

INSTITUTO POLITÉCNICO NACIONAL

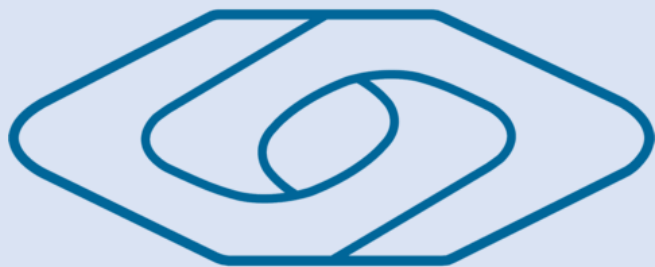
ESCUELA SUPERIOR DE COMPUTO



“Practica 5”

Sistemas Operativos

INSTITUTO POLITÉCNICO NACIONAL



ESCOM

Equipo 7

- Héctor Chávez Rodríguez
 - María Fernanda Delgado Mendoza
 - Jonathan Said Gómez Marbán
 - Luis Antonio Ramírez Fárias
-
- **Grupo:** 4CM1

I. Introducción teórica:

Un proceso independiente no se ve afectado por la ejecución de otros procesos, mientras que un proceso cooperativo puede verse afectado por otros procesos en ejecución. Aunque uno puede pensar que esos procesos, que se ejecutan de forma independiente, se ejecutarán de manera muy eficiente, en realidad, hay muchas situaciones en las que la naturaleza cooperativa se puede utilizar para aumentar la velocidad, la conveniencia y la modularidad computacionales. La comunicación entre procesos (IPC) es un mecanismo que permite a los procesos comunicarse entre sí y sincronizar sus acciones. La comunicación entre estos procesos puede verse como un método de cooperación entre ellos. Los procesos pueden comunicarse entre sí a través de ambos:

1. Memoria compartida
2. Paso de mensajes

Un sistema operativo puede implementar ambos métodos de comunicación.

Primero, los métodos de comunicación de memoria compartida: la comunicación entre procesos que usan memoria compartida requiere que los procesos compartan alguna variable o buffers, y depende completamente de cómo el programador la implemente.

Por otra parte, la comunicación entre procesos a través del paso de mensajes. En este método, los procesos se comunican entre sí sin utilizar ningún tipo de memoria compartida. Si dos procesos p1 y p2 quieren comunicarse entre sí, proceden de la siguiente manera:

- Establecer un enlace de comunicación (si ya existe un enlace, no es necesario volver a establecerlo).

Empezar a intercambiar mensajes utilizando primitivas básicas.

Se requieren al menos dos primitivas:

- enviar (mensaje, destino) o enviar (mensaje)
- recibir (mensaje, host) o recibir (mensaje)

El tamaño del mensaje puede ser de tamaño fijo o variable. Si es de tamaño fijo, es fácil para un diseñador de SO pero complicado para un programador y si es de tamaño variable, entonces es fácil para un programador pero complicado para el diseñador de SO.

La comunicación se establece siguiendo una serie de reglas (protocolos de comunicación). Los protocolos desarrollados para internet son los mayormente usados: IP (capa de red), protocolo de control de transmisión (capa de transporte) y protocolo de transferencia de archivos, protocolo de transferencia de hipertexto (capa de aplicación).

Los procesos pueden estar ejecutándose en una o más computadoras conectadas a una red. Las técnicas de IPC están divididas dentro de métodos para: paso de mensajes,

sincronización, memoria compartida y llamadas de procedimientos remotos (RPC). El método de IPC usado puede variar dependiendo del ancho de banda y latencia (el tiempo desde el pedido de información y el comienzo del envío de esta) de la comunicación entre procesos, y del tipo de datos que están siendo comunicados.

Administración de memoria: Para obtener una utilización adecuada de la memoria, la asignación de memoria debe asignarse de manera eficiente. Uno de los métodos más simples para asignar memoria es dividir la memoria en varias particiones de tamaño fijo y cada partición contiene exactamente un proceso. Así, el grado de multiprogramación se obtiene por el número de particiones.

Asignación de partición múltiple: en este método, se selecciona un proceso de la cola de entrada y se carga en la partición libre. Cuando el proceso termina, la partición está disponible para otros procesos.

Asignación de partición fija: en este método, el sistema operativo mantiene una tabla que indica qué partes de la memoria están disponibles y cuáles están ocupadas por procesos. Inicialmente, toda la memoria está disponible para los procesos del usuario y se considera un gran bloque de memoria disponible. Esta memoria disponible se conoce como "Agujero". Cuando llega el proceso y necesita memoria, buscamos un agujero que sea lo suficientemente grande para almacenar este proceso. Si el requisito se cumple, asignamos memoria para procesar, de lo contrario, mantenemos el resto disponible para satisfacer solicitudes futuras. Al asignar una memoria, a veces se producen problemas de asignación de almacenamiento dinámico, que se refieren a cómo satisfacer una solicitud de tamaño n de una lista de huecos libres.

II. Desarrollo de la practica

Pipe():

Para usar pipes desde C sólo hay que llamar a la función **pipe** y pedirle los dos [descriptores de archivo](#) que serán los extremos del "tubo" que se habrá construido.

Algo así:

```
int p[2];  
pipe(p);
```

Después de la invocación a **pipe** el arreglo de enteros p contiene en p[0] un descriptor de archivo para *leer*, y en p[1] un descriptor de archivo para *escribir*. Esto quiere decir que los pipes funcionan en una sola dirección.



¿Cómo se usan estos “descriptores de archivo” que nos devolvió la función? Bueno, se pueden usar para leer y escribir bytes mediante las funciones **read**, **write** y **close**, pero usualmente es mucho más conveniente crear un FILE* y de esta manera poder usar funciones más cómodas tales como **fread**, **fwrite**, **fprintf**, etc. Esto se logra mediante la función **fdopen**. Por ejemplo para leer de p[0] podemos simplemente hacer FILE *f=fdopen(p[0], "r+"), y ya disponemos de un FILE* listo para leer los bytes que emergen del pipe. Para escribir (por ejemplo en p[1]), se hace lo mismo pero se debe pasar como segundo parámetro "w" (al igual que en un **fopen** normal).

Shmget():

Se emplea en la creación o acceso a una zona de memoria compartida

Retorno:

- Identificador de la memoria compartida si no ha habido error
- -1 si ha habido error

Utilización:

- Primer argumento: key – es una clave que genera el sistema y que será un elemento esencial para poder acceder a la zona de memoria que crearemos. La clave se obtiene previamente utilizando la función `ftok()`; que produce siempre una clave fija con los mismos argumentos: Para usarla:
`key = ftok(«.», 'S');`
La clave debe ser la misma para todos los procesos que quieran usar esta zona de memoria.
- Segundo argumento: size – indicará el tamaño de la memoria compartida. Se suele utilizar la opción de `sizeof(tipo_variable)` para que se tome el tamaño del tipo de dato que habrá en la memoria compartida.
- Tercer argumento: indicará los derechos para acceder a la memoria, si se crea si no existe y caso de que exista, se da o no un error.
Para crear la zona de memoria usaremos `IPC_CREAT | 0660` (ese número indica ciertos permisos de lectura y escritura).
Cuando se esté utilizando esta llamada al sistema para acceder a una memoria que ya se ha creado previamente, este argumento tomará el valor 0.

```
int shmget (key_t key, int size, int shmflg)
```

Shmat():

Para que un proceso pueda acceder a esa memoria compartida previamente creada, será necesario que alguna variable del proceso «apunte» a esa zona de memoria que no pertenece a su espacio de direccionamiento. Para ello, se utiliza esta llamada al sistema que permite vincular esa zona de memoria al direccionamiento lógico del proceso.

Retorno:

- Puntero a la zona de memoria compartida
- -1 si ha habido error

Utilización:

- Primer argumento: shmid – identificador de la memoria compartida. Lo habremos obtenido con la llamada shmget.
- Segundo argumento shmaddr: normalmente, valdrá 0 o NULL que indicará al sistema operativo que busque esa zona de memoria en una zona de memoria libre, no en una dirección absoluta
- Tercer argumento shmflg: se suele poner también 0 en este campo, que corresponde a los flags.

```
void *shmat (int shmid, char * shmaddr, int shmflg)
```

Programa 2

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#define VALOR 1
int main(void){
    int desc_arch[2];
    char bufer[100];

    if(pipe(desc_arch) != 0)
        exit(1);

    if(fork() == 0){
        while(VALOR){
            read(desc_arch[0], bufer, sizeof(bufer));
            printf("Se recibió: %s\n", bufer);
        }
    }
    while(VALOR){
        fgets(bufer, 255, stdin);
        write(desc_arch[1], bufer, strlen(bufer)+1);
    }
}
```

```
jona@ubuntu:~/Documents$ ./Punto2
Hola, esto es un ejemplo de tuberias en Linux
Se recibió: Hola, esto es un ejemplo de tuberias en Linux
```

```
jona@ubuntu:~/Documents$ ./Punto2
El ejemplo trata sobre tuberias
Se recibió: El ejemplo trata sobre tuberias

se uso pipe, fork, read y write dentro del programa
Se recibió: se uso pipe, fork, read y write dentro del programa

