



# INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE COMPUTO

## PRACTICA 6

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

SISTEMAS OPERATIVOS

**Profesor:** Cortes Galicia Jorge

**Grupo:** 2CM9

**Integrantes:**

Beltran García Juan

Hernández Méndez Oliver Manuel

López López Rodrigo

Rangel Lozada Kevin Sebastián



INSTITUTO POLITÉCNICO NACIONAL



ESCOM

# Índice

1	Introducción Teórica .....	1
2	Desarrollo Experimental .....	2
2.1	Ejercicio 1 .....	2
2.2	Ejercicio 2 .....	5
2.3	Ejercicio 3 .....	9
2.4	Ejercicio 4 .....	11
2.4.1	Codificación en Linux .....	11
2.4.2	Codificación en Windows .....	22
3	Conclusiones .....	32

# 1 Introducción Teórica

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

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

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

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

Problemas que resuelven principalmente los semáforos:

- La exclusión mutua.
- Sincronización de Procesos

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

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

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

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

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

## 2 Desarrollo Experimental

### 2.1 Ejercicio 1

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

#### Función **semget()**

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

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

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

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

#### Sintaxis:

```
int semget ( key_t key, int nsems, int semflg );
```

#### Parámetros:

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

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

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

- **IPC\_CREAT** Si este flag está activo, se creará el conjunto de semáforos, en caso de que no hayan sido ya creados.
- **IPC\_EXCL** Esta bandera se utiliza en conjunción con **IPC\_CREAT**, para lograr que **semget** de un error en el caso de que se intente crear un grupo de semáforos que ya exista. En este caso, **semget** devolvería -1 e inicializaría la variable **errno** con un valor **EEXIST**.

#### Retorno:

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

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

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

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

```
struct semid_ds {
    struct ipc_perm sem_perm; /* estructura permiso */
    int *pad;                /* usado por sistema */
    ushort sem_nsems;        /* número semáforos */
    time_t sem_otime;        /* tiempo último semop */
    time_t sem_ctime;        /* tiempo último cambio */
}
```

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

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

## Función semop()

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

```
unsigned short semval; /* valor del semáforo */
unsigned short semzcnt; /* # esperando por cero */
unsigned short semncnt; /* # esperando por incremento */
pid_t sempid; /* proceso que hizo la última operación */
```

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

```
unsigned short sem_num; /* número de semáforo */
short sem_op; /* operación sobre el semáforo */
short sem_flg; /* banderas o indicadores para la operación */
```

### Sintaxis:

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

### Parámetros:

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

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

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

### Retorno:

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

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

E2BIG -- El argumento nsops es mayor que SEMOPM, el número máximo de operaciones permitidas por llamada del sistema.

EACCES -- El proceso invocador no tiene permisos de acceso al semáforo como se requiere por una de las operaciones especificadas.

EAGAIN -- Una operación no puede ser ejecutada inmediatamente y IPC\_NOWAIT ha sido invocada en su sem\_flg.

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

## Teoria Semaforos

Los semáforos son un mecanismo de sincronización de procesos inventados por Edsger Dijkstra en 1965. Los semáforos permiten al programador **asistir al planificador del sistema operativo en su toma de decisiones** de manera que permiten sincronizar la ejecución de dos o más procesos. A diferencia de los cerrojos, los semáforos nos ofrecen un mecanismo de espera no ocupada.

Los semáforos son un tipo de datos que están compuestos por dos atributos:

- Un contador, que siempre vale  $\geq 0$ .
- Una cola de procesos inicialmente vacía.

Y disponen de dos operaciones básicas que pasamos a describir en pseudocódigo:

- Down
- Up

Tipos de semáforos

Dependiendo de la función que cumple el semáforo, vamos a diferenciar los siguientes tipos:

- Semáforo de exclusión mutua, inicialmente su contador vale 1 y permite que haya un único proceso simultáneamente dentro de la sección crítica.
- Semáforo contador, permiten llevar la cuenta del número de unidades de recurso compartido disponible, que va desde 0 hasta N.
- Semáforo de espera, generalmente se emplea para forzar que un proceso pase a estado bloqueado hasta que se cumpla la condición que le permite ejecutarse. Por lo general, el contador vale 0 inicialmente, no obstante, podría tener un valor distinto de cero.

## 2.2 Ejercicio 2

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

### Explicación

El programa consta de la comunicación entre tres procesos a través de tres secciones de memoria compartida de 400 bits. El control de acceso a el recurso de memoria compartida está implementando a través de semáforos, a través de la llamada semop, nosotros podemos bloquear o desbloquear el semáforo, esto dependiendo del valor colocado en el atributo sem\_op de la estructura sembuf, como se puede observar al recibir las matrices a multiplicar el proceso hijo bloquea el semáforo, realiza las operaciones pertinentes y lo desbloquea, lo mismo ocurre con el proceso nieto, en este caso con las matrices a sumar y finalmente con el proceso padre que despliega los resultados y guarda en un archivo.

### Codigo capturado

```

1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <sys/ipc.h>
4  #include <sys/sem.h>
5  #include <stdlib.h>
6
7  int main(void)
8  {
9
10     int i,j;
11     int pid;
12     int semid;
13
14     key_t llave = 1234;
15
16     int semban = IPC_CREAT | 0666;
17     int nsems = 1;
18     int nsops;
19     struct sembuf *sops = (struct sembuf *) malloc(2*sizeof(struct sembuf));
20
21     printf("Iniciando semaforo...\n");
22
23     if ((semid = semget(llave, nsems, semban)) == -1) {
24         perror("semget: error al iniciar semaforo");
25         exit(1);
26     }
27     else
28         printf("Semaforo iniciado...\n");
29
30
31     if ((pid = fork()) < 0) {
32         perror("fork: error al crear proceso\n");
33         exit(1);
34     }
35
36
37     if (pid == 0) {
38         i = 0;
39         while (i < 3) {
40             nsops = 2;
41             sops[0].sem_num = 0;
42             sops[0].sem_op = 0;
43             sops[0].sem_flg = SEM_UNDO;
44
45             sops[1].sem_num = 0;
46             sops[1].sem_op = 1;
47             sops[1].sem_flg = SEM_UNDO | IPC_NOWAIT;
48             printf("semop: hijo llamando a semop(%d, &sops, %d) con:", semid, nsops);
49
50             for (j = 0; j < nsops; j++) {
51                 printf("\n\t%sops[%d].sem_num = %d, ", j, sops[j].sem_num);
52                 printf("sem_op = %d, ", sops[j].sem_op);
53                 printf("sem_flg = %#o\n", sops[j].sem_flg);
54             }
55

```

```

56         if ((j = semop(semid, sops, nsops)) == -1) {
57             perror("semop: error en operacion del semaforo\n");
58         }
59         else {
60             printf("\tsemop: regreso de semop() %d\n", j);
61             printf("\n\nProceso hijo toma el control del semaforo: %d/3 veces\n", i+1);
62             sleep(5);
63             nsops = 1;
64             sops[0].sem_num = 0;
65             sops[0].sem_op = -1;
66             sops[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
67             if ((j = semop(semid, sops, nsops)) == -1) {
68                 perror("semop: error en operacion del semaforo\n");
69             }
70             else
71                 printf("Proceso hijo regresa el control del semaforo: %d/3 veces\n", i+1);
72             sleep(5);
73         }
74         ++i;
75     }
76 }

```



```

77 else {
78
79
80     i = 0;
81     while (i < 3) {
82
83         nsops = 2;
84         sops[0].sem_num = 0;
85         sops[0].sem_op = 0;
86         sops[0].sem_flg = SEM_UNDO;
87
88         sops[1].sem_num = 0;
89         sops[1].sem_op = 1;
90         sops[1].sem_flg = SEM_UNDO | IPC_NOWAIT;
91         printf("\nsemop: Padre llamando semop(%d, &sops, %d) con:", semid, nsops);
92         for (j = 0; j < nsops; j++) {
93             printf("\n\tsops[%d].sem_num = %d, ", j, sops[j].sem_num);
94             printf("sem_op = %d, ", sops[j].sem_op);
95             printf("sem_flg = %o\n", sops[j].sem_flg);
96         }
97         if ((j = semop(semid, sops, nsops)) == -1) {
98             perror("semop: error en operacion del semaforo\n");
99         }
100     }
101     else {
102         printf("semop: regreso de semop() %d\n", j);
103         printf("Proceso padre toma el control del semaforo: %d/3 veces\n", i+1);
104         sleep(5);
105         nsops = 1;
106         sops[0].sem_num = 0;
107         sops[0].sem_op = -1;
108         sops[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
109         if ((j = semop(semid, sops, nsops)) == -1) {
110             perror("semop: error en semop()\n");
111         }
112         else {
113             printf("Proceso padre regresa el control del semaforo: %d/3 veces\n", i+1);
114             sleep(5);
115         }
116     }
117     ++i;
118 }
119

