1 7 2 8 6 5 4
                        Matriz resultado de la suma:
                        5 10 7 5 9 10 13 10 5 7
                        8 5 8 3 14 9 4 10 12 9
                        2 4 2 10 3 10 10 11 12 18
                        11 12 15 3 8 12 6 13 14 5
                        0 7 7 2 6 7 4 6 13 3
                        8 11 2 11 7 6 11 13 8 10
                        8 4 9 9 9 12 16 15 4 8
                        12 8 9 15 13 9 15 15 5 5
                        12 7 14 9 11 17 8 1 8 13
                        14 7 15 7 10 5 11 7 12 11
                        Matriz resultado enviada al padre

        PROCESO PADRE
```

```
        PROCESO PADRE
Matriz resultado de la Multiplicacion
        154 112 146 171 194 218 136 211 158 142
        148 101 172 187 177 229 126 172 163 115
        225 112 218 179 243 235 207 219 192 152
        193 127 154 200 228 228 176 217 165 209
        87 40 127 94 100 140 48 71 89 75
        164 87 133 101 175 140 132 156 118 111
        248 197 209 233 306 309 298 342 249 221
        229 202 241 184 303 330 218 322 239 229
        273 225 249 206 340 286 263 319 251 273
        235 168 234 189 269 271 195 249 223 253
Matriz resultado de la Suma
        5 10 7 5 9 10 13 10 5 7
        8 5 8 3 14 9 4 10 12 9
        2 4 2 10 3 10 10 11 12 18
        11 12 15 3 8 12 6 13 14 5
        0 7 7 2 6 7 4 6 13 3
        8 11 2 11 7 6 11 13 8 10
        8 4 9 9 9 12 16 15 4 8
        12 8 9 15 13 9 15 15 5 5
        12 7 14 9 11 17 8 1 8 13
        14 7 15 7 10 5 11 7 12 11

Inversa de la multiplicacion:
        -0.0364 0.0429  0.0017  0.0141  -0.0599 0.0328  -0.0179 0.0194  -0.0312 0.0232
        -0.1079 0.1451  0.1933  0.1009  -0.3116 -0.1408 -0.1786 0.1386  -0.0170 -0.0018
        -0.1872 0.2884  0.6510  0.2751  -0.8390 -0.4990 -0.5239 0.3763  -0.0415 -0.0150
        -0.0644 0.1177  0.2348  0.1071  -0.3148 -0.1828 -0.1942 0.1298  -0.0109 -0.0095
        0.2005  -0.2975 -0.5681 -0.2512 0.8031  0.4192  0.4642  -0.3482 0.0792  -0.0263
        0.0295  -0.0819 -0.2162 -0.0753 0.2735  0.1705  0.1721  -0.1030 -0.0020 0.0020
        -0.0421 0.0308  0.0801  0.0392  -0.0901 -0.0655 -0.0488 0.0450  -0.0057 -0.0075
        -0.1095 0.2016  0.4795  0.1973  -0.6328 -0.3625 -0.3863 0.2775  -0.0421 -0.0029
        0.2738  -0.3772 -0.7872 -0.3781 1.0503  0.5863  0.6503  -0.4882 0.0665  0.0367
        -0.0192 0.0232  0.0884  0.0421  -0.1066 -0.0768 -0.0685 0.0479  -0.0081 0.0067

Inversa de la suma:
        -0.3188 0.0051  -0.2685 -0.0878 0.1311  0.4078  0.3668  -0.3073 0.1045  0.0209
        0.1901  -0.0235 0.0821  0.0752  -0.0923 -0.0970 -0.2156 0.1090  -0.0297 -0.0146
        0.3141  -0.0263 0.2864  0.1647  -0.2168 -0.4740 -0.3966 0.3428  -0.1094 0.0300
        0.0628  -0.0373 0.1386  0.0444  -0.0548 -0.1698 -0.2029 0.2261  -0.0177 -0.0214
        0.1020  0.0841  0.0249  -0.0364 -0.0277 -0.0930 -0.1050 0.0899  -0.0211 -0.0052
        -0.1654 -0.0066 -0.1325 -0.0403 0.1125  0.1940  0.2283  -0.1695 0.0999  -0.0682
        -0.1800 -0.0445 -0.2186 -0.1540 0.2062  0.2969  0.3409  -0.2659 0.0617  0.0524
        0.1162  0.0302  0.1379  0.1112  -0.1391 -0.1805 -0.1424 0.1449  -0.0911 -0.0274
        -0.2880 -0.0101 -0.2024 -0.1188 0.2493  0.3185  0.3052  -0.2479 0.0706  0.0265
        0.2193  0.0309  0.1999  0.0754  -0.1932 -0.2360 -0.2512 0.1498  -0.0607 0.0182
Archivos creados.
Fin del programa.