Para poder salir se requiere presionar Ctrl + c
Se recibió: Para poder salir se requiere presionar Ctrl + c

^C
jona@ubuntu:~/Documents$
```

Programa 3

```

programa_hijo.c  programa_padre.c
1  #include "windows.h"
2  #include "stdio.h"
3  #include "string.h"
4  int main (int argc, char *argv[])
5  {
6      char mensaje[]="Tuberias en Windows";
7      DWORD escritos;
8      HANDLE hLecturaPipe, hEscrituraPipe;
9      PROCESS_INFORMATION piHijo;
10     STARTUPINFO siHijo;
11     SECURITY_ATTRIBUTES pipeSeg =
12     {sizeof(SECURITY_ATTRIBUTES), NULL, TRUE};
13     /* Obtención de información para la inicialización del proceso hijo */
14     GetStartupInfo (&siHijo);
15     /* Creación de la tubería sin nombre */
16     CreatePipe (&hLecturaPipe, &hEscrituraPipe, &pipeSeg, 0);
17     /* Escritura en la tubería sin nombre */
18     WriteFile(hEscrituraPipe, mensaje, strlen(mensaje)+1, &escritos, NULL);
19
20     siHijo.hStdInput = hLecturaPipe;
21     siHijo.hStdError = GetStdHandle (STD_ERROR_HANDLE);
22     siHijo.hStdOutput = GetStdHandle (STD_OUTPUT_HANDLE);
23     siHijo.dwFlags = STARTF_USESTDHANDLES;
24     CreateProcess (NULL, argv[1], NULL, NULL,
25     TRUE, /* Hereda el proceso hijo los manejadores de la tubería del padre */
26     0, NULL, NULL, &siHijo, &piHijo);
27     WaitForSingleObject (piHijo.hProcess, INFINITE);
28     printf("Mensaje recibido en el proceso hijo, termina el proceso padre\n");
29     CloseHandle(hLecturaPipe);
30     CloseHandle(hEscrituraPipe);
31     CloseHandle(piHijo.hThread);
32     CloseHandle(piHijo.hProcess);
33     return 0;
34 }
35

```

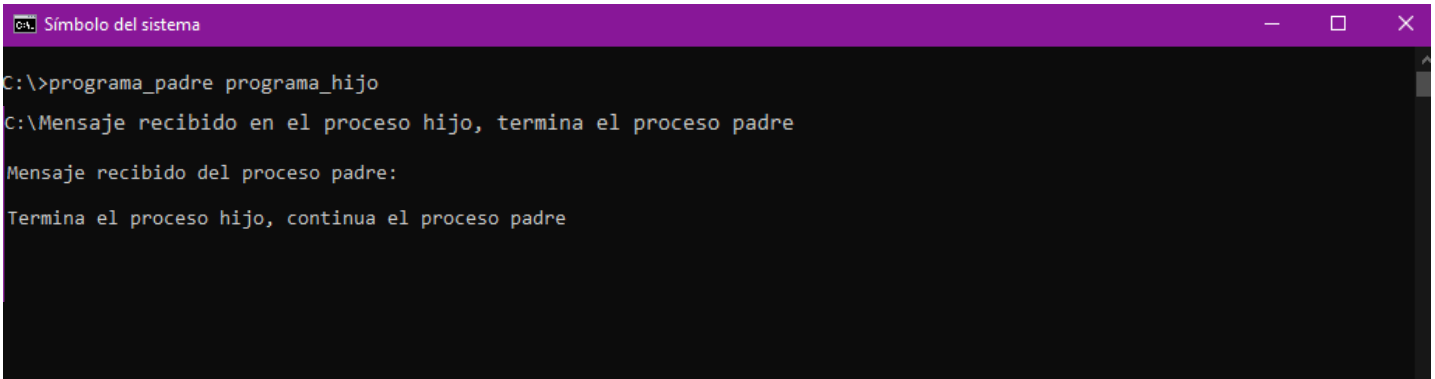
Programa Hijo:

```

programa_hijo.c  programa_padre.c
1  /* Programa hijo.c */
2  #include "windows.h"
3  #include "stdio.h"
4  int main ()
5  {
6      char mensaje[20];
7      DWORD leidos;
8      HANDLE hStdIn = GetStdHandle(STD_INPUT_HANDLE);
9      SECURITY_ATTRIBUTES pipeSeg =
10     {sizeof(SECURITY_ATTRIBUTES), NULL, TRUE};
11     /* Lectura desde la tubería sin nombre */
12     ReadFile(hStdIn, mensaje, sizeof(mensaje), &leidos, NULL);
13     printf("Mensaje recibido del proceso padre: %s\n", mensaje);
14     CloseHandle(hStdIn);
15     printf("Termina el proceso hijo, continua el proceso padre\n");
16     return 0;
17 }

```


Al ejecutarlos:



```
C:\>programa_padre programa_hijo
C:\Mensaje recibido en el proceso hijo, termina el proceso padre
Mensaje recibido del proceso padre:
Termina el proceso hijo, continua el proceso padre
```

Al compilar los programas podemos observar que consisten en una cadena de procesos conectados de forma tal que la salida de cada elemento de la cadena es la entrada del próximo.

Programa 4

LINUX

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

int main(){
    int i,j,k;
    int desc_arch[2] , desc_arch2[2];
    int desc_arch3[2] , desc_arch4[2];
    int desc_arch5[2] , desc_arch6[2];
    int mat1[10][10] , mat2[10][10];
    int mat3[10][10] , mat4[10][10];
    int mul[10][10] , sum[10][10];
    FILE *f,*g;

    //PROCESO HIJO
    if(pipe(desc_arch) != 0)
        exit(1);
    if(pipe(desc_arch2) != 0)
        exit(1);
    if(fork() == 0){

        //MULTIPLICACION
        read(desc_arch[0],mat1,sizeof(mat1));
        read(desc_arch2[0],mat2,sizeof(mat2));
        for(i=0;i<10;i++){
            printf("\n");
            for(j=0;j<10;j++){mul[i][j] = 0;
                for(k=0;k<10;k++){
                    mul[i][j] += (mat1[i][k]*mat2[k][j]);
                }
                printf("%d ",mul[i][j]);}
            printf("\n");
        }

        //ENVIAR MUL AL PADRE
        if(pipe(desc_arch5) != 0)
            exit(1);
        write(desc_arch5[1],mul,sizeof(mul)+1);

        //PROCESO HIJO DEL HIJO
        if(pipe(desc_arch3) != 0)
            exit(1);
        if(pipe(desc_arch4) != 0)
            exit(1);
        if(fork() == 0){

            //SUMA
            read(desc_arch3[0],mat3,sizeof(mat3));
            read(desc_arch4[0],mat4,sizeof(mat4));
            for(i=0;i<10;i++){
                printf("\n");
                for(j=0;j<10;j++){
                    sum[i][j] = mat3[i][j]+mat4[i][j];
                    printf("%d ",sum[i][j]);}
                printf("\n");
            }

            //ENVIAR SUM AL ABUELO
            if(pipe(desc_arch6) != 0)
                exit(1);
            write(desc_arch6[1],sum,sizeof(sum)+1);
        }

        //MATRIZ 3
        for(i=0;i<10;i++){
            printf("\n");
            for(j=0;j<10;j++){
                mat3[i][j] = 2*j;
                printf("%d ",mat3[i][j]);
            }
            write(desc_arch3[1],mat3,sizeof(mat3)+1);printf("\n");
        }

        //MATRIZ 4
        for(i=0;i<10;i++){
            printf("\n");
            for(j=0;j<10;j++){
                mat4[i][j] = 2*j+1;
                printf("%d ",mat4[i][j]);
            }
            write(desc_arch4[1],mat4,sizeof(mat4)+1);printf("\n");
        } //fin del if

        //MATRIZ 1
        for(i=0;i<10;i++){
            printf("\n");
            for(j=0;j<10;j++){
                mat1[i][j] = j+1;
                printf("%d ",mat1[i][j]);
            }
            write(desc_arch[1],mat1,sizeof(mat1)+1);printf("\n");
        }
    }
}
```

```
//MATRIZ 2
for(i=0;i<10;i++){
    printf("\n");
    for(j=0;j<10;j++){
        mat2[i][j] = 10-j;
        printf("%d ",mat2[i][j]);
    }
}
write(desc_arch2[1],mat2,sizeof(mat2)+1);printf("\n");

//TRASPUESTA MULTIPLICACION
f = fopen("PRODUCTO TRASPUESTA.txt","w");
read(desc_arch5[0],mul,sizeof(mul));
for(i=0;i<10;i++){
    printf("\n");fprintf(f,"\n");
    for(j=0;j<10;j++){
        printf("%d ",mul[j][i]);fprintf(f,"%d ",mul[j][i]);
    }
}printf("\n");

//TRASPUESTA SUMA
g = fopen("SUMA TRASPUESTA.txt","w");
read(desc_arch6[0],sum,sizeof(sum));
for(i=0;i<10;i++){
    printf("\n");fprintf(g,"\n");
    for(j=0;j<10;j++){
        printf("%d ",sum[j][i]);fprintf(g,"%d ",sum[j][i]);
    }
}printf("\n");
```