```

## Compilación y ejecución del programa

```
juanbel@LAPTOP-V8EFNE6L:/mnt/c/users/juan-/Desktop/practica6_prog$ gcc programa2.c -o prog2
programa2.c: In function 'main':
programa2.c:32:13: warning: implicit declaration of function 'fork' [-Wimplicit-function-declaration]
    if ((pid = fork()) < 0) {
                ^~~~~~
programa2.c:63:5: warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
    sleep(5);
    ^~~~~~
juanbel@LAPTOP-V8EFNE6L:/mnt/c/users/juan-/Desktop/practica6_prog$ ./prog2
Iniciando semaforo...
Semaforo iniciado...

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

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

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

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

Proceso padre regresa el control del semaforo: 2/3 veces
    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

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

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

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

    sops[1].sem_num = 0, sem_op = 1, sem_flg = 014000
semop: regreso de semop() 0

Proceso hijo toma el control del semaforo: 3/3 veces
juanbel@LAPTOP-V8EFNE6L:/mnt/c/users/juan-/Desktop/practica6_prog$
juanbel@LAPTOP-V8EFNE6L:/mnt/c/users/juan-/Desktop/practica6_prog$
Proceso hijo regresa el control del semaforo: 3/3 veces
juanbel@LAPTOP-V8EFNE6L:/mnt/c/users/juan-/Desktop/practica6_prog$
```

## 2.3 Ejercicio 3

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

### Explicación

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

### Código capturado

#### Programa Padre

```
1  #include <windows.h> /*Programa padre*/
2  #include <stdio.h>
3  int main(int argc, char *argv[]){
4
5      STARTUPINFO si; /* Estructura de información inicial para Windows */
6      PROCESS_INFORMATION pi; /* Estructura de información del adm. de procesos */
7      HANDLE hSemaphore;
8      int i=1;
9      ZeroMemory(&si, sizeof(si));
10     si.cb = sizeof(si);
11     ZeroMemory(&pi, sizeof(pi));
12
13
14     if(argc!=2){
15         printf("Usar: %s Nombre_programa_hijo\n", argv[0]);
16         return;
17     }
18
19     // Creación del semáforo
20     if((hSemaphore = CreateSemaphore(NULL, 1, 1, "Semaforo")) == NULL){
21         printf("Falla al invocar CreateSemaphore: %d\n", GetLastError());
22         return -1;
23     }
24
25     // Creación proceso hijo
26     if(!CreateProcess(NULL, argv[1], NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)){
27         printf("Falla al invocar CreateProcess: %d\n", GetLastError() );
28         return -1;
29     }
30 }
```

```

31
32 while(i<4){
33     // Prueba del semáforo
34     WaitForSingleObject(hSemaforo, INFINITE);
35
36     //Sección crítica
37     printf("Soy el padre entrando %i de 3 veces al semaforo\n",i);
38     Sleep(5000);
39
40     //Liberación el semáforo
41     if (!ReleaseSemaphore(hSemaforo, 1, NULL) ){
42         printf("Falla al invocar ReleaseSemaphore: %d\n", GetLastError());
43     }
44     printf("Soy el padre liberando %i de 3 veces al semaforo\n",i);
45     Sleep(5000);
46
47     i++;
48 }
49 // Terminación controlada del proceso e hilo asociado de ejecución
50 CloseHandle(pi.hProcess);
51 CloseHandle(pi.hThread);
52 }
53

```

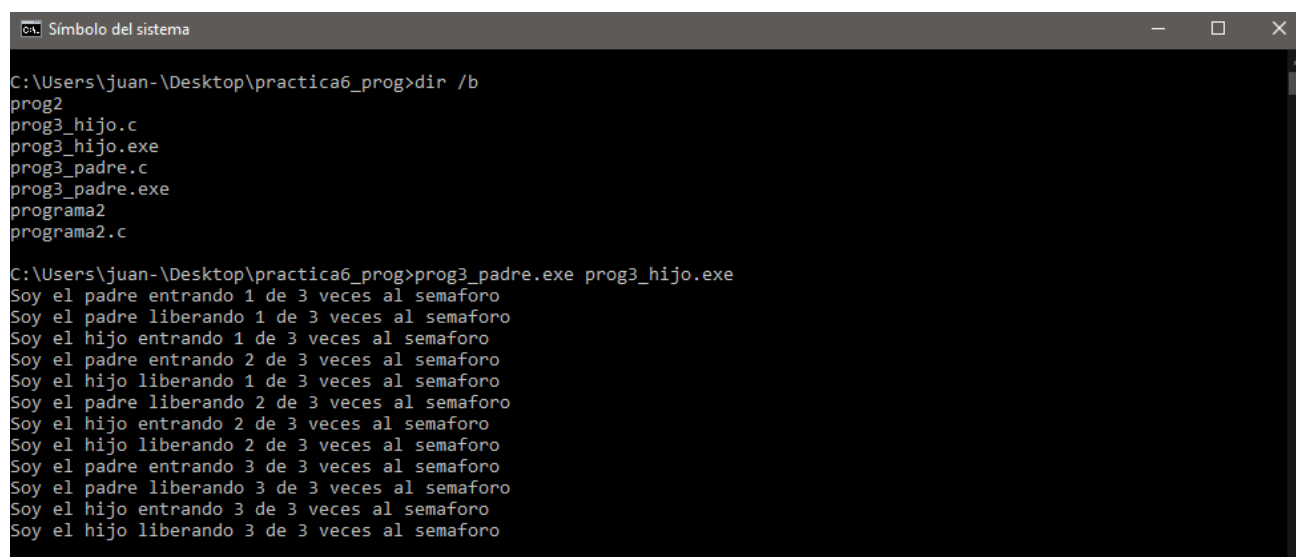
## Programa Hijo

```

1  #include <windows.h> /*Programa hijo*/
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int main()
6  {
7      HANDLE hSemaforo;
8      int i=1;
9      // Apertura del semáforo
10     if((hSemaforo = OpenSemaphore(SEMAPHORE_ALL_ACCESS, FALSE, "Semaforo")) == NULL){
11         printf("Falla al invocar OpenSemaphore: %d\n", GetLastError());
12         return -1;
13     }
14     while(i<4){
15         // Prueba del semáforo
16         WaitForSingleObject(hSemaforo, INFINITE);
17         //Sección crítica
18         printf("Soy el hijo entrando %i de 3 veces al semaforo\n",i);
19         Sleep(5000);
20
21         //Liberación el semáforo
22         if (!ReleaseSemaphore(hSemaforo, 1, NULL) ){
23             printf("Falla al invocar ReleaseSemaphore: %d\n", GetLastError());
24         }
25         printf("Soy el hijo liberando %i de 3 veces al semaforo\n",i);
26         Sleep(5000);
27
28         i++;
29     }
30 }

```

## Compilación y ejecución del programa



```
C:\Users\juan-\Desktop\practica6_prog>dir /b
prog2
prog3_hijo.c
prog3_hijo.exe
prog3_padre.c
prog3_padre.exe
programa2
programa2.c

C:\Users\juan-\Desktop\practica6_prog>prog3_padre.exe prog3_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 padre entrando 2 de 3 veces al semaforo
Soy el hijo liberando 1 de 3 veces al semaforo
Soy el padre liberando 2 de 3 veces al semaforo
Soy el hijo entrando 2 de 3 veces al semaforo
Soy el hijo liberando 2 de 3 veces al semaforo
Soy el padre entrando 3 de 3 veces al semaforo
Soy el padre liberando 3 de 3 veces al semaforo
Soy el hijo entrando 3 de 3 veces al semaforo
Soy el hijo liberando 3 de 3 veces al semaforo
```

