



**INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO**



SISTEMAS OPERATIVOS

CORTÉS GALICIA JORGE

MENDOZA PARRA SERGIO

OLVERA AGUILA LEONARDO DANIEL

PAZ SÁNCHEZ BRANDON

2CM7

**PRÁCTICA 6. COMUNICACIÓN INTER PROCESOS (IPC) EN LINUX Y
WINDOWS**

LUNES, 04 DE DICIEMBRE DE 2017

Contenido

Competencias 3

Desarrollo 4

Análisis critico 48

Conclusiones..... 49

Competencias

El alumno comprende el funcionamiento de las tuberías (pipes) sin nombre y de la memoria compartida como mecanismos de comunicación entre procesos tanto en el sistema operativo Linux como Windows para el desarrollo de aplicaciones concurrentes con soporte de comunicación.

Desarrollo

1. A través de la ayuda en línea que proporciona Linux, investigue el funcionamiento de la función: **pipe()**, **shmget()**, **shmat()**. Explique los argumentos y retorno de la función.

PIPE ()

Sintaxis

```
#include <unistd.h>
int pipe (int *descriptores);
int descriptores [2];
```

Descripción

Crea un canal de comunicación entre procesos emparentados. Los parámetros que recibe el arreglo son los descriptores de entrada y salida de la tubería (int descriptores [2]). Devuelve 0 si se ha completado correctamente y -1 en caso de error.

- descriptores [0] es el descriptor para el extremo de la tubería que será utilizado para lectura y descriptores [1] es el descriptor para el extremo de la tubería que será utilizado para escritura.
- La operación de lectura sobre la tubería utilizando descriptores [0] accede a los datos escritos en la tubería por medio del descriptor descriptores [1] como en una cola FIFO (primero en llegar, primero en salir)

SHMGET ()

Sintaxis

```
#include <sys/shm.h>
int shmget (key_t key, size_t size, int shmflg);
```

Descripción

La función **shmget** retorna el identificador de memoria compartida asociada a **key**. Un identificador de memoria compartida y la estructura de datos asociada se crearán para **key** si una de las siguientes condiciones se cumple:

Si **key** es igual a **IPC_PRIVATE**. Cuando esto ocurre se creará un nuevo identificador, si existen disponibilidades, este identificador no será devuelto por posteriores invocaciones a **shmget** mientras no se libere mediante la función **shmctl**. El identificador creado podrá ser utilizado por el proceso invocador y sus descendientes; sin embargo, esto no es un requerimiento. El segmento podrá ser accedido por cualquier proceso que posea los permisos adecuados.

Si **key** aún no tiene asociado un identificador de memoria compartida y además **shmflg & IPC_CREAT** es verdadero.

Como consecuencia de la creación, la estructura de datos asociada al nuevo identificador se inicializa de la siguientes manera: **shm_perm.cuid** y **shm_perm.uid** al identificador de usuario efectivo del proceso invocador, **shm_perm.gcuid** y **shm_perm.guid** al identificador de grupo efectivo del proceso invocador, los 9 bits menos significativos de **shm_perm.mode** se inicializan a los 9 bits menos significativos del parámetro **shmflg**, **shm_segsz** al valor especificado por **size**, **shm_ctime** a la fecha y hora que el sistema poseía en el momento de la invocación y por último se ponen a cero **shm_lpid**, **shm_nattach**, **shm_atime** y **shm_dtime**.

Si la ejecución se realiza con éxito, entonces retornará un valor no negativo denominado identificador de segmento compartido. En caso contrario, retornará -1 y la variable global **errno** tomará en código del error producido.

SHMAT ()

Sintaxis

```
#include <sys/shm.h>
char *shmat(int shmid, void *shmaddr, int shmflg );
int shmdt(void *shmaddr )
```