Impresiones de pantalla

```

hector@hector-VirtualBox:~/Documentos$ ./a.out

1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10

10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1

0 0 -624223724 -626317424 0 1323872750 1131932848 0 -624033440 0
0 0 32657 32657 0 717 32767 0 32657 0
0 -624224272 -624225088 -624223488 0 0 -624207957 4 1131932417 0
0 32657 32657 32657 0 0 32657 0 32767 0
1131932288 -624032520 -624037816 1224 0 -624033392 0 1 -624035352 0
32767 32657 32657 0 0 32657 0 0 32657 0
0 1131932288 0 42 0 -8 0 -624032000 0 1132323368
0 32767 0 0 0 -1 0 32657 0 32767
0 -624226304 1 0 -624308768 -624119952 -669270828 -624306880 0 0
0 32657 0 0 32657 32657 -21937 32657 0 0

0 61765110 0 0 0 0 0 0 669270080 669277469
32 1 0 0 0 0 0 0 21936 21936
0 0 0 0 0 0 0 0 15775231 -624316472
0 0 0 0 0 0 0 0 0 32657
0 0 0 0 0 0 0 0 194 669277392
0 0 0 0 0 0 0 0 0 21936
0 0 0 0 0 0 1131937768 0 1131932887 0
0 0 0 0 0 0 32767 0 32767 0
1 0 0 0 0 0 0 0 1131932886 669274528
0 0 0 0 0 0 0 0 32767 21936

```

```
hector@hector-VirtualBox:~/Documentos$  
550 495 440 385 330 275 220 165 110 55  
550 495 440 385 330 275 220 165 110 55  
550 495 440 385 330 275 220 165 110 55  
550 495 440 385 330 275 220 165 110 55  
550 495 440 385 330 275 220 165 110 55  
550 495 440 385 330 275 220 165 110 55  
550 495 440 385 330 275 220 165 110 55  
550 495 440 385 330 275 220 165 110 55  
550 495 440 385 330 275 220 165 110 55  
550 495 440 385 330 275 220 165 110 55  
  
0 2 4 6 8 10 12 14 16 18  
0 2 4 6 8 10 12 14 16 18  
0 2 4 6 8 10 12 14 16 18  
0 2 4 6 8 10 12 14 16 18  
0 2 4 6 8 10 12 14 16 18  
0 2 4 6 8 10 12 14 16 18  
0 2 4 6 8 10 12 14 16 18  
0 2 4 6 8 10 12 14 16 18  
0 2 4 6 8 10 12 14 16 18  
0 2 4 6 8 10 12 14 16 18  
0 2 4 6 8 10 12 14 16 18  
  
1 3 5 7 9 11 13 15 17 19  
1 3 5 7 9 11 13 15 17 19  
1 3 5 7 9 11 13 15 17 19  
1 3 5 7 9 11 13 15 17 19  
1 3 5 7 9 11 13 15 17 19  
1 3 5 7 9 11 13 15 17 19  
1 3 5 7 9 11 13 15 17 19  
1 3 5 7 9 11 13 15 17 19  
1 3 5 7 9 11 13 15 17 19  
1 3 5 7 9 11 13 15 17 19  
1 3 5 7 9 11 13 15 17 19  
  
1 2 3 4 5 6 7 8 9 10  
1 2 3 4 5 6 7 8 9 10  
1 2 3 4 5 6 7 8 9 10  
1 2 3 4 5 6 7 8 9 10  
1 2 3 4 5 6 7 8 9 10  
1 2 3 4 5 6 7 8 9 10  
1 2 3 4 5 6 7 8 9 10  
1 2 3 4 5 6 7 8 9 10  
1 2 3 4 5 6 7 8 9 10  
1 2 3 4 5 6 7 8 9 10  
1 2 3 4 5 6 7 8 9 10
```

```
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1

550 550 550 550 550 550 550 550 550 550
495 495 495 495 495 495 495 495 495 495
440 440 440 440 440 440 440 440 440 440
385 385 385 385 385 385 385 385 385 385
330 330 330 330 330 330 330 330 330 330
275 275 275 275 275 275 275 275 275 275
220 220 220 220 220 220 220 220 220 220
165 165 165 165 165 165 165 165 165 165
110 110 110 110 110 110 110 110 110 110
55 55 55 55 55 55 55 55 55 55

0 61765110 0 0 0 0 0 0 669270080 669277469
32 1 0 0 0 0 0 0 21936 21936
0 0 0 0 0 0 0 0 15775231 -624316472
0 0 0 0 0 0 0 0 0 32657
0 0 0 0 0 0 0 0 194 669277392
0 0 0 0 0 0 0 0 0 21936
0 0 0 0 0 0 1131937768 0 1131932887 0
0 0 0 0 0 0 32767 0 32767 0
1 0 0 0 0 0 0 0 1131932886 669274528
0 0 0 0 0 0 0 0 32767 21936

1 5 9 13 17 21 25 29 33 37
1 5 9 13 17 21 25 29 33 37
1 5 9 13 17 21 25 29 33 37
1 5 9 13 17 21 25 29 33 37
1 5 9 13 17 21 25 29 33 37
1 5 9 13 17 21 25 29 33 37
1 5 9 13 17 21 25 29 33 37
1 5 9 13 17 21 25 29 33 37
1 5 9 13 17 21 25 29 33 37
1 5 9 13 17 21 25 29 33 37
1 5 9 13 17 21 25 29 33 37
```

```
0 2 4 6 8 10 12 14 16 18
0 2 4 6 8 10 12 14 16 18
0 2 4 6 8 10 12 14 16 18
0 2 4 6 8 10 12 14 16 18
0 2 4 6 8 10 12 14 16 18
0 2 4 6 8 10 12 14 16 18
0 2 4 6 8 10 12 14 16 18
0 2 4 6 8 10 12 14 16 18
0 2 4 6 8 10 12 14 16 18
0 2 4 6 8 10 12 14 16 18
```

```
1 3 5 7 9 11 13 15 17 19
1 3 5 7 9 11 13 15 17 19
1 3 5 7 9 11 13 15 17 19
1 3 5 7 9 11 13 15 17 19
1 3 5 7 9 11 13 15 17 19
1 3 5 7 9 11 13 15 17 19
1 3 5 7 9 11 13 15 17 19
1 3 5 7 9 11 13 15 17 19
1 3 5 7 9 11 13 15 17 19
1 3 5 7 9 11 13 15 17 19
1 3 5 7 9 11 13 15 17 19
```

```
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
```

```
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
10 9 8 7 6 5 4 3 2 1
```

```
512 550 550 550 550 550 550 550 550 550
495 495 495 495 495 495 495 495 495 495
440 440 440 440 440 440 440 440 440 440
385 385 385 385 385 385 385 385 385 385
330 330 330 330 330 330 330 330 330 330
275 275 275 275 275 275 275 275 275 275
220 220 220 220 220 220 220 220 220 220
165 165 165 165 165 165 165 165 165 165
110 110 110 110 110 110 110 110 110 110
55 55 55 55 55 55 55 55 55 55

1 1 1 1 1 1 1 1 1 1
5 5 5 5 5 5 5 5 5 5
9 9 9 9 9 9 9 9 9 9
13 13 13 13 13 13 13 13 13 13
17 17 17 17 17 17 17 17 17 17
21 21 21 21 21 21 21 21 21 21
25 25 25 25 25 25 25 25 25 25
29 29 29 29 29 29 29 29 29 29
33 33 33 33 33 33 33 33 33 33
37 37 37 37 37 37 37 37 37 37
hector@hector-VirtualBox:~/Documentos$
```


Windows

Hijo

```
#include <windows.h>
#include <stdio.h>
#include <string.h>
struct matriz{
    int mat1[40][40],mat2[40][40],suma[40][40],multi[40][40];
};
struct matriz *datos;
int sumaMatrices(int arr1[40][40],int arr2[40][40]){
    int i, j;
    for(i=0; i<10; i++) {
        for(j=0; j<10; j++) {
            datos->multi[i][j]=(arr1[i][j]+arr2[i][j]);
        }
    }
    return 0;
}
int main(){
    // datos=malloc(sizeof(int)*40*40*4);
    DWORD leidos,escritos;
    HANDLE hStdIn=GetStdHandle(STD_INPUT_HANDLE),hStdOut=GetStdHandle(STD_OUTPUT_HANDLE);
    ReadFile(hStdIn,datos,40*40*4*4,&leidos,NULL);
    HANDLE hLecturaPipe, hEscrituraPipe;
    PROCESS_INFORMATION piHijo;
    STARTUPINFO siHijo;
    SECURITY_ATTRIBUTES pipeSeg={sizeof(SECURITY_ATTRIBUTES), NULL,TRUE};
    GetStartupInfo(&siHijo);
    CreatePipe(&hLecturaPipe, &hEscrituraPipe, &pipeSeg, 0);
    WriteFile(hEscrituraPipe, datos,40*40*4*4, &escritos, NULL);
    siHijo.hStdInput=hLecturaPipe;
    siHijo.hStdError=GetStdHandle(STD_ERROR_HANDLE);
    siHijo.hStdOutput=hEscrituraPipe;
    siHijo.dwFlags=STARTF_USESTDHANDLES;
    CreateProcess(NULL, "grandson", NULL, NULL, TRUE,0, NULL, NULL, &siHijo, &piHijo);
    WaitForSingleObject(piHijo.hProcess,INFINITE);
    ReadFile(hStdIn,datos,40*40*4*4,&leidos,NULL);
    sumaMatrices(datos->mat1,datos->mat2);
    WriteFile(hStdOut,datos,40*40*4*4,&leidos,NULL);
    CloseHandle(hLecturaPipe);
    CloseHandle(hEscrituraPipe);
    CloseHandle(piHijo.hThread);
    CloseHandle(piHijo.hProcess);
    CloseHandle(hStdIn);
    CloseHandle(hStdOut);
    return 0;
}
```