## 2.4 Ejercicio 4

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

### 2.4.1 Codificación en Linux

## Codigo fuente

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

*seminit.c	×	*ejercicio6.c
1		#include <stdio.h>
2		#include <stdlib.h>
3		#include <errno.h>
4		#include <sys/types.h>
5		#include <sys/ipc.h>
6		#include <sys/sem.h>
7		
8		int main(){
9		
10		key_t key;
11		int semid;
12		union semun{
13		
14		int val;
15		struct semid_ds *buf;
16		unsigned short *array;
17		
18		} arg;
19		
20		//definicion de la llave
21		if((key = ftok("p6.c", 'J'))== -1){
22		perror("ftok");
23		exit(1);
24		}
25		
26		//crear un bloque de semaforos
27		if((semid = semget(key, 3, 0666   IPC_CREAT)) == -1){
28		perror("semget");
29		exit(1);
30		}
31		
32		//inicializamos el semaforo #0 a 1
33		arg.val = 1;
34		if(semctl(semid, 0, SETVAL, arg) == -1){
35		perror("semctl");
36		exit(1);
37		}
38		
39		//inicializamos el semaforo #1 a 1
40		arg.val = 1;
41		if(semctl(semid, 0, SETVAL, arg) == -1){
42		perror("semctl");
43		exit(1);
44		}
45		
46		//inicializamos el semaforo #2 a 1
47		arg.val = 1;
48		if(semctl(semid, 0, SETVAL, arg) == -1){
49		perror("semctl");
50		exit(1);
51		}
52		
53		return 0;
54		
55		}

## Archivo: p6.c (Código principal)

```
semdemo.c  ×  p6.c  ×  invMulti.c

1 #include <stdio.h>
2 #include <unistd.h> //libreria fork
3 #include <stdlib.h>
4 #include <errno.h>
5 #include <math.h>
6 #include <sys/types.h>
7 #include <sys/ipc.h>
8 #include <sys/sem.h>
9 #include <sys/wait.h>
10 #include <sys/shm.h>
11
12 #define SHM_SIZE 400
13
14 void insertM(double mN[10][10]);
15 void showM(double mN[10][10]);
16 void multi(double m1[10][10], double m2[10][10], double mr[10][10]);
17 void suma(double m1[10][10], double m2[10][10], double mr[10][10]);
18
19 struct data{
20     double m1[10][10];
21     double m2[10][10];
22     double mr[10][10];
23 };
24
25 //Conversión de valores matriz double a float
26 void conversionMat(double matrix[10][10], float transf[][10]){
27
28     for(int i=0; i<10; i++){
29         for(int j=0; j<10; j++){
30
31             transf[i][j]= (float) matrix[i][j];
32
33         }
34     }
35 }
36
37 float determinant(float a[10][10], float k) {
38     float s = 1, det = 0, b[10][10];
39     int i, j, m, n, c;
40     if (k == 1)
41     {
42         return (a[0][0]);
43     }
44     else
45     {
46         det = 0;
47         for (c = 0; c < k; c++)
48         {
49             m = 0;
50             n = 0;
51             for (i = 0; i < k; i++)
52             {
53                 for (j = 0; j < k; j++)
54                 {
55                     b[i][j] = 0;
56                     if (i != 0 && j != c)
57                     {
58                         b[m][n] = a[i][j];
59                         if (n < (k - 2))
60                             n++;
61                         else
62                             {
```

```

63         n = 0;
64         m++;
65     }
66 }
67 }
68 }
69 det = det + s * (a[0][c] * determinant(b, k - 1));
70 s = -1 * s;
71 }
72 }
73
74 return (det);
75 }
76
77 void transpose(float num[10][10], float fac[10][10], float r, FILE* archivo){
78     int i, j;
79     r = 10;
80
81     float b[10][10], inverse[10][10], d;
82
83     for (i = 0; i < r; i++)
84     {
85         for (j = 0; j < r; j++)
86         {
87             b[i][j] = fac[j][i];
88         }
89     }
90     d = determinant(num, r);
91     for (i = 0; i < r; i++)
92     {
93         for (j = 0; j < r; j++)
94         {
95             inverse[i][j] = b[i][j] / d;
96         }
97     }
98
99     fprintf(archivo, "The inverse of matrix is : \n");
100
101     for (i = 0; i < r; i++) {
102         for (j = 0; j < r; j++) {
103             fprintf(archivo, "\t%f", inverse[i][j]);
104         }
105         fprintf(archivo, "\n");
106     }
107 }
108
109 void cofactor(float num[10][10], float f, FILE* archivo){
110     float b[10][10], fac[10][10];
111     int p, q, m, n, i, j;
112     for (q = 0; q < f; q++)
113     {
114         for (p = 0; p < f; p++)
115         {
116             m = 0;
117             n = 0;
118             for (i = 0; i < f; i++)
119             {
120                 for (j = 0; j < f; j++)
121                 {
122                     if (i != q && j != p)
123                     {
124                         b[m][n] = num[i][j];
125                         if (n < (f - 2))
126                             n++;

```



```

126         n++;
127     else
128     {
129         n = 0;
130         m++;
131     }
132 }
133 }
134 }
135     fac[q][p] = pow(-1, q + p) * determinant(b, f - 1);
136 }
137 }
138 transpose(num, fac, f, archivo);
139 }
140
141
142 int main(){
143
144     int id;
145     key_t key, key_MC, key_MC2;
146     int semid1, semid2, semid3;
147     int shmid1, shmid2;
148
149     //llave del espacio de memoria compartida 1
150     if((key_MC = ftok("p6.c", 'D'))==-1){
151         perror("ftok");
152         exit(1);
153     }
154
155     //Creacion de espacio de memoria compartida 1
156     shmid1 = shmget(key_MC, SHM_SIZE, 0666 | IPC_CREAT);
157     if(shmid1 == -1){
158         perror("Fallo shmget");
159         exit(0);
160     }
161
162     //Adjuntamos la estructura al espacio de memoria compartida 1
163     struct data *com1 = shmat(shmid1, 0, 0);
164     if(com1 == (void*)-1){
165         perror("Fallo shmat");
166         exit(0);
167     }
168
169     //llave del espacio de memoria compartida 2
170     if((key_MC2 = ftok("p6.c", 'E'))==-1){
171         perror("ftok");
172         exit(1);
173     }
174
175     //Creacion de espacio de memoria compartida 2
176     shmid2 = shmget(key_MC2, SHM_SIZE, 0666 | IPC_CREAT);
177     if(shmid2 == -1){
178         perror("Fallo shmget");
179         exit(0);
180     }
181
182     //Adjuntamos la estructura al espacio de memoria compartida 2
183     struct data *com2 = shmat(shmid2, 0, 0);
184     if(com2 == (void*)-1){
185         perror("Fallo shmat");
186         exit(0);
187     }

```

```

188     }
189
190     //definicion de la llave
191     if((key = ftok("p6.c", 'J'))==-1){
192         perror("ftok");
193         exit(1);
194     }
195
196     //acceder al grupo de semaforos
197     if((semid3 = semget(key, 3, 0)) == -1){
198         perror("semget");
199         exit(1);
200     }
201
202
203
204     //Llenamos las matrices
205     insertM(com1->m1);
206     insertM(com1->m2);
207
208     //bloqueo semaforo 0
209     struct sembuf sb = {0, -1, 0};
210
211
212     //creacion de un hilo
213     id = fork();
214
215     if(id == 0){
216         printf("%s %d %s\n", "Soy el hijo id = ", getpid(), " y recibo las siguientes dos matrices y las multiplico");
217
218         //Bloqueo semaforo
219         if(semop(semid3, &sb, 1) == -1){
220             perror("semop");
221             exit(1);
222         }
223
224         //Observamos las matrices
225         printf("Matriz 1:\n");
226         showM(com1->m1);
227         printf("\n Matriz 2:\n");
228         showM(com1->m2);
229         //Multiplicacion
230
231
232         multi(com1->m1, com1->m2, com1->mr);
233
234         //Desbloqueo semaforo
235         sb.sem_op = 1;
236         if(semop(semid3, &sb, 1) == -1){
237             perror("semop");
238             exit(1);
239         }
240
241
242         //Llenamos las matrices
243         insertM(com2->m1);
244         insertM(com2->m2);
245         //bloqueo semaforo 1
246         struct sembuf sb1 = {1, -1, 0};
247
248         if(fork()==0){
249
250             //Bloqueo semaforo