C:\Users\cesar\OneDrive\Documentos\Cuarto Semestre\SISTEMAS OPERATIVOS\Tercer Parcial\Practica6\Windows\Pruebas>
```

ARCHIVOS CREADOS

matriz1.txt

```
<0|5|4|3|3|4|6|7|3|0|>
<0|0|2|3|5|6|4|5|9|0|>
<2|2|2|3|1|3|7|5|8|9|>
<3|6|6|0|1|7|4|5|5|3|>
<0|1|2|2|2|0|3|1|9|0|>
<2|5|0|2|0|2|4|5|2|5|>
<7|2|6|5|4|6|9|8|0|8|>
<6|7|6|7|8|0|9|8|1|3|>
<6|7|5|3|6|9|7|1|4|9|>
<6|4|9|6|3|3|3|1|7|7|>
```

matriz2.txt

```
<5|5|3|2|6|6|7|3|2|7|>
<8|5|6|0|9|3|0|5|3|9|>
<0|2|0|7|2|7|3|6|4|9|>
<8|6|9|3|7|5|2|8|9|2|>
<0|6|5|0|4|7|1|5|4|3|>
<6|6|2|9|7|4|7|8|6|5|>
<1|2|3|4|5|6|7|7|4|0|>
<6|1|3|8|5|9|6|7|4|2|>
<6|0|9|6|5|8|1|0|4|4|>
<8|3|6|1|7|2|8|6|5|4|>
```

Multiplicacion.txt

```
<154|112|146|171|194|218|136|211|158|142|>
<148|101|172|187|177|229|126|172|163|115|>
<225|112|218|179|243|235|207|219|192|152|>
<193|127|154|200|228|228|176|217|165|209|>
<87|40|127|94|100|140|48|71|89|75|>
<164|87|133|101|175|140|132|156|118|111|>
<248|197|209|233|306|309|298|342|249|221|>
<229|202|241|184|303|330|218|322|239|229|>
<273|225|249|206|340|286|263|319|251|273|>
<235|168|234|189|269|271|195|249|223|253|>
```

Suma.txt

```
<5|10|7|5|9|10|13|10|5|7|>
<8|5|8|3|14|9|4|10|12|9|>
<2|4|2|10|3|10|10|11|12|18|>
<11|12|15|3|8|12|6|13|14|5|>
<0|7|7|2|6|7|4|6|13|3|>
<8|11|2|11|7|6|11|13|8|10|>
<8|4|9|9|9|12|16|15|4|8|>
<12|8|9|15|13|9|15|15|5|5|>
<12|7|14|9|11|17|8|1|8|13|>
<14|7|15|7|10|5|11|7|12|11|>
```

MatrizInversaSuma.txt

```
Matriz inversa de la Suma:
-0.3188 0.0051  -0.2685 -0.0878 0.1311  0.4078  0.3668  -0.3073 0.1045  0.0209
0.1901  -0.0235 0.0821  0.0752  -0.0923 -0.0970 -0.2156 0.1090  -0.0297 -0.0146
0.3141  -0.0263 0.2864  0.1647  -0.2168 -0.4740 -0.3966 0.3428  -0.1094 0.0300
0.0628  -0.0373 0.1386  0.0444  -0.0548 -0.1698 -0.2029 0.2261  -0.0177 -0.0214
0.1020  0.0841  0.0249  -0.0364 -0.0277 -0.0930 -0.1050 0.0899  -0.0211 -0.0052
-0.1654 -0.0066 -0.1325 -0.0403 0.1125  0.1940  0.2283  -0.1695 0.0999  -0.0682
-0.1800 -0.0445 -0.2186 -0.1540 0.2062  0.2969  0.3409  -0.2659 0.0617  0.0524
0.1162  0.0302  0.1379  0.1112  -0.1391 -0.1805 -0.1424 0.1449  -0.0911 -0.0274
-0.2880 -0.0101 -0.2024 -0.1188 0.2493  0.3185  0.3052  -0.2479 0.0706  0.0265
0.2193  0.0309  0.1999  0.0754  -0.1932 -0.2360 -0.2512 0.1498  -0.0607 0.0182
```

MatrizInversaMultiplicacion.txt