Nieto

```
#include <windows.h>
#include <stdio.h>
#include <string.h>
struct matriz{
    int mat1[40][40],mat2[40][40],suma[40][40],multi[40][40];
};
struct matriz *datos;
int multiMatrices(int arr1[40][40],int arr2[40][40]){
    int i, j, k;
    for (i = 0; i < 10; i++){
        for (j = 0; j < 10; j++){
            datos->multi[i][j] = 0;
            for (k = 0; k < 10; k++)
                datos->multi[i][j] += arr1[i][k]*arr2[k][j];
        }
    }
    return 0;
}
int main(){
    // datos=malloc(sizeof(int)*40*40*4);
    DWORD leidos,escritos;
    HANDLE hStdIn=GetStdHandle(STD_INPUT_HANDLE),hStdOut=GetStdHandle(STD_OUTPUT_HANDLE);
    ReadFile(hStdIn,datos,40*40*4*4,&leidos,NULL);
    multiMatrices(datos->mat1,datos->mat2);
    WriteFile(hStdOut,datos,40*40*4*4,&leidos,NULL);
    CloseHandle(hStdIn);
    CloseHandle(hStdOut);
    return 0;
}
```

Padre

```

#include <stdio.h>
#include <string.h>
#include <windows.h>
struct matriz{
    int mat1[40][40],mat2[40][40],suma[40][40],multi[40][40];
};
int mat1inv[40][40];
int mat2inv[40][40];
int matr1[40][40]={1,0,0,0,0,0,0,0,0,0},
{1,2,0,0,0,0,0,0,0,0},
{1,2,3,0,0,0,0,0,0,0},
{1,2,3,4,0,0,0,0,0,0},
{1,2,3,4,5,0,0,0,0,0},
{1,2,3,4,5,6,0,0,0,0},
{1,2,3,4,5,6,7,0,0,0},
{1,2,3,4,5,6,7,8,0,0},
{1,2,3,4,5,6,7,8,9,0},
{1,2,3,4,5,6,7,8,9,10}};
int matr2[40][40]={10,9,8,7,6,5,4,3,2,1},
{0,9,8,7,6,5,4,3,2,1},
{0,0,8,7,6,5,4,3,2,1},
{0,0,0,7,6,5,4,3,2,1},
{0,0,0,0,6,5,4,3,2,1},
{0,0,0,0,0,5,4,3,2,1},
{0,0,0,0,0,0,4,3,2,1},
{0,0,0,0,0,0,0,3,2,1},
{0,0,0,0,0,0,0,0,2,1},
{0,0,0,0,0,0,0,0,0,1}};
struct matriz *datos;
void InverseOfMatrix(int matrix[][40], int order,int opc){
    int i, j, k;
    int temp;
    for (i = 0; i < order; i++) {
        for (j = 0; j < 2 * order; j++) {
            if (j == (i + order))
                matrix[i][j] = 1;
        }

        for (i = order - 1; i > 0; i--) {
            if (matrix[i - 1][0] < matrix[i][0])
                for (j = 0; j < 2 * order; j++) {
                    temp = matrix[i][j];
                    matrix[i][j] = matrix[i - 1][j];
                    matrix[i - 1][j] = temp;
                }
        }

        for (i = 0; i < order; i++) {
            for (j = 0; j < 2 * order; j++) {
                if (j != i) {
                    temp = matrix[j][i] / matrix[i][i];
                    for (k = 0; k < 2 * order; k++) {
                        matrix[j][k] -= matrix[i][k] * temp;
                    }
                }
            }
        }

        for (i = 0; i < order; i++) {
            temp = matrix[i][i];
            for (j = 0; j < 2 * order; j++) {
                matrix[i][j] = matrix[i][j] / temp;
            }
        }
    }
}

```

```

        for (i = 0; i < order; i++) {
            for (j = order; j < 2 * order; j++) {
                if(opc==0)
                    mat1inv[i][j-order]=matrix[i][j];
                else
                    mat2inv[i][j-order]=matrix[i][j];
            }
        }

        return;
    }
}

int inversa(){

    InverseOfMatrix(datos->mat1,10,0);
    InverseOfMatrix(datos->mat2,10,1);
    // printf("gg");
    FILE* fichero;
    fichero = fopen("inversa1.txt", "w");
    int i, j;
    for(i=0; i<10; i++) {
        for(j=0; j<10; j++) {
            fprintf(fichero, "%.3f ", (float)(mat1inv[i][j]));
        }
        fprintf(fichero, "\n");
    }
    fprintf(fichero, "\n");
    for(i=0; i<10; i++) {
        for(j=0; j<10; j++) {
            fprintf(fichero, "%.3f ", (float)(mat2inv[i][j]));
        }
        fprintf(fichero, "\n");
    }
    fclose(fichero);
    return 0;
}

int main(int argc, char *argv[]){
    int i, j;
    datos=malloc(sizeof(int)*40*40*4);
    for(i=0;i<10;i++){
        for(j=0;j<10;j++){
            datos->mat1[i][j]=matr1[i][j];
            datos->mat2[i][j]=matr2[i][j];
        }
    }
    DWORD escritos,leidos;
    HANDLE hLecturaPipe, hEscrituraPipe;
    PROCESS_INFORMATION piHijo;
    STARTUPINFO siHijo;
    SECURITY_ATTRIBUTES pipeSeg={sizeof(SECURITY_ATTRIBUTES), NULL,TRUE};
    GetStartupInfo(&siHijo);
    CreatePipe(&hLecturaPipe, &hEscrituraPipe, &pipeSeg, 0);
    printf("DONE");
    WriteFile(hEscrituraPipe,datos,40*40*4*4+1, &escritos, NULL);
    siHijo.hStdInput=hLecturaPipe;
    siHijo.hStdError=GetStdHandle(STD_ERROR_HANDLE);
    siHijo.hStdOutput=hEscrituraPipe;
    siHijo.dwFlags=STARTF_USESTDHANDLES;
    CreateProcess(NULL, "son", NULL, NULL, TRUE, 0, NULL, NULL, &siHijo, &piHijo);
    WaitForSingleObject(piHijo.hProcess,INFINITE);
    printf("Mensaje recibido en el proceso hijo, termina el proceso padre\n");
    ReadFile(hLecturaPipe,datos,40*40*4*4,&leidos,NULL);
    inversa();
    CloseHandle(hLecturaPipe);
    CloseHandle(hEscrituraPipe);
    CloseHandle(piHijo.hThread);
    CloseHandle(piHijo.hProcess);
    return 0;
}

```

Resultado



Programa 5

```

1  #include <sys/types.h> /* Cliente de la memoria compartida */
2  #include <sys/ipc.h>
3  #include <sys/shm.h>
4  #include <stdio.h>
5  #define TAM_MEM 27 /*Tamaño de la memoria compartida en bytes */
6  int main()
7  {
8      int shmid;
9      key_t llave;
10     char *shm, *s;
11     llave = 5678;
12     if ((shmid = shmget(llave, TAM_MEM, 0666)) < 0) {
13         perror("Error al obtener memoria compartida: shmget");
14         exit(-1);
15     }
16     if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
17         perror("Error al enlazar la memoria compartida: shmat");
18         exit(-1);
19     }
20     for (s = shm; *s != '\0'; s++)
21         putchar(*s);
22     putchar('\n');
23     *shm = '*';
24     exit(0);
25 }

```

```

1  #include <sys/types.h> /* Servidor de la memoria compartida */
2  #include <sys/ipc.h> /* (ejecutar el servidor antes de ejecutar el cliente)*/
3  #include <sys/shm.h>
4  #include <stdio.h>
5  #define TAM_MEM 27 /*Tamaño de la memoria compartida en bytes */
6  int main()
7  {
8      char c;
9      int shmid;
10     key_t llave;
11     char *shm, *s;
12     llave = 5678;
13     if ((shmid = shmget(llave, TAM_MEM, IPC_CREAT | 0666)) < 0) {
14         perror("Error al obtener memoria compartida: shmget");
15         exit(-1);
16     }
17     if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
18         perror("Error al enlazar la memoria compartida: shmat");
19         exit(-1);
20     }
21     s = shm;
22     for (c = 'a'; c <= 'z'; c++)
23         *s++ = c;
24     *s = '\0';
25     while (*shm != '*')
26         sleep(1);
27     exit(0);
28 }
29