```

```

251         if(semop(semid3, &sb, 1) == -1){
252             perror("semop");
253             exit(1);
254         }
255
256         printf("%s %d %s\n", "Soy el nieto id = ", getpid(), "Y recibo las dos matrices siguientes y las sumo");
257         printf("Matriz 3:\n");
258         showM(com2->m1);
259         printf("\n Matriz 4:\n");
260         showM(com2->m2);
261
262
263         suma(com2->m1, com2->m2, com2->mr);
264
265         //Desbloqueo semaforo
266         sb.sem_op = 1;
267         if(semop(semid3, &sb, 1) == -1){
268             perror("semop");
269             exit(1);
270         }
271
272         sleep(5);
273     }
274     else{
275         wait(NULL);
276
277     }
278     sleep(5);
279     sb.sem_op = 1; //UP
280
281     if(semop(semid3, &sb, 1) == -1){
282         perror("semop");
283         exit(1);
284     }
285
286 }
287 else{
288
289     wait(NULL);
290
291     //Bloqueo semaforo
292     sb.sem_op = -1;
293     if(semop(semid3, &sb, 1) == -1){
294         perror("semop");
295         exit(1);
296     }
297
298     printf("%s %d %s\n", "Soy el padre id = ", getpid(), " y estos son los resultado que obtuve");
299     printf("\nMultiplicación:\n");
300     showM(com1->mr);
301     printf("\nSuma:\n");
302     showM(com2->mr);
303
304     //Desbloqueo semaforo
305     sb.sem_op = 1;
306     if(semop(semid3, &sb, 1) == -1){
307         perror("semop");
308         exit(1);
309     }
310
311     //Determinante de la Matriz Multiplicación
312     float transf[10][10];
313     conversionMat(com1->mr, transf);
314     FILE* archivo = fopen("invMulti.txt", "w");
315     cofactor(transf, 10, archivo);
316
317     //Determinante de la Matriz Suma
318     float transf1[10][10];
319     conversionMat(com2->mr, transf1);
320     FILE* archivo1 = fopen("invSuma.txt", "w");
321     cofactor(transf1, 10, archivo1);
322
323 }

```

```

323     }
324
325     //Liberación de espacios de memoria
326     if(shmdt(com1)==-1){
327         perror("shmdt in shrmem2_sysV failed");
328         exit(0);
329     }
330
331     if(shmctl(shmid1, IPC_RMID, NULL)==-1){
332         perror("shmctl in shrmem2_sysV failed");
333         exit(0);
334     }
335
336
337     return 0;
338
339 }
340
341 void insertM(double mN[10][10]){
342     for(int i=0 ; i<10 ; i++){
343         for(int j = 0; j < 10; j++){
344             mN[i][j] = rand() % (9-1+1)+1;
345         }
346     }
347 }
348
349 void showM(double mN[10][10]){
350     for(int i=0 ; i<10 ; i++){
351         for(int j = 0; j < 10; j++){
352             printf("%f ", mN[i][j]);
353         }
354         printf("\n");
355     }
356 }
357
358 void multi(double m1[10][10], double m2[10][10], double mr[10][10]){
359     for(int i = 0 ; i < 10 ; i++){
360         for(int j = 0 ; j < 10 ; j++){
361             mr[i][j] = (m1[i][0]*m2[0][j]) + (m1[i][1]*m2[1][j]) + (m1[i][2]*m2[2][j])
362             + (m1[i][3]*m2[3][j])+(m1[i][4]*m2[4][j]) + (m1[i][5]*m2[5][j])
363             + (m1[i][6]*m2[6][j]) + (m1[i][7]*m2[7][j]) + (m1[i][8]*m2[8][j])
364             + (m1[i][9]*m2[9][j]);
365         }
366     }
367 }
368
369 void suma(double m1[10][10], double m2[10][10], double mr[10][10]){
370     for(int i = 0 ; i < 10 ; i++){
371         for(int j = 0 ; j < 10 ; j++){
372             mr[i][j] = m1[i][j] + m2[i][j];
373         }
374     }
375 }
376
377
378
379
380
381
382
383

```

## Archivo: semrm.c (Borra semáforo)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <errno.h>
4 #include <sys/types.h>
5 #include <sys/ipc.h>
6 #include <sys/sem.h>
7
8
9 int main(){
10
11     key_t key;
12     int semid;
13     union semun{
14
15         int val;
16         struct semid_ds *buf;
17         unsigned short *array;
18
19     } arg;
20
21     //definicion de la llave
22     if((key = ftok("p6.c", 'J'))== -1){
23         perror("ftok");
24         exit(1);
25     }
26
27     if((semid = semget(key, 1, 0)) == -1){
28         perror("semget");
29         exit(1);
30     }
31
32     //removemos el semaforo
33     if(semctl(semid, 0, IPC_RMID, arg) == -1){
34         perror("semctl");
35         exit(1);
36     }
37
38     return 0;
39
40 }
41
42
```

## Compilación y ejecución del programa

Primero ejecutamos el archivo que inicia los semáforos

```
oliver@oliver-VirtualBox:~/Documents/practica6$ ./semitit.out
```

Ejecución código principal

```
oliver@oliver-VirtualBox:~/Documents/practica6$ ./p6.out
Soy el hijo id = 35009 y recibo las siguientes dos matrices y las multiplico
Matriz 1:
2.000000 8.000000 1.000000 8.000000 6.000000 8.000000 2.000000 4.000000 7.000000 2.000000
6.000000 5.000000 6.000000 8.000000 6.000000 5.000000 7.000000 1.000000 8.000000 2.000000
9.000000 9.000000 7.000000 7.000000 9.000000 9.000000 9.000000 5.000000 2.000000 2.000000
6.000000 1.000000 1.000000 4.000000 6.000000 4.000000 2.000000 8.000000 5.000000 8.000000
7.000000 1.000000 1.000000 3.000000 6.000000 5.000000 6.000000 3.000000 3.000000 4.000000
3.000000 2.000000 2.000000 9.000000 9.000000 1.000000 6.000000 6.000000 5.000000 5.000000
7.000000 1.000000 6.000000 7.000000 3.000000 9.000000 8.000000 4.000000 5.000000 3.000000
1.000000 1.000000 1.000000 1.000000 3.000000 7.000000 3.000000 6.000000 7.000000 6.000000
8.000000 7.000000 7.000000 9.000000 6.000000 4.000000 7.000000 3.000000 9.000000 2.000000
7.000000 7.000000 9.000000 1.000000 2.000000 2.000000 8.000000 1.000000 4.000000 3.000000

Matriz 2:
1.000000 2.000000 3.000000 2.000000 9.000000 4.000000 6.000000 3.000000 7.000000 1.000000
8.000000 3.000000 8.000000 3.000000 9.000000 2.000000 7.000000 6.000000 2.000000 6.000000
5.000000 7.000000 1.000000 5.000000 7.000000 3.000000 4.000000 3.000000 1.000000 5.000000
4.000000 8.000000 6.000000 4.000000 7.000000 6.000000 5.000000 3.000000 6.000000 3.000000
2.000000 4.000000 3.000000 9.000000 4.000000 3.000000 1.000000 1.000000 8.000000 3.000000
5.000000 4.000000 7.000000 3.000000 6.000000 2.000000 3.000000 7.000000 5.000000 3.000000
3.000000 8.000000 9.000000 6.000000 2.000000 6.000000 2.000000 5.000000 9.000000 5.000000
7.000000 8.000000 6.000000 9.000000 7.000000 1.000000 9.000000 5.000000 1.000000 8.000000
5.000000 3.000000 9.000000 2.000000 6.000000 3.000000 4.000000 8.000000 9.000000 8.000000
9.000000 2.000000 4.000000 8.000000 6.000000 3.000000 4.000000 7.000000 7.000000 1.000000
Soy el nieto id = 35010 Y recibo las dos matrices siguientes y las sumo
Matriz 3:
3.000000 2.000000 8.000000 8.000000 9.000000 3.000000 6.000000 8.000000 8.000000 5.000000
4.000000 1.000000 7.000000 1.000000 3.000000 1.000000 3.000000 4.000000 7.000000 3.000000
9.000000 6.000000 2.000000 3.000000 2.000000 7.000000 3.000000 3.000000 3.000000 8.000000
4.000000 3.000000 7.000000 9.000000 8.000000 4.000000 1.000000 5.000000 3.000000 6.000000
9.000000 4.000000 6.000000 4.000000 5.000000 6.000000 5.000000 5.000000 9.000000 2.000000
7.000000 8.000000 5.000000 7.000000 8.000000 6.000000 4.000000 9.000000 8.000000 4.000000
7.000000 9.000000 6.000000 2.000000 7.000000 3.000000 6.000000 7.000000 7.000000 6.000000
1.000000 4.000000 9.000000 6.000000 7.000000 2.000000 3.000000 9.000000 7.000000 9.000000
1.000000 2.000000 8.000000 3.000000 8.000000 4.000000 8.000000 3.000000 3.000000 5.000000
4.000000 7.000000 2.000000 1.000000 9.000000 8.000000 3.000000 3.000000 3.000000 7.000000