```
Matriz inversa de la Multiplicación:
-0.0364 0.0429  0.0017  0.0141  -0.0599 0.0328  -0.0179 0.0194  -0.0312 0.0232
-0.1079 0.1451  0.1933  0.1009  -0.3116 -0.1408 -0.1786 0.1386  -0.0170 -0.0018
-0.1872 0.2884  0.6510  0.2751  -0.8390 -0.4990 -0.5239 0.3763  -0.0415 -0.0150
-0.0644 0.1177  0.2348  0.1071  -0.3148 -0.1828 -0.1942 0.1298  -0.0109 -0.0095
0.2005  -0.2975 -0.5681 -0.2512 0.8031  0.4192  0.4642  -0.3482 0.0792  -0.0263
0.0295  -0.0819 -0.2162 -0.0753 0.2735  0.1705  0.1721  -0.1030 -0.0020 0.0020
-0.0421 0.0308  0.0801  0.0392  -0.0901 -0.0655 -0.0488 0.0450  -0.0057 -0.0075
-0.1095 0.2016  0.4795  0.1973  -0.6328 -0.3625 -0.3863 0.2775  -0.0421 -0.0029
0.2738  -0.3772 -0.7872 -0.3781 1.0503  0.5863  0.6503  -0.4882 0.0665  0.0367
-0.0192 0.0232  0.0884  0.0421  -0.1066 -0.0768 -0.0685 0.0479  -0.0081 0.0067
```

EXPLICACIÓN

Se usaron archivos de texto para la comunicación entre los procesos del último punto en windows, pues debido a algunas circunstancias ajenas a nosotros no pudimos tener el tiempo suficiente para hacer este punto con memoria compartida, lo que si se usó fue los semáforos pues esos solo se usaron como en el punto 4 de linux, en el proceso hijo y en el proceso padre, pues una vez que estos dos procesos dejaran de usar el espacio no importaba quien lo tomara pues estaba libre, en este caso el que seguía ejecutándose era el proceso nieto. Las funciones utilizadas son las mismas que en linux con algunos pequeños cambios para que funcionaran en windows. Como lo fue la función getline().

III.CONCLUSIÓN

En esta práctica pudimos entender la utilidad de los semáforos para controlar la intercomunicación de procesos evitando que los procesos interfieran unos con otros. Asimismo, pudimos observar la complejidad que implica combinar algunas de las técnicas de intercomunicación como por ejemplo lo que es la memoria compartida con los semáforos. A su vez, notamos que la programación es más sencilla en el sistema operativo de Windows, ya que se utilizan menos funciones que las que ocupamos en el sistema operativo de Linux.

Con respecto a los semáforos podemos concluir que son mecanismos de intercomunicación con los que no se mueven los datos, esto es posible debido a que solo se puede consultar y modificar su valor al tener un carácter únicamente informativo. De acuerdo a lo revisado en esta práctica, un semáforo puede verse como una especie de variable positiva o nula sobre la cual se realizan solo dos operaciones:

wait.- La cual permite adquirir el semáforo o bloquearlo al decrementar su valor cuando éste vale más que 0;

signal.- La que viene siendo la operación contraria, es decir, libera o “inicializa” el semáforo.

Estas operaciones son procedimientos que se implementan como acciones indivisibles. En sistemas con un único procesador, bastará simplemente con “reprimir” las interrupciones durante la ejecución de las operaciones del semáforo.

IV.REFERENCIAS

1) https://sistemasoperativos.angelfire.com/html/2.4.2.html
2) http://www1.frm.utn.edu.ar/soperativos/Archivos/sincro.pdf
3) http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro26/comunicacin_y_sincronizacin_de_procesos.html
4) https://lsi.vc.ehu.eus/pablogn/docencia/manuales/SO/TemasSOuJaen/CONCURRENCIA/1ComunicacionySincronizacion.html