```

```
david@David: /mnt/d/herna/Documents/tareas/SO/practica5

david@David:/mnt/d/herna/Documents/tareas/SO/practica5$ ./c
Error al obtener memoria compartida: shmget: No such file or directory
david@David:/mnt/d/herna/Documents/tareas/SO/practica5$ ./a
david@David:/mnt/d/herna/Documents/tareas/SO/practica5$

david@David: /mnt/d/herna/Documents/tareas/SO/practica5

david@David:/mnt/d/herna/Documents/tareas/SO/practica5$ ./c
abcdefghijklmnopqrstuvwxyz
david@David:/mnt/d/herna/Documents/tareas/SO/practica5$
```

Programa 6

Punto6.cpp Punto6.2.cpp

```

1  #include <windows.h> /* Cliente de la memoria compartida */
2  #include <stdio.h>
3  #define TAM_MEM 27 /*Tamaño de la memoria compartida en bytes */
4  int main(void)
5  {
6      HANDLE hArchMapeo;
7      char *idMemCompartida = "MemoriaCompatida";
8      char *apDatos, *apTrabajo, c;
9      if((hArchMapeo=OpenFileMapping(
10         FILE_MAP_ALL_ACCESS, // acceso lectura/escritura de la memoria compartida
11         FALSE, // no se hereda el nombre
12         idMemCompartida) // identificador de la memoria compartida
13         ) == NULL)
14      {
15          printf("No se abrio archivo de mapeo de la memoria compartida: (%i)\n", GetLastError());
16          exit(-1);
17      }
18      if((apDatos=(char *)MapViewOfFile(hArchMapeo, // Manejador del mapeo
19         FILE_MAP_ALL_ACCESS, // Permiso de lectura/escritura en la memoria
20         0,
21         0,
22         TAM_MEM)) == NULL)
23      {
24          printf("No se accedio a la memoria compartida: (%i)\n", GetLastError());
25          CloseHandle(hArchMapeo);
26          exit(-1);
27      }
28      for (apTrabajo = apDatos; *apTrabajo != '\0'; apTrabajo++)
29          putchar(*apTrabajo);
30          putchar('\n');
31          *apDatos = '*';
32          UnmapViewOfFile(apDatos);
33          CloseHandle(hArchMapeo);
34          exit(0);
35      }

```

Punto6.cpp Punto6.2.cpp

```

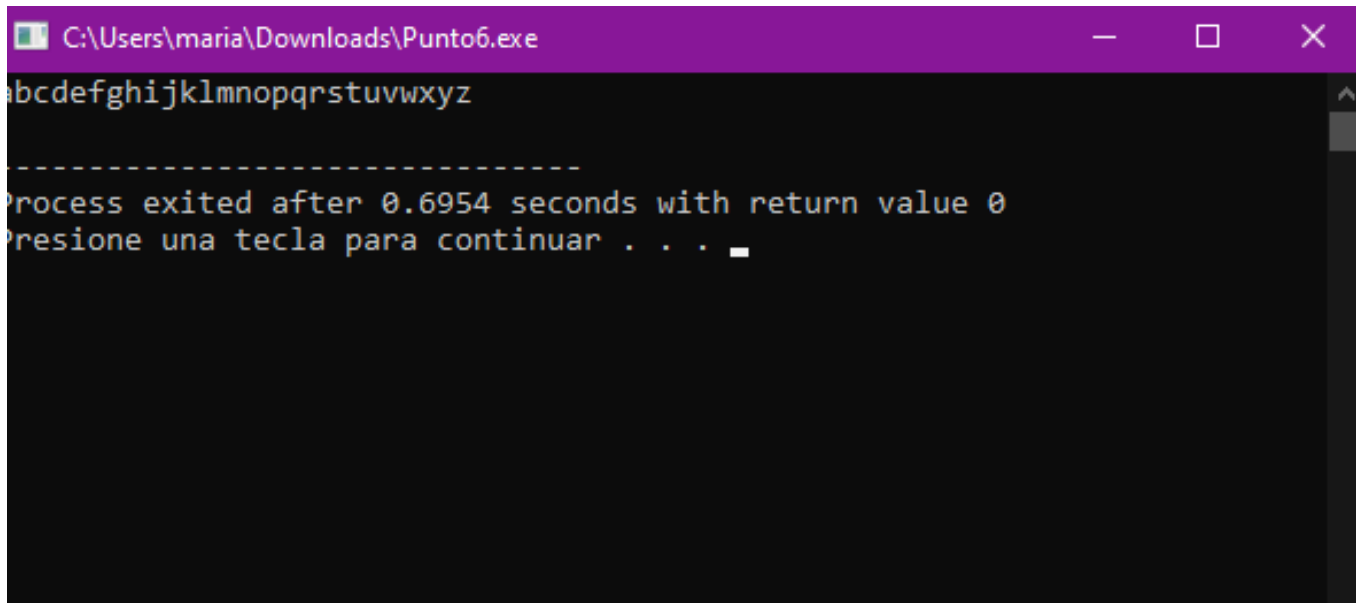
1  #include <windows.h> /* Servidor de la memoria compartida */
2  #include <stdio.h> /* (ejecutar el servidor antes de ejecutar el cliente)*/
3  #define TAM_MEM 27 /*Tamaño de la memoria compartida en bytes */
4  int main(void)
5  {
6      HANDLE hArchMapeo;
7      char *idMemCompartida = "MemoriaCompatida";
8      char *apDatos, *apTrabajo, c;
9      if((hArchMapeo=CreateFileMapping(
10         INVALID_HANDLE_VALUE, // usa memoria compartida
11         NULL, // seguridad por default
12         PAGE_READWRITE, // acceso lectura/escritura a la memoria
13         0, // tamaño maxixmo parte alta de un DWORD
14         TAM_MEM, // tamaño maxixmo parte baja de un DWORD
15         idMemCompartida) // identificador de la memoria compartida
16         ) == NULL)
17      {
18          printf("No se mapeo la memoria compartida: (%i)\n", GetLastError());
19          exit(-1);
20      }
21      if((apDatos=(char *)MapViewOfFile(hArchMapeo, // Manejador del mapeo
22         FILE_MAP_ALL_ACCESS, // Permiso de lectura/escritura en la memoria
23         0,
24         0,
25         TAM_MEM)) == NULL)
26      {

```



```
27     printf("No se creo la memoria compartida: (%i)\n", GetLastError());
28     CloseHandle(hArchMapeo);
29     exit(-1);
30 }
31 apTrabajo = apDatos;
32 for (c = 'a'; c <= 'z'; c++)
33     *apTrabajo++ = c;
34 *apTrabajo = '\0';
35 while (*apDatos != '*')
36     Sleep(1);
37 UnmapViewOfFile(apDatos);
38 CloseHandle(hArchMapeo);
39 exit(0);
40 }
41
```

Se compila primero el programa 6.2 y después el 6 para poder obtener:



```
C:\Users\maria\Downloads\Punto6.exe
abcdefghijklmnopqrstuvwxyz
-----
Process exited after 0.6954 seconds with return value 0
Presione una tecla para continuar . . .
```

Y se compilan de esa manera ya que en el primer programa se crea la memoria compartida y en el segundo se verifica que sí exista esta y si es así se muestra lo que viene del primer programa que es el abecedario.

Programa 7

Padre windows

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "matrices.c"

int imprimir_matriz(float matriz[100]);
void crear_memoria();
void recibir_inversas();

int main(void) {
    srand(time(NULL));
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));
    if (!CreateProcess(NULL, "hijo.exe", NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)) {
        printf("Fallo al invocar CreateProcess (%d)\n", (int)GetLastError());
        return -1;
    }
    crear_memoria();
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
    return 0;
}

void crear_memoria() {
    HANDLE hArchMapeo;
    char *idMemCompartida = "MemoriaPadre";
    size_t TAM_MEM = sizeof(struct Contenedor);
    struct Contenedor *apDatos;
    if ((hArchMapeo=CreateFileMapping(INVALID_HANDLE_VALUE, NULL, PAGE_READWRITE, 0, TAM_MEM, idMemCompartida))==NULL) {
        printf("No se mapeo la memoria compartida(padre): %i\n", GetLastError());
        exit(-1);
    }

    if ((apDatos=(struct Contenedor*)MapViewOfFile(hArchMapeo, FILE_MAP_ALL_ACCESS, 0, 0, TAM_MEM))==NULL) {
        printf("No se creo la memoria compartida(padre): %i\n", GetLastError());
        CloseHandle(hArchMapeo);
        exit(-1);
    }
    apDatos->estado = 0;
    int pos, i, j;
    for (i = 0, pos=0; i<COL; i++)
        for (j=0; j<COL; j++, pos++){
            (apDatos->matrizUno)[pos] = rand()%11;
            (apDatos->matrizDos)[pos] = rand()%11;
        }
    while(!apDatos->estado)
        sleep(1);
    recibir_inversas();
    UnmapViewOfFile(apDatos);
    CloseHandle(hArchMapeo);
}

void recibir_inversas() {
    HANDLE hArchMapeo, hArchMapeo2;
    char *idMemCompartida = "HijoPadre";
    char *idMemCompartida2 = "NietoPadre";

    size_t TAM_MEM = 100*sizeof(int);
    int *apDatos, *apDatos2;
    /*Para la multiplicacion*/
    if ((hArchMapeo=OpenFileMapping(FILE_MAP_ALL_ACCESS, FALSE, idMemCompartida))==NULL) {
        printf("No se abrio el archivo de mapeo de la memoria compartida(hijo_padre): %i\n", GetLastError());
        exit(-1);
    }
}
```