Matriz 4:
8.000000 3.000000 1.000000 7.000000 7.000000 5.000000 7.000000 7.000000 5.000000 4.000000
6.000000 3.000000 5.000000 2.000000 4.000000 4.000000 6.000000 9.000000 4.000000 6.000000
4.000000 7.000000 2.000000 4.000000 5.000000 8.000000 9.000000 5.000000 1.000000 3.000000
2.000000 8.000000 3.000000 9.000000 3.000000 7.000000 3.000000 9.000000 4.000000 5.000000
1.000000 8.000000 5.000000 6.000000 9.000000 8.000000 7.000000 3.000000 8.000000 1.000000
9.000000 9.000000 5.000000 1.000000 3.000000 8.000000 8.000000 3.000000 3.000000 6.000000
3.000000 3.000000 4.000000 3.000000 9.000000 4.000000 1.000000 2.000000 4.000000 2.000000
6.000000 2.000000 9.000000 9.000000 7.000000 9.000000 7.000000 2.000000 2.000000 5.000000
9.000000 8.000000 5.000000 5.000000 6.000000 5.000000 3.000000 2.000000 5.000000 3.000000
7.000000 7.000000 5.000000 8.000000 8.000000 3.000000 3.000000 8.000000 4.000000 4.000000
```



Soy el padre id = 35008 y estos son los resultado que obtuve

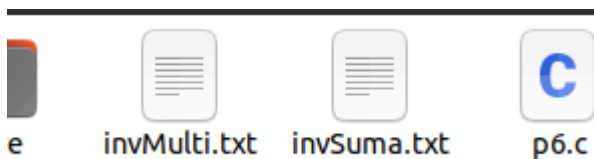
Multiplicación:

```
242.000000 228.000000 306.000000 221.000000 311.000000 152.000000 218.000000 243.000000 266.000000 221.000000
231.000000 269.000000 314.000000 241.000000 332.000000 201.000000 219.000000 249.000000 329.000000 232.000000
297.000000 344.000000 375.000000 335.000000 427.000000 233.000000 295.000000 295.000000 365.000000 276.000000
226.000000 205.000000 240.000000 260.000000 284.000000 138.000000 213.000000 219.000000 264.000000 181.000000
159.000000 181.000000 216.000000 204.000000 229.000000 139.000000 156.000000 177.000000 255.000000 142.000000
218.000000 259.000000 270.000000 282.000000 278.000000 177.000000 203.000000 205.000000 298.000000 205.000000
228.000000 280.000000 302.000000 247.000000 321.000000 193.000000 222.000000 253.000000 310.000000 215.000000
199.000000 165.000000 226.000000 196.000000 212.000000 101.000000 158.000000 210.000000 213.000000 170.000000
272.000000 309.000000 357.000000 277.000000 396.000000 225.000000 273.000000 284.000000 360.000000 275.000000
204.000000 212.000000 238.000000 197.000000 281.000000 155.000000 193.000000 207.000000 234.000000 192.000000
```

Suma:

```
11.000000 5.000000 9.000000 15.000000 16.000000 8.000000 13.000000 15.000000 13.000000 9.000000
10.000000 4.000000 12.000000 3.000000 7.000000 5.000000 9.000000 13.000000 11.000000 9.000000
13.000000 13.000000 4.000000 7.000000 7.000000 15.000000 12.000000 8.000000 4.000000 11.000000
6.000000 11.000000 10.000000 18.000000 11.000000 11.000000 4.000000 14.000000 7.000000 11.000000
10.000000 12.000000 11.000000 10.000000 14.000000 14.000000 12.000000 8.000000 17.000000 3.000000
16.000000 17.000000 10.000000 8.000000 11.000000 14.000000 12.000000 12.000000 11.000000 10.000000
10.000000 12.000000 10.000000 5.000000 16.000000 7.000000 7.000000 9.000000 11.000000 8.000000
7.000000 6.000000 18.000000 15.000000 14.000000 11.000000 10.000000 11.000000 9.000000 14.000000
10.000000 10.000000 13.000000 8.000000 14.000000 9.000000 11.000000 5.000000 8.000000 8.000000
11.000000 14.000000 7.000000 9.000000 17.000000 11.000000 6.000000 11.000000 7.000000 11.000000
```

Archivos creados:



Inversa de la multiplicación

	semdemo.c	×	p6.c	×	invMulti.txt	×
1	The inverse of matrix is :					
2	0.048185	0.079104	-0.040438	0.055047	-0.054025	0.018853
3	-0.016251	0.067630	0.013178	0.040420	-0.068185	-0.009937
4	0.036925	-0.052223	-0.004369	-0.035980	0.043547	0.012452
5	-0.016881	-0.039162	0.024799	-0.033329	0.037636	0.002146
6	-0.031199	0.081913	0.049609	0.022753	-0.030788	-0.034428
7	0.039258	-0.111469	-0.041741	-0.066621	0.091795	0.036692
8	0.032135	-0.119119	-0.050235	-0.011986	0.038594	0.023160
9	-0.027153	-0.046589	0.011365	-0.031495	0.034606	-0.015031
10	-0.006872	0.070371	-0.010994	0.048928	-0.046009	-0.007915
11	-0.042385	0.000502	0.021121	-0.017465	-0.008982	-0.003360

Inversa de la suma

	invSuma.txt ~/Documents/practica6									
1	The inverse of matrix is :									
2	0.203291	-0.484715	-0.568528	-0.611307	-0.143093	1.073090	-0.318022	0.646558	-0.558143	0.287897
3	-0.092059	0.156775	0.243129	0.328307	0.003046	-0.379094	0.251841	-0.337712	0.284100	-0.240544
4	-0.099120	0.195714	-0.026973	0.119662	0.042002	-0.149381	-0.188168	-0.155478	0.272518	0.089162
5	0.128383	-0.296199	-0.188179	-0.170912	-0.085050	0.431416	0.008496	0.278278	-0.250484	-0.050787
6	-0.032063	0.140707	0.052553	0.076620	0.053975	-0.267599	-0.083949	-0.170537	0.201663	0.141925
7	-0.082330	0.110116	-0.011010	-0.044565	0.132046	-0.082270	-0.282164	0.005799	0.013981	0.259282
8	-0.095916	0.408057	0.465542	0.476510	0.071984	-0.829449	0.177682	-0.597087	0.586535	-0.258891
9	-0.167338	0.595250	0.302557	0.506784	0.133646	-0.823645	-0.164961	-0.678601	0.627145	0.080408
10	0.152353	-0.474986	-0.202300	-0.391278	-0.075246	0.650579	0.310479	0.574183	-0.679967	-0.175424
11	0.126496	-0.440152	-0.067740	-0.334400	-0.158415	0.509377	0.392660	0.551168	-0.611927	-0.212211

## 2.4.2 Codificación en Windows

### Codigo fuente

#### Archivo: Servidor.c

```
1  #include <windows.h> /* Servidor de la memoria compartida */
2  #include <stdio.h> /* (ejecutar el servidor antes de ejecutar el cliente)*/
3  #include "Matrices.h"
4  #define TAM_MEM 400 /*Tamaño de la memoria compartida en bytes */
5  int main(void)
6  {
7      HANDLE hArchMapeo;
8      char *idMemCompartida = "MemoriaCompatida";
9      int *apDatos, *apTrabajo, c;
10     if ((hArchMapeo = CreateFileMapping(
11         INVALID_HANDLE_VALUE, // usa memoria compartida
12         NULL, // seguridad por default
13         PAGE_READWRITE, // acceso lectura/escritura a la memoria
14         0, // tamaño maximo parte alta de un DWORD
15         TAM_MEM, // tamaño maximo parte baja de un DWORD
16         idMemCompartida) // identificador de la memoria compartida
17     ) == NULL)
18     {
19         printf("No se mapeo la memoria compartida: (%i)\n", GetLastError());
20         exit(-1);
21     }
22     if ((apDatos = (int *)MapViewOfFile(hArchMapeo, // Manejador del mapeo
23         FILE_MAP_ALL_ACCESS, // Permiso de lectura/escritura en la memoria
24         0,
25         0,
26         TAM_MEM)) == NULL)
27     {
28         printf("No se creo la memoria compartida: (%i)\n", GetLastError());
29         CloseHandle(hArchMapeo);
30         exit(-1);
31     }
32
33     int **matriz1;
34     int **matriz2;
35     int **matriz3;
36     int **matriz4;
37     apTrabajo = apDatos;
38     int fin_intervalo = 10;
39
40     for (c = 0; c < 200; c++)
41     {
42         *apTrabajo++ = rand() % fin_intervalo;
43     }
44     *apDatos = 0;
45
46     ////////////////////////////////// SEMAFORO //////////////////////////////////
47     HANDLE hSemaforo;
48     // Creación del semáforo
49     if ((hSemaforo = CreateSemaphore(NULL, 1, 1, "Semaforo")) == NULL)
50     {
51         printf("Falla al invocar CreateSemaphore: %d\n", GetLastError());
52         return -1;
53     }
54
55     //////////////////////////////////
56     STARTUPINFO si; /* Estructura de información inicial para Windows */
57     PROCESS_INFORMATION pi; /* Estructura de información del adm. de procesos */
58     ZeroMemory(&si, sizeof(si));
59     si.cb = sizeof(si);
60     ZeroMemory(&pi, sizeof(pi));
61     // Creación proceso hijo
```



```

62     if (!CreateProcess(NULL, "C:\\Users\\rodri\\Desktop\\Escuela\\Sistemas\\Practica6\\ProcesoH.exe",
63         NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
64     {
65         printf("Fallo al invocar CreateProcess (%d)\n", GetLastError());
66         return 0;
67     }
68     // Prueba del semáforo
69     WaitForSingleObject(hSemaforo, INFINITE);
70
71     //Sección crítica
72     printf("Soy el padre entrando al semaforo\n");
73     Sleep(5000);
74     //Liberación el semáforo
75     if (!ReleaseSemaphore(hSemaforo, 1, NULL))
76     {
77         printf("Falla al invocar ReleaseSemaphore: %d\n", GetLastError());
78     }
79     printf("Soy el padre liberando al semaforo\n");
80     Sleep(5000);
81     // Terminación controlada del proceso e hilo asociado de ejecución
82     CloseHandle(pi.hProcess);
83     CloseHandle(pi.hThread);
84     ///////////////////////////////////
85     AsignarMatrizN(&matriz1, 10, apDatos, 0);
86     AsignarMatrizN(&matriz2, 10, apDatos, 100);
87     AsignarMatrizN(&matriz3, 10, apDatos, 200);
88     AsignarMatrizN(&matriz4, 10, apDatos, 300);
89     printf("Matriz 1\n");
90     ImprimirMatriz(matriz1, 10);
91     printf("Matriz 2\n");
92     ImprimirMatriz(matriz2, 10);
93     printf("Matriz Multiplicacion\n");
94     ImprimirMatriz(matriz3, 10);
95     printf("Matriz Suma\n");
96     ImprimirMatriz(matriz4, 10);
97     //Convertimos matriz de int a float
98     float MatrizOut1[10][10];
99     float MatrizOut2[10][10];
100    float inversa1[10][10];
101    float inversa2[10][10];
102    conversionMat(matriz1, MatrizOut1);
103    conversionMat(matriz2, MatrizOut2);
104    conversionMat(matriz3, inversa1);
105    conversionMat(matriz4, inversa2);
106    printf("Soy el awuelo\n");
107    ImprimirArchivoIM("PrimerMatriz.txt", MatrizOut1);
108    ImprimirArchivoIM("SegundaMatriz.txt", MatrizOut2);
109    ImprimirArchivoM("Inversa1Matriz.txt", inversa1);
110    ImprimirArchivoM("Inversa2Matriz.txt", inversa2);
111
112    UnmapViewOfFile(apDatos);
113    CloseHandle(hArchMapeo);
114    exit(0);
115 }
116

```

## Archivo: ProcesoH.c

```
1  #include <windows.h> /* Cliente de la memoria compartida */
2  #include <stdio.h>
3  #include "Matrices.h"
4  #define TAM_MEM 400 /*Tamaño de la memoria compartida en bytes */
5  int main(void)
6  {
7      ////////////////////////////////////////////////// SEMAFORO ///////////////////////////////////
8      HANDLE hSemaforo;
9      // Creación del semáforo
10     if ((hSemaforo = CreateSemaphore(NULL, 1, 1, "Semaforo")) == NULL)
11     {
12         printf("Falla al invocar CreateSemaphore: %d\n", GetLastError());
13         return -1;
14     }
15     // Prueba del semáforo
16     WaitForSingleObject(hSemaforo, INFINITE);
17
18     //Sección crítica
19     printf("Soy el hijo entrando al semaforo\n");
20     ///////////////////////////////////
21     HANDLE hArchMapeo;
22     char *idMemCompartida = "MemoriaCompartida";
23     int *apDatos, *apTrabajo, c;
24     if ((hArchMapeo = OpenFileMapping(
25         FILE_MAP_ALL_ACCESS, // acceso lectura/escritura de la memoria compartida
26         FALSE, // no se hereda el nombre
27         idMemCompartida) // identificador de la memoria compartida
28     ) == NULL)
29     {
30         printf("No se abrio archivo de mapeo de la memoria compartida: (%i)\n", GetLastError());
31         exit(-1);
32     }
33     if ((apDatos = (int *)MapViewOfFile(hArchMapeo, // Manejador del mapeo
34         FILE_MAP_ALL_ACCESS, // Permiso de lectura/escritura en la memoria
35         0,
36         0,
37         TAM_MEM)) == NULL)
38     {
39         printf("No se accedio a la memoria compartida: (%i)\n", GetLastError());
40         CloseHandle(hArchMapeo);
41         exit(-1);
42     }
43
44     int **matriz1;
45     int **matriz2;
46     int **Result;
47     apTrabajo = apDatos;
48     AsignarMatrizN(&matriz1, 10, apDatos, 0);
49     AsignarMatrizN(&matriz2, 10, apDatos, 100);
50     MultiplicacionM(matriz1, matriz2, 10, &Result);
51     int k = 200;
52     for (int i = 0; i < 10; i++)
53     {
54         for (int j = 0; j < 10; j++)
55         {
56             apTrabajo[k++] = Result[i][j];
57         }
58     }
59     ////////////////////////////////////////////////// SEMAFORO ///////////////////////////////////
60     HANDLE hSemaforo2;
61     // Creación del semáforo
62     if ((hSemaforo2 = CreateSemaphore(NULL, 1, 1, "Semaforo2")) == NULL)
63     {
64         printf("Falla al invocar CreateSemaphore: %d\n", GetLastError());
65         return -1;
66     }
67     ///////////////////////////////////
68     STARTUPINFO si; /* Estructura de información inicial para Windows */
69     PROCESS_INFORMATION pi; /* Estructura de información del adm. de procesos */
70     ZeroMemory(&si, sizeof(si));
71     si.cb = sizeof(si);
72     ZeroMemory(&pi, sizeof(pi));
73     // Creación proceso hijo
74     if (!CreateProcess(NULL, "C:\\Users\\rodri\\Desktop\\Escuela\\Sistemas\\Practica6\\ProcesoN.exe", NULL, NULL,
75         0, 0, NULL, NULL, &si, &pi))
76     {
77         printf("Fallo al invocar CreateProcess (%d)\n", GetLastError());
78         return 0;
79     }
```

```

79 // Prueba del semáforo
80 WaitForSingleObject(hSemaforo2, INFINITE);
81
82 //Sección crítica
83 printf("Soy el padre entrando al semaforo\n");
84 {
85 //Liberación el semáforo
86 if (!ReleaseSemaphore(hSemaforo2, 1, NULL))
87 {
88     printf("Falla al invocar ReleaseSemaphore: %d\n", GetLastError());
89 }
90 printf("Soy el padre liberando al semaforo\n");
91
92 // Terminación controlada del proceso e hilo asociado de ejecución
93 CloseHandle(pi.hProcess);
94 CloseHandle(pi.hThread);
95 ///////////////////////////////////////////////////
96 UnmapViewOfFile(apDatos);
97 CloseHandle(hArchMapeo);
98
99 //Liberación el semáforo
100 if (!ReleaseSemaphore(hSemaforo, 1, NULL))
101 {
102     printf("Falla al invocar ReleaseSemaphore: %d\n", GetLastError());
103 }
104 printf("Soy el hijo liberando al semaforo\n");
105 exit(0);
106 }
107

```

## Archivo: ProcesoN.c