```

    if ((apDatos=(int*)MapViewOfFile(hArchMapeo, FILE_MAP_ALL_ACCESS, 0, 0, TAM_MEM))==NULL) {
        printf("No se accedio a la memoria compartida(hijo_padre): %i\n", GetLastError());
        CloseHandle(hArchMapeo);
        exit(-1);
    }
    /*Otro*/
    if ((hArchMapeo2=OpenFileMapping(FILE_MAP_ALL_ACCESS, FALSE, idMemCompartida2))==NULL) {
        printf("No se abrio el archivo de mapeo de la memoria compartida(nieto_padre): %i\n", GetLastError());
        exit(-1);
    }

    if ((apDatos2=(int*)MapViewOfFile(hArchMapeo2, FILE_MAP_ALL_ACCESS, 0, 0, TAM_MEM))==NULL) {
        printf("No se accedio a la memoria compartida(nieto_padre): %i\n", GetLastError());
        CloseHandle(hArchMapeo);
        exit(-1);
    }

    float **inversa_suma = matriz();
    float **inversa_multiplicacion = matriz();
    int i, j, pos;
    for (i = 0, pos=0; i<COL; i++)
        for (j=0; j<COL; j++, pos++){
            inversa_suma[i][j] = (apDatos2)[pos];
            inversa_multiplicacion[i][j] = (apDatos)[pos];
        }
    printf(" \n");
    gaussj(inversa_suma);
    gaussj(inversa_multiplicacion);
    int ficheroInv = abrir("inversa_suma.txt");
    escribir_texto(ficheroInv, "INV SUMK:");
    escribir_float(ficheroInv, inversa_suma);
    close(ficheroInv);
    ficheroInv = abrir("inversa_multiplicacion.txt");
    escribir_texto(ficheroInv, "INV MULT:");
    escribir_float(ficheroInv, inversa_multiplicacion);
    close(ficheroInv);
    *apDatos = -90;
    *apDatos2 = -90;
    UnmapViewOfFile(apDatos);
    CloseHandle(hArchMapeo);
    UnmapViewOfFile(apDatos2);
    CloseHandle(hArchMapeo2);
    return;
}

int imprimir_matriz(float matriz[100]) {
    int i;
    printf("");
    for (i=0; i<100; i++){
        if (i%10 == 0)
            printf("\n");
        printf("%.0f ", matriz[i]);
    }
    printf("\n\n");
    return 0;
}

```

Hijo

```

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "matrices.c"

void crear_memoria();
int imprimir_matriz(float matriz[100]);
void hijo_padre(int matrizProducto[COL][COL], struct Contenedor*);
void hijo_nieto();

int main(void) {
    srand(time(NULL));
    crear_memoria();
    exit(0);
    return 0;
}

void crear_memoria() {
    HANDLE hArchMapeo;
    char *idMemCompartida = "MemoriaPadre";
    size_t TAM_MEM = sizeof(struct Contenedor);
    struct Contenedor *apDatos;
    if ((hArchMapeo=OpenFileMapping(FILE_MAP_ALL_ACCESS, FALSE, idMemCompartida))==NULL) {
        printf("No se abrio el archivo de mapeo de la memoria compartida(hijo): %i\n", GetLastError());
        exit(-1);
    }

    if ((apDatos=(struct Contenedor*)MapViewOfFile(hArchMapeo, FILE_MAP_ALL_ACCESS, 0, 0, TAM_MEM))==NULL) {
        printf("No se accedio a la memoria compartida(hijo): %i\n", GetLastError());
        CloseHandle(hArchMapeo);
        exit(-1);
    }

    float **matrizA = matriz();
    float **matrizB = matriz();
    int pos, i, j;
    for (i = 0, pos=0; i<COL; i++)
        for (j=0; j<COL; j++, pos++){
            matrizA[i][j] = (apDatos->matrizUno)[pos];
            matrizB[i][j] = (apDatos->matrizDos)[pos];
        }
    int matrizProducto[COL][COL];
    printf("M1:\n");
    imprimir_matriz(apDatos->matrizUno);
    printf("M2:\n");
    imprimir_matriz(apDatos->matrizDos);
    multiplicar(matrizA, matrizB, matrizProducto);
    printf("MULTIPLICACION:\n");
    mostrar_matriz(matrizProducto);
    hijo_padre(matrizProducto, apDatos);
    UnmapViewOfFile(apDatos);
    CloseHandle(hArchMapeo);
    return;
}

void hijo_padre(int matrizProducto[COL][COL], struct Contenedor *anterior) {
    HANDLE hArchMapeo;
    char *idMemCompartida = "HijoPadre";
    size_t TAM_MEM = 100*sizeof(int);
    int *apDatos;
    if ((hArchMapeo=CreateFileMapping(INVALID_HANDLE_VALUE, NULL, PAGE_READWRITE, 0, TAM_MEM, idMemCompartida))==NULL) {
        printf("No se mapeo la memoria compartida(hijo_padre): %i\n", GetLastError());
        exit(-1);
    }

    if ((apDatos=(int*)MapViewOfFile(hArchMapeo, FILE_MAP_ALL_ACCESS, 0, 0, TAM_MEM))==NULL) {
        printf("No se creo la memoria compartida(hijo_padre): %i\n", GetLastError());
        CloseHandle(hArchMapeo);
        exit(-1);
    }
}

```

```

    int i, j, pos;
    for (i = 0, pos=0; i<COL; i++)
        for (j=0; j<COL; j++, pos++)
            apDatos[pos] = matrizProducto[i][j];
    hijo_nieto();
    anterior->estado = 1;
    while(*apDatos != -90)
        sleep(1);
    UnmapViewOfFile(apDatos);
    CloseHandle(hArchMapeo);
    return;
}

void hijo_nieto(){
    /*Crear*/
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));
    if (!CreateProcess(NULL, "nieto.exe", NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)) {
        printf("Fallo al invocar CreateProcess (%d)\n", (int)GetLastError());
        return -1;
    }
    /*Termina*/
    HANDLE hArchMapeo;
    char *idMemCompartida = "HijoNieto";
    size_t TAM_MEM = sizeof(struct Contenedor);
    struct Contenedor *apDatos;
    if ((hArchMapeo>CreateFileMapping(INVALID_HANDLE_VALUE, NULL, PAGE_READWRITE, 0, TAM_MEM, idMemCompartida))==NULL) {
        printf("No se mapeo la memoria compartida(hijo_nieto): %i\n", GetLastError());
        exit(-1);
    }

    if ((apDatos=(struct Contenedor*)MapViewOfFile(hArchMapeo, FILE_MAP_ALL_ACCESS, 0, 0, TAM_MEM))==NULL) {
        printf("No se creo la memoria compartida(hijo_nieto): %i\n", GetLastError());
        CloseHandle(hArchMapeo);
        exit(-1);
    }
    apDatos->estado = 0;
    int pos, i, j;
    for (i = 0, pos=0; i<COL; i++)
        for (j=0; j<COL; j++, pos++){
            (apDatos->matrizUno)[pos] = rand()%11;
            (apDatos->matrizDos)[pos] = rand()%11;
        }
    while(!apDatos->estado)
        sleep(1);

    UnmapViewOfFile(apDatos);
    CloseHandle(hArchMapeo);
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
    return;
}

int imprimir_matriz(float matriz[100]) {
    int i;
    printf("");
    for (i=0; i<100; i++){
        if (i%10 == 0)
            printf("\n");
        printf("%.0f ", matriz[i]);
    }
    printf("\n\n");
    return 0;
}

```

Nieto

```

int main(void) {
    HANDLE hArchMapeo;
    char *idMemCompartida = "HijoNieto";
    size_t TAM_MEM = sizeof(struct Contenedor);
    struct Contenedor *apDatos;
    if ((hArchMapeo=OpenFileMapping(FILE_MAP_ALL_ACCESS, FALSE, idMemCompartida))==NULL) {
        printf("No se abrio el archivo de mapeo de la memoria compartida(nieto): %i\n", GetLastError());
        exit(-1);
    }

    if ((apDatos=(struct Contenedor*)MapViewOfFile(hArchMapeo, FILE_MAP_ALL_ACCESS, 0, 0, TAM_MEM))==NULL) {
        printf("No se accedio a la memoria compartida(nieto): %i\n", GetLastError());
        CloseHandle(hArchMapeo);
        exit(-1);
    }

    float **matrizA = matriz();
    float **matrizB = matriz();
    int pos, i, j;
    for (i = 0, pos=0; i<COL; i++)
        for (j=0; j<COL; j++, pos++){
            matrizA[i][j] = (apDatos->matrizUno)[pos];
            matrizB[i][j] = (apDatos->matrizDos)[pos];
        }
    int matrizSuma[COL][COL];
    printf("M3:\n");
    imprimir_matriz(apDatos->matrizUno);
    printf("M4:\n");
    imprimir_matriz(apDatos->matrizDos);
    sumar(matrizA, matrizB, matrizSuma);
    printf("RESULTADO SUMA:\n");
    mostrar_matriz(matrizSuma);
    nieto_padre(matrizSuma, apDatos);
    UnmapViewOfFile(apDatos);
    CloseHandle(hArchMapeo);
    exit(0);
    return 0;
}

void nieto_padre(int matrizSuma[10][10], struct Contenedor *anterior) {
    HANDLE hArchMapeo;
    char *idMemCompartida = "NietoPadre";
    size_t TAM_MEM = 100*sizeof(int);
    int *apDatos;
    if ((hArchMapeo=CreateFileMapping(INVALID_HANDLE_VALUE, NULL, PAGE_READWRITE, 0, TAM_MEM, idMemCompartida))==NULL) {
        printf("No se mapeo la memoria compartida(nieto_padre): %i\n", GetLastError());
        exit(-1);
    }

    if ((apDatos=(int*)MapViewOfFile(hArchMapeo, FILE_MAP_ALL_ACCESS, 0, 0, TAM_MEM))==NULL) {
        printf("No se creo la memoria compartida(nieto_padre): %i\n", GetLastError());
        CloseHandle(hArchMapeo);
        exit(-1);
    }

    anterior->estado = 1;
    int i, j, pos;
    for (i = 0, pos=0; i<COL; i++)
        for (j=0; j<COL; j++, pos++){
            apDatos[pos] = matrizSuma[i][j];
        }
    while(*apDatos != -90)
        sleep(1);