```

1
2 #include <windows.h> /* Cliente de la memoria compartida */
3 #include <stdio.h>
4 #include "Matrices.h"
5 #define TAM_MEM 400 /*Tamaño de la memoria compartida en bytes */
6 int main(void)
7 {
8     /////////////////////////////////////////////////// SEMAFORO ///////////////////////////////////
9     HANDLE hSemaforo2;
10    // Creación del semáforo
11    if ((hSemaforo2 = CreateSemaphore(NULL, 1, 1, "Semaforo2")) == NULL)
12    {
13        printf("Falla al invocar CreateSemaphore: %d\n", GetLastError());
14        return -1;
15    }
16    // Prueba del semáforo
17    WaitForSingleObject(hSemaforo2, INFINITE);
18    //Sección crítica
19    printf("Soy el hijo entrando al semaforo\n");
20    ///////////////////////////////////////////////////
21    HANDLE hArchMapeo;
22    char *idMemCompartida = "MemoriaCompatida";
23    int *apDatos, *apTrabajo, c;
24    if ((hArchMapeo = OpenFileMapping(
25        FILE_MAP_ALL_ACCESS, // acceso lectura/escritura de la memoria compartida
26        FALSE, // no se hereda el nombre
27        idMemCompartida) // identificador de la memoria compartida
28    ) == NULL)
29    {
30        printf("No se abrio archivo de mapeo de la memoria compartida: (%i)\n", GetLastError());
31        exit(-1);
32    }
33

```

```

32     }
33     if ((apDatos = (int *)MapViewOfFile(hArchMapeo,           // Manejador del mapeo
34                                         FILE_MAP_ALL_ACCESS, // Permiso de lectura/escritura en la memoria
35                                         0,
36                                         0,
37                                         TAM_MEM)) == NULL)
38     {
39         printf("No se accedio a la memoria compartida: (%i)\n", GetLastError());
40         CloseHandle(hArchMapeo);
41         exit(-1);
42     }
43
44     int **matriz1;
45     int **matriz2;
46     int **Result;
47     apTrabajo = apDatos;
48     AsignarMatrizN(&matriz1, 10, apDatos, 0);
49     AsignarMatrizN(&matriz2, 10, apDatos, 100);
50     SumaM(matriz1, matriz2, 10, &Result);
51     int k = 300;
52     for (int i = 0; i < 10; i++)
53     {
54         for (int j = 0; j < 10; j++)
55         {
56             apTrabajo[k++] = Result[i][j];
57         }
58     }
59
60     UnmapViewOfFile(apDatos);
61     CloseHandle(hArchMapeo);
62
63     //Liberación el semáforo
64     if (!ReleaseSemaphore(hSemaforo2, 1, NULL))
65     {
66         printf("Falla al invocar ReleaseSemaphore: %d\n", GetLastError());
67     }
68     printf("Soy el hijo liberando al semaforo\n");
69
70     exit(0);
71 }

```

## Archivo: Matrices.c

```

1  #include <windows.h>
2  #include "Matrices.h"
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <unistd.h>
6  #include <math.h>
7
8  void SumaM(int **m1, int **m2, int size_m, int ***MatrizR)
9  {
10     (*MatrizR) = (int **)calloc(size_m, sizeof(int *));
11     for (int i = 0; i < size_m; i++)
12     {
13         (*MatrizR)[i] = (int *)calloc(size_m, sizeof(int));
14     }
15     for (int i = 0; i < size_m; i++)
16     {
17         for (int j = 0; j < size_m; j++)
18         {
19             (*MatrizR)[i][j] = m1[i][j] + m2[i][j];
20         }
21     }
22 }
23 void RestaM(int **m1, int **m2, int size_m, int ***MatrizR)
24 {
25     (*MatrizR) = (int **)calloc(size_m, sizeof(int *));
26     for (int i = 0; i < size_m; i++)
27     {
28         (*MatrizR)[i] = (int *)calloc(size_m, sizeof(int));
29     }
30     for (int i = 0; i < size_m; i++)
31     {
32         for (int j = 0; j < size_m; j++)
33         {
34             (*MatrizR)[i][j] = m1[i][j] - m2[i][j];

```

```

35     }
36 }
37 }
38 void MultiplicacionM(int **m1, int **m2, int size_m, int ***MatrizR)
39 {
40     (*MatrizR) = (int **)calloc(size_m, sizeof(int *));
41     for (int i = 0; i < size_m; i++)
42     {
43         (*MatrizR)[i] = (int *)calloc(size_m, sizeof(int));
44     }
45     for (int i = 0; i < size_m; i++)
46     {
47         for (int j = 0; j < size_m; j++)
48         {
49             int suma = 0;
50             for (int k = 0; k < size_m; k++)
51             {
52                 suma += m1[j][k] * m2[k][i];
53             }
54             (*MatrizR)[j][i] = suma;
55         }
56     }
57 }
58 }
59 void TranspuestaM(int **m1, int size_m, int ***MatrizR)
60 {
61     (*MatrizR) = (int **)calloc(size_m, sizeof(int *));
62     for (int i = 0; i < size_m; i++)
63     {
64         (*MatrizR)[i] = (int *)calloc(size_m, sizeof(int));
65     }
66     for (int i = 0; i < size_m; i++)
67     {
68         for (int j = 0; j < size_m; j++)

```

```

69         {
70             (*MatrizR)[i][j] = m1[j][i];
71         }
72     }
73 }
74 void ImprimirMatriz(int **m1, int size_m)
75 {
76     printf("La matriz es : \n");
77     for (int i = 0; i < size_m; i++)
78     {
79         for (int j = 0; j < size_m; j++)
80         {
81             printf("%d\t", m1[i][j]);
82         }
83         printf("\n");
84     }
85 }
86 void AsignarMatriz(int ***m1, int size_m)
87 {
88     (*m1) = (int **)calloc(size_m, sizeof(int *));
89     for (int i = 0; i < size_m; i++)
90     {
91         (*m1)[i] = (int *)calloc(size_m, sizeof(int));
92     }
93     for (int i = 0; i < size_m; i++)
94     {
95         for (int j = 0; j < size_m; j++)
96         {
97             (*m1)[i][j] = j;
98         }
99     }
100 }
101 }
102

```

```

103 void AsignarMatrizN(int ***m1, int size_m, int *mN, int start)
104 {
105     int k = start;
106     (*m1) = (int **)calloc(size_m, sizeof(int *));
107     for (int i = 0; i < size_m; i++)
108     {
109         (*m1)[i] = (int *)calloc(size_m, sizeof(int));
110     }
111
112     for (int i = 0; i < size_m; i++)
113     {
114         for (int j = 0; j < size_m; j++)
115         {
116             (*m1)[i][j] = mN[k];
117             k++;
118         }
119     }
120 }
121
122 float determinant(float a[10][10], float k)
123 {
124     float s = 1, det = 0, b[10][10];
125     int i, j, m, n, c;
126     if (k == 1)
127     {
128         return (a[0][0]);
129     }
130     else
131     {
132         det = 0;
133         for (c = 0; c < k; c++)
134         {
135             m = 0;
136             n = 0;
137             for (i = 0; i < k; i++)
138             {
139                 for (j = 0; j < k; j++)
140                 {
141                     b[i][j] = 0;
142                     if (i != 0 && j != c)
143                     {
144                         b[m][n] = a[i][j];
145                         if (n < (k - 2))
146                             n++;
147                         else
148                         {
149                             n = 0;
150                             m++;
151                         }
152                     }
153                 }
154             }
155             det = det + s * (a[0][c] * determinant(b, k - 1));
156             s = -1 * s;
157         }
158     }
159     return (det);
160 }
161
162 void transpose(float num[10][10], float fac[10][10], float r, FILE *archivo)
163 {
164     int i, j;
165     r = 10;
166
167     float b[10][10], inverse[10][10], d;
168
169     for (i = 0; i < r; i++)
170     {
171         for (j = 0; j < r; j++)
172         {
173             b[i][j] = fac[j][i];
174         }
175     }
176
177     d = determinant(num, r);
178     for (i = 0; i < r; i++)
179     {
180         for (j = 0; j < r; j++)
181         {
182             inverse[i][j] = b[i][j] / d;
183         }
184     }
185
186     fprintf(archivo, "The inverse of matrix is : \n");
187
188     for (i = 0; i < r; i++)
189     {
190         for (j = 0; j < r; j++)
191         {
192             fprintf(archivo, "%f\t", inverse[i][j]);
193         }
194         fprintf(archivo, "\n");
195     }
196 }

```