    UnmapViewOfFile(apDatos);
    CloseHandle(hArchMapeo);
}

```

```

int imprimir_matriz(float matriz[100]) {
    int i;
    printf("");
    for (i=0; i<100; i++){
        if (i%10 == 0)
            printf("\n");
        printf("%.0f ", matriz[i]);
    }
    printf("\n\n");
    return 0;
}

```

Impresión de pantalla

```

7 5 0 2 3 8 4 0 6 1
3 10 9 0 8 8 0 1 1 9
1 1 6 9 6 5 10 4 0 4
7 2 0 5 0 7 0 1 8 6
3 7 0 10 6 9 0 0 1 0
5 10 10 10 3 6 2 5 6 8
8 5 7 3 0 8 4 3 10 3
5 5 6 8 10 4 8 4 3 7
4 8 0 7 8 7 6 3 0 9
2 5 6 9 6 8 8 3 4 10

```

M2:

```

0 6 7 6 0 3 8 6 7 2
8 7 0 5 4 5 8 6 8 1
1 10 0 0 2 9 0 7 8 9
2 2 10 5 9 3 10 9 5 2
7 2 10 10 2 9 3 0 0 0
1 1 10 10 7 7 6 9 6 4
2 7 7 6 0 4 6 0 8 2
4 10 2 4 3 5 4 8 8 3
6 1 5 9 6 4 7 4 5 3
1 1 9 6 2 5 7 1 1 7

```

MULTIPLICACION:

```

118 130 246 271 138 180 246 187 210 88
172 222 269 295 157 322 250 234 243 198
119 222 321 266 164 258 258 216 254 155
91 97 265 259 165 162 266 208 186 123
133 109 276 262 199 195 259 235 186 72
205 310 351 354 265 351 395 376 375 247
144 238 277 312 192 263 309 293 324 190
193 271 397 368 196 335 348 261 305 187
174 198 377 351 187 278 343 231 250 142
176 245 416 378 234 335 370 287 313 220

```

```

7 5 0 2 3 8 4 0 6 1
3 10 9 0 8 8 0 1 1 9
1 1 6 9 6 5 10 4 0 4
7 2 0 5 0 7 0 1 8 6
3 7 0 10 6 9 0 0 1 0
5 10 10 10 3 6 2 5 6 8
8 5 7 3 0 8 4 3 10 3
5 5 6 8 10 4 8 4 3 7
4 8 0 7 8 7 6 3 0 9
2 5 6 9 6 8 8 3 4 10

```

M4:

```

0 6 7 6 0 3 8 6 7 2
8 7 0 5 4 5 8 6 8 1
1 10 0 0 2 9 0 7 8 9
2 2 10 5 9 3 10 9 5 2
7 2 10 10 2 9 3 0 0 0
11 10 10 7 7 6 9 6 4
2 7 7 6 0 4 6 0 8 2
4 10 2 4 3 5 4 8 8 3
6 1 5 9 6 4 7 4 5 3
1 1 9 6 2 5 7 1 1 7

```

RESULTADO SUMA:

```

7 11 7 8 3 11 12 6 13 3
11 17 9 5 12 13 8 7 9 10
2 11 6 9 8 14 10 11 8 13
9 4 10 10 9 10 10 10 13 8
10 9 10 20 8 18 3 0 1 0
6 11 20 20 10 13 8 14 12 12
10 12 14 9 0 12 10 3 18 5
9 15 8 12 13 9 12 12 11 10
10 9 5 16 14 11 13 7 5 12
3 6 15 15 8 13 15 4 5 17

```

Linux

Programa 1.-

```

int mat1[10][10]={1,0,0,0,0,0,0,0,0,0},
                  {1,2,0,0,0,0,0,0,0,0},
                  {1,2,3,0,0,0,0,0,0,0},
                  {1,2,3,4,0,0,0,0,0,0},
                  {1,2,3,4,5,0,0,0,0,0},
                  {1,2,3,4,5,6,0,0,0,0},
                  {1,2,3,4,5,6,7,0,0,0},
                  {1,2,3,4,5,6,7,8,0,0},
                  {1,2,3,4,5,6,7,8,9,0},
                  {1,2,3,4,5,6,7,8,9,10}};

int mat2[10][10]={10,9,8,7,6,5,4,3,2,1},
                  {0,9,8,7,6,5,4,3,2,1},
                  {0,0,8,7,6,5,4,3,2,1},
                  {0,0,0,7,6,5,4,3,2,1},
                  {0,0,0,0,6,5,4,3,2,1},
                  {0,0,0,0,0,5,4,3,2,1},
                  {0,0,0,0,0,0,4,3,2,1},
                  {0,0,0,0,0,0,0,3,2,1},
                  {0,0,0,0,0,0,0,0,2,1},
                  {0,0,0,0,0,0,0,0,0,1}};

float inv1[10][10]={};
float inv2[10][10]={};
float multi[40][40]={};
float suma[40][40]={};

void FuncInv(float matrix[][40], int order,int opc)
{
    int i, j, k;
    float temp;
    for (i = 0; i < order; i++)
    {
        for (j = 0; j < 2 * order; j++)
        {
            if (j == (i + order))
                matrix[i][j] = 1;
        }
    }
}

```



```

for (i = order - 1; i > 0; i--)
{
    if (matrix[i - 1][0] < matrix[i][0])
        for (j = 0; j < 2 * order; j++)
        {
            temp = matrix[i][j];
            matrix[i][j] = matrix[i - 1][j];
            matrix[i - 1][j] = temp;
        }
}

for (i = 0; i < order; i++)
{
    for (j = 0; j < 2 * order; j++)
    {
        if (j != i) {
            temp = matrix[j][i] / matrix[i][i];
            for (k = 0; k < 2 * order; k++)
            {
                matrix[j][k] -= matrix[i][k] * temp;
            }
        }
    }
}

for (i = 0; i < order; i++)
{
    temp = matrix[i][i];
    for (j = 0; j < 2 * order; j++)
    {
        matrix[i][j] = matrix[i][j] / temp;
    }
}

for (i = 0; i < order; i++) {
    for (j = order; j < 2 * order; j++)
    {
        if(opc==0)
            inv1[i][j-order]=matrix[i][j];
        else
            inv2[i][j-order]=matrix[i][j];
    }
}

return;

```

Main

```
if((shmid = shmget(llave, TAM_MEM, IPC_CREAT | 0666)) < 0)
{
    perror("1 ERROR DE MEMORIA COMPARTIDA 1: shmget");
    exit(-1);
}

if((shm = shmat(shmid, NULL, 0)) == (int *)-1)
{
    perror("ERROR DE ENLACE EN MEMORIA 1: shmat");
    exit(-1);
}

llave = 5679;
if((shmid2 = shmget(llave, TAM_MEM, IPC_CREAT | 0666)) < 0)
{
    perror("ERROR DE OBTENCION DE MEMORIA 2: shmget"); exit(-1);
}

if((shm2 = shmat(shmid2, NULL, 0)) == (int*)-1)
{
    perror("ERROR DE ENLACE EN MEMORIA 2: shmat");
    exit(-1);
}

llave = 5681;
if((shmid3 = shmget(llave, TAM_MEM, IPC_CREAT | 0666)) < 0)
{
    perror("ERROR AL OBTENER MEMORIA 3: shmget"); exit(-1);
}

if((shm3 = shmat(shmid3, NULL, 0)) == (int*)-1)
{
    perror("ERROR DE ENLACE EN MEMORIA 3: shmat");
    exit(-1);
}

s = shm;
for(i = 0, k = 0; i < 10; i++)
{
    for(j = 0; j < 10; j++)
    {
        s[k++] = mat1[i][j];
    }
}
```