```

197
198 void cofactor(float num[10][10], float f, FILE *archivo)
199 {
200     float b[10][10], fac[10][10];
201     int p, q, m, n, i, j;
202     for (q = 0; q < f; q++)
203     {
204         for (p = 0; p < f; p++)
205         {
206             m = 0;
207             n = 0;
208             for (i = 0; i < f; i++)
209             {
210                 for (j = 0; j < f; j++)
211                 {
212                     if (i != q && j != p)
213                     {
214                         b[m][n] = num[i][j];
215                         if (n < (f - 2))
216                             n++;
217                         else
218                         {
219                             n = 0;
220                             m++;
221                         }
222                     }
223                 }
224             }
225             fac[q][p] = pow(-1, q + p) * determinant(b, f - 1);
226         }
227     }
228     transpose(num, fac, f, archivo);
229 }
230

```

```

231 void conversionMat(int **origi, float transf[][10])
232 {
233
234     for (int i = 0; i < 10; i++)
235     {
236
237         for (int j = 0; j < 10; j++)
238         {
239
240             transf[i][j] = (float)origi[i][j];
241         }
242     }
243 }
244
245 void ImprimirArchivoIM(char *path, float inversa1[10][10])
246 {
247     FILE *archivo = fopen(path, "w");
248     for (int i = 0; i < 10; i++)
249     {
250         for (int j = 0; j < 10; j++)
251         {
252             fprintf(archivo, "%.2f\t", inversa1[i][j]);
253         }
254         fprintf(archivo, "\n");
255     }
256     fclose(archivo);
257 }
258
259 void ImprimirArchivoM(char *path, float inversa1[10][10])
260 {
261     FILE *archivo = fopen(path, "w");
262     float d1 = determinant(inversa1, 10);
263
264     if (d1 != 0)
265         cofactor(inversa1, 10, archivo);
266     else
267         fprintf(archivo, "La matriz no tiene inversa\n");
268
269     fclose(archivo);
270 }
271
272

```

## Compilación y ejecución del programa

### Ejecución código principal

```

TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE

PS C:\Users\rodri\Desktop\Escuela\Sistemas\Practica6> ./Servidor
Soy el padre entrando al semaforo
Soy el padre liberando al semaforo
Soy el hijo entrando al semaforo
Soy el padre entrando al semaforo
Soy el padre liberando al semaforo
Soy el hijo liberando al semaforo
Soy el hijo entrando al semaforo
Soy el hijo liberando al semaforo
Matriz 1
La matriz es :
0      7      4      0      9      4      8      8      2      4
5      5      1      7      1      1      5      2      7      6
1      4      2      3      2      2      1      6      8      5
7      6      1      8      9      2      7      9      5      4
3      1      2      3      3      4      1      1      3      8
7      4      2      7      7      9      3      1      9      8
6      5      0      2      8      6      0      2      4      8
6      5      0      9      0      0      6      1      3      8
9      3      4      4      6      0      6      6      1      8
4      9      6      3      7      8      8      2      9      1
Matriz 2
La matriz es :
3      5      9      8      4      0      7      6      3      6
1      5      4      2      0      9      7      3      7      2
6      0      1      6      5      7      5      4      1      2
0      0      1      4      6      0      7      1      7      7
7      7      3      3      5      9      9      8      1      8
2      6      6      0      3      8      0      1      2      5
0      9      4      7      8      3      5      1      2      0
1      6      4      0      6      1      8      9      8      4
1      4      3      9      8      8      0      8      7      7
8      3      8      3      7      1      0      7      3      4
Matriz Multiplicacion
La matriz es :
144      262      185      151      233      256      254      237      176      176
92       166      179      203      225      148      174      186      196      185
91       143      140      140      191      162      137      201      181      165
146      289      246      239      310      232      340      302      266      281
119      116      150      117      161      118      99       150      105      146
178      251      270      251      300      275      222      274      226      303
161      199      220      150      194      207      179      229      155      222
91       151      184      187      212      96       178      155      181      166
167      220      234      211      258      149      264      255      177      207
141      285      225      259      278      349      261      247      223      246

```





Matriz Suma  
La matriz es :

3	12	13	8	13	4	15	14	5	10
6	10	5	9	1	10	12	5	14	8
7	4	3	9	7	9	6	10	9	7
7	6	2	12	15	2	14	10	12	11
10	8	5	6	8	13	10	9	4	16
9	10	8	7	10	17	3	2	11	13
6	14	4	9	16	9	5	3	6	8
7	11	4	9	6	1	14	10	11	12
10	7	7	13	14	8	6	14	8	15
12	12	14	6	14	9	8	9	12	5

Soy el abuelo

## Archivos creados:

 Inversa1Matriz.txt	19/12/2020 09:32 p. m.	Documento de te...	1 KB
 Inversa2Matriz.txt	19/12/2020 09:32 p. m.	Documento de te...	1 KB

## Inversa de la multiplicación

Inversa1Matriz.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

The inverse of matrix is :

-0.055041	0.336631	-0.196814	0.161603	0.409682	-0.316162	0.085429	-0.197373	-0.144203	0.057333
0.013822	0.305336	-0.157667	0.074240	0.182662	-0.145997	0.058943	-0.166135	-0.087956	-0.014214
0.040784	-0.333910	0.183710	-0.148250	-0.348355	0.261634	-0.062832	0.207284	0.126747	-0.037275
-0.022362	0.174772	-0.089238	0.055072	0.141082	-0.119625	0.037022	-0.102645	-0.045755	0.019459
0.030866	-0.212195	0.111639	-0.079573	-0.183686	0.170653	-0.071010	0.119956	0.073092	-0.026009
0.002517	-0.153597	0.080202	-0.046508	-0.108834	0.085425	-0.029824	0.084679	0.044530	0.004713
-0.012453	-0.127034	0.055222	-0.014739	-0.054984	0.040668	-0.021276	0.067191	0.030856	0.014682
0.017608	-0.075826	0.062890	-0.054366	-0.132239	0.093148	-0.020283	0.034787	0.055377	-0.023692
-0.028446	0.187458	-0.100078	0.087793	0.215822	-0.181307	0.056799	-0.095955	-0.089537	0.030291
-0.015362	0.073392	-0.049560	0.045523	0.081097	-0.049360	0.015248	-0.050849	-0.036152	0.004738

## Inversa de la suma

Inversa2Matriz.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

The inverse of matrix is :

-0.113421	0.100867	-0.151323	0.015400	0.098063	-0.139564	0.013790	-0.067268	0.108936	0.105062
0.004202	-0.055256	0.075762	-0.095596	-0.024147	0.005042	0.075393	0.124643	-0.053282	-0.011515
0.025669	0.112104	-0.179351	0.014717	-0.009488	-0.018679	-0.049762	-0.109483	0.122347	0.042273
-0.042917	0.264097	-0.231573	0.028317	0.009535	-0.165890	0.051243	-0.206536	0.245126	0.004304
0.019568	-0.087298	0.048632	0.052706	-0.012042	0.046797	0.013064	0.001312	-0.063679	0.000348
0.023534	0.000052	0.096256	-0.010309	0.027575	0.018639	0.012700	-0.045886	-0.061633	-0.021226
0.005829	0.094572	-0.086278	0.075303	0.076609	-0.074995	-0.004503	-0.096893	-0.012508	0.019260
0.045363	-0.159488	0.252736	-0.081419	-0.036707	0.047690	-0.011155	0.141080	-0.103430	-0.027237
0.016199	-0.123209	0.151762	0.004938	-0.094768	0.143969	-0.051447	0.133715	-0.116826	-0.017779
0.014745	-0.064470	-0.029121	0.002477	-0.007251	0.098264	-0.044881	0.075784	0.005520	-0.049931

### 3 Conclusiones

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

De igual manera se pudo observar cómo los semáforos es una forma de sincronizar los procesos, esto como un semáforo de calle o la vida real, pues cuando el recurso está ocupado el semáforo espera que se termine de ocupar y lugar asigna al siguiente proceso para que lo utilice