```
while(*shm != -10)
    sleep(1);

s2 = shm2;
for(i = 0, k = 0; i < 10; i++)
{
    for(j = 0; j < 10; j++)
    {
        multi[i][j] = s2[k++];
    }
}
*shm2=-10;

s3 = shm3;
for(i = 0, k = 0; i < 10; i++)
{
    for(j = 0; j < 10; j++)
    {
        suma[i][j] = s3[k++];
    }
}
*shm3=-10;

FuncInv(multi,10,0);

for(int i=0; i<10; i++)
{
    for(int j=0; j<10; j++)
    {
        printf("%.3f ", (float)(inv1[i][j]));
    }
    printf("\n");
}
printf("\n");

FuncInv(suma,10,1);

for(int i=0; i<10; i++)
{
    for(int j=0; j<10; j++)
    {
        printf("%.3f ", (float)(inv2[i][j]));
    }
    printf("\n");
}
```

Programa 2.-

```

llave = 5678;
if((shmid = shmget(llave,TAM_MEM, 0666)) <0)
{
    perror("ERROR AL OBTENER MEMORIA COMPARTIDA: shmget"); exit(-1);
}

if((shm = shmat(shmid,NULL,0)) == (int*)-1)
{
    perror("ERROR AL ENLAZAR MEMORIA COMPARTIDA: shmat");
    exit(-1);
}

llave = 5679;
if((shmid2 = shmget(llave,TAM_MEM,IPC_CREAT | 0666)) <0)
{
    perror("ERROR AL OBTENER MEMORIA COMPARTIDA: shmget"); exit(-1);
}

if((shm2 = shmat(shmid2,NULL,0)) == (int*)-1)
{
    perror("ERROR AL ENLAZAR MEMORIA COMPARTIDA: shmat");
    exit(-1);
}

llave = 5680;
if((shmid3 = shmget(llave, TAM_MEM, IPC_CREAT | 0666)) <0)
{
    perror("ERROR AL OBTENER MEMORIA COMPARTIDA: shmget"); exit(-1);
}

if((shm3 = shmat(shmid3,NULL,0)) == (int*)-1)
{
    perror("ERROR AL ENLAZAR MEMORIA COMPARTIDA: shmat");
    exit(-1);
}

int mat1h[10][10]={};
int mat2h[10][10]={};
float multih[10][10]={};

s = shm;
for(i = 0, k =0; i < 10; i++)
{
    for(j = 0; j < 10; j++)
    {
        mat1h[i][j] = s[k++];
    }
}

```

```

for(i = 0; i < 10; i++)
{
    for(j = 0; j < 10; j++)
    {
        mat2h[i][j] = s[k++];
    }
}
*shm=-10;

s3 = shm3;
for(i = 0, k = 0; i < 10; i++)
{
    for(j = 0; j < 10; j++)
    {
        s3[k++] = mat1h[i][j];
    }
}
for(i = 0; i < 10; i++)
{
    for(j = 0; j < 10; j++)
    {
        s3[k++] = mat2h[i][j];
    }
}
while(*shm3 != -10)
    sleep(1);
printf("sos\n");
s2 = shm2;
int z=0;
for (i = 0, z = 0; i < 10; i++)
{
    for (j = 0; j < 10; j++)
    {
        s2[z]=0;
        for (k = 0; k < 10; k++)
            s2[z]+=mat1h[i][k]*mat2h[k][j];
        z++;
    }
}
while(*shm2 != -10)
    sleep(1);

printf("se envio\n");

```

Programa 3.-

```

llave = 5680;
if((shmid = shmget(llave,TAM_MEM, IPC_CREAT | 0666)) <0)
{
    perror("ERROR AL OBTENER MEMORIA COMPARTIDA: shmget");
    exit(-1);
}

if((shm = shmat(shmid,NULL,0)) == (int*)-1)
{
    perror("ERROR AL ENLAZAR MEMORIA COMPARTIDA: shmat");
    exit(-1);
}

llave = 5681;
if((shmid3= shmget(llave, TAM_MEM, IPC_CREAT | 0666)) <0)
{
    perror("ERROR AL OBTENER MEMORIA COMPARTIDA: shmget"); exit(-1);
}

if((shm3 = shmat(shmid3,NULL,0)) == (int*)-1)
{
    perror("ERROR AL OBTENER MEMORIA COMPARTIDA: shmat");
    exit(-1);
}

s = shm;
for(i = 0, k =0; i < 10; i++)
{
    for(j = 0; j < 10; j++)
    {
        mat1h[i][j] = s[k++];
    }
}
for(i = 0; i < 10; i++)
{
    for(j = 0; j < 10; j++)
    {
        mat2h[i][j] = s[k++];
    }
}
*shm=-10;

printf("se recibio\n");

```

```

printf("se recibio\n");

s3 = shm3;
int z=0;
for (i = 0, z = 0; i < 10; i++)
{
    for (j = 0; j < 10; j++)
    {
        s3[z]=mat1h[i][j]+mat2h[i][j];
        z++;
    }
}
while(*shm3 != -10)
    sleep(1);

printf("se envio\n");

```

```

0.150 -0.050 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
-0.056 0.093 -0.037 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 -0.042 0.073 -0.031 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 -0.036 0.064 -0.029 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 -0.033 0.061 -0.028 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 -0.033 0.062 -0.029 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 -0.036 0.067 -0.031 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 -0.042 0.079 -0.037 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 -0.056 0.106 -0.050
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 -0.100 0.100

0.100 -0.078 -0.014 -0.004 -0.001 -0.001 -0.000 -0.000 -0.000 -0.000
-0.000 0.110 -0.071 -0.020 -0.007 -0.003 -0.002 -0.001 -0.001 -0.000
-0.001 -0.002 0.122 -0.059 -0.022 -0.010 -0.005 -0.003 -0.002 -0.001
-0.003 -0.004 -0.007 0.132 -0.044 -0.020 -0.011 -0.007 -0.004 -0.003
-0.004 -0.006 -0.009 -0.015 0.138 -0.028 -0.015 -0.009 -0.006 -0.004
-0.004 -0.006 -0.009 -0.015 -0.028 0.138 -0.015 -0.009 -0.006 -0.004
-0.003 -0.004 -0.007 -0.011 -0.020 -0.044 0.132 -0.007 -0.004 -0.003
-0.001 -0.002 -0.003 -0.005 -0.010 -0.022 -0.059 0.122 -0.002 -0.001
-0.000 -0.001 -0.001 -0.002 -0.003 -0.007 -0.020 -0.071 0.110 -0.000
-0.000 -0.000 -0.000 -0.000 -0.001 -0.001 -0.004 -0.014 -0.078 0.100

```

```

0.150 -0.050 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
-0.056 0.093 -0.037 0.000 0.000 0.000 0.000 0.000 0.000 0.000
0.000 -0.042 0.073 -0.031 0.000 0.000 0.000 0.000 0.000 0.000
0.000 0.000 -0.036 0.064 -0.029 0.000 0.000 0.000 0.000 0.000
0.000 0.000 0.000 -0.033 0.061 -0.028 0.000 0.000 0.000 0.000
0.000 0.000 0.000 0.000 -0.033 0.062 -0.029 0.000 0.000 0.000
0.000 0.000 0.000 0.000 0.000 -0.036 0.067 -0.031 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.000 -0.042 0.079 -0.037 0.000
0.000 0.000 0.000 0.000 0.000 0.000 0.000 -0.056 0.106 -0.050
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 -0.100 0.100

```


0.100	-0.078	-0.014	-0.004	-0.001	-0.001	-0.000	-0.000	-0.000	-0.000
-0.000	0.110	-0.071	-0.020	-0.007	-0.003	-0.002	-0.001	-0.001	-0.000
-0.001	-0.002	0.122	-0.059	-0.022	-0.010	-0.005	-0.003	-0.002	-0.001
-0.003	-0.004	-0.007	0.132	-0.044	-0.020	-0.011	-0.007	-0.004	-0.003
-0.004	-0.006	-0.009	-0.015	0.138	-0.028	-0.015	-0.009	-0.006	-0.004
-0.004	-0.006	-0.009	-0.015	-0.028	0.138	-0.015	-0.009	-0.006	-0.004
-0.003	-0.004	-0.007	-0.011	-0.020	-0.044	0.132	-0.007	-0.004	-0.003
-0.001	-0.002	-0.003	-0.005	-0.010	-0.022	-0.059	0.122	-0.002	-0.001
-0.000	-0.001	-0.001	-0.002	-0.003	-0.007	-0.020	-0.071	0.110	-0.000
-0.000	-0.000	-0.000	-0.000	-0.001	-0.001	-0.004	-0.014	-0.078	0.100

Conclusiones

Comprendemos gracias a esta práctica que la memoria compartida, junto con los procesos que utilizamos y las tuberías, son los recursos compartidos que pone tanto Linux como Windows a disposición de los programas para que puedan intercambiarse información.

Utilizando el lenguaje C en los sistemas operativos Linux y Windows C es posible hacer que dos procesos o dos programas en este caso, distintos sean capaces de compartir una zona de memoria común y, de esta manera, compartir o comunicarse datos.

También aprendimos que, aunque la memoria utilizada por diferentes procesos suele estar protegida, algunos procesos puede que sí tengan que compartir información y, para ello, han de acceder la misma sección de memoria. La memoria compartida es una de las técnicas más rápidas para posibilitar la comunicación entre procesos.