Descripción

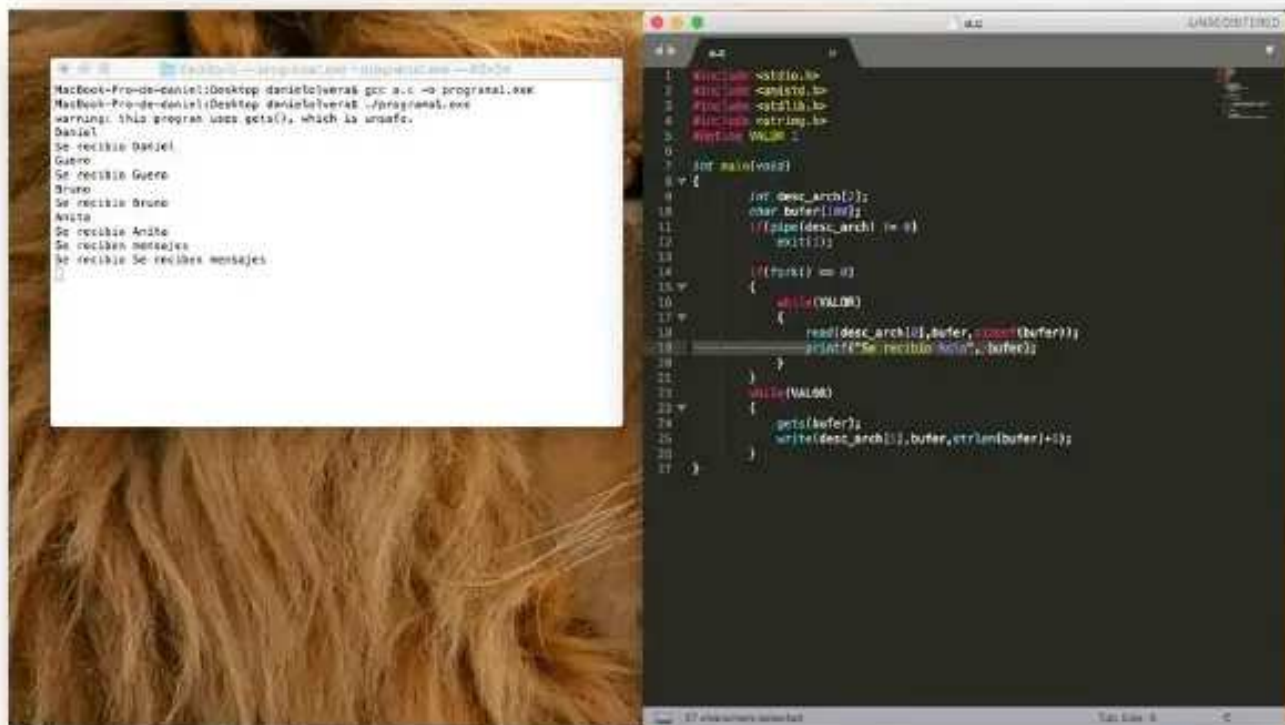
shmat asocia el segmento de memoria compartida especificado por *shmid* al segmento de datos del proceso invocador. Si el segmento de memoria compartida aún no había sido asociado al proceso invocador, entonces *shmaddr* debe tener el valor de cero y el segmento se asocia a una posición en memoria seleccionado por el sistema operativo. Dicha localización será la misma en todos los procesos que acceden al objeto de memoria

compartida. Si el segmento de memoria compartida ya había sido asociado por el proceso invocador, *shmaddr* podrá tener un valor distinto de cero, en ese caso deberá tomar la dirección asociada actual del segmento referenciado por *shmid*. Un segmento se asocia en modo sólo lectura si **shmflg & SHM_RDONLY** es verdadero; si no entonces se podrá acceder en modo lectura y escritura. No es posible la asociación en modo sólo escritura. Si la función se ejecuta con éxito, entonces retornará la dirección de comienzo del segmento compartido, si ocurre un error devolverá -1 y la variable global **errno** tomará el código del error producido.

shmdt desasocia del segmento de datos del proceso invocador el segmento de memoria compartida ubicado en la localización de memoria especificada por *shmaddr*. Si la función se ejecuta sin error, entonces devolverá 0, en caso contrario retornará -1 y **errno** tomará el código del error producido.

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

```
#include <stdio.h>
#include <unistd.h>
#include <string.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)
    {
        gets(bufer);
        write(desc_arch[1], bufer, strlen(bufer)+1);
    }
}
```



Pantalla 1. Compilación en la terminal del programa programa1.c

Observación

Como se puede observar el proceso padre primero ejecuta el gets y escribe una cadena para comunicarse con su hijo, dado que se creó por copia exacta de código se lee lo que se escribe el proceso padre y eso se debe a que se mando la tubería el arreglo que determina la lectura y escritura, es decir, se conectaron los procesos para lograr la comunicación enviando cadenas de texto.

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

```
/*Programa Padre*/
#include "windows.h"
#include "stdio.h"
#include "string.h"

int main(int argc, char *argv[]){
    char mensaje[]="Tuberias en Windows";
    DWORD escritos;
    HANDLE hLecturaPipe,hEscrituraPipe;
    PROCESS_INFORMATION piHijo;
    STARTUPINFO siHijo;
    SECURITY_ATTRIBUTES pipeSeg={sizeof(SECURITY_ATTRIBUTES),NULL,TRUE};
    /*Obtencion de informacion para la inicializacion del proceso hijo*/
    GetStartupInfo(&siHijo);
    /*Creacion de la tuberia sin nombre*/
    CreatePipe(&hLecturaPipe,&hEscrituraPipe,&pipeSeg,0);
    /*Escritura en la tuberia sin nombre*/
    WriteFile(hEscrituraPipe,mensaje,strlen(mensaje)+1,&escritos,NULL);

    siHijo.hStdInput = hLecturaPipe;
    siHijo.hStdError = GetStdHandle(STD_ERROR_HANDLE);
    siHijo.hStdOutput = GetStdHandle(STD_OUTPUT_HANDLE);
    siHijo.dwFlags = STARTF_USESTDHANDLES;

    CreateProcess(NULL,argv[1],NULL,NULL,TRUE,0,NULL,NULL,&siHijo,&piHijo);

    WaitForSingleObject(piHijo.hProcess,INFINITE);
    printf("Mensaje recibido en el proceso hijo, termina el proceso padre\n");
    CloseHandle(hLecturaPipe);
    CloseHandle(hEscrituraPipe);
    CloseHandle(piHijo.hThread);
    CloseHandle(piHijo.hProcess);
    return 0;
}
```



```

/*Programa hijo*/
#include "windows.h"
#include "stdio.h"

int main() {
    char mensaje[20];
    DWORD leidos;
    HANDLE hStdIn=GetStdHandle(STD_INPUT_HANDLE);
    SECURITY_ATTRIBUTES pipeSeg={sizeof(SECURITY_ATTRIBUTES),NULL,TRUE};

    /*Lectura de la tubería sin nombre*/
    ReadFile(hStdIn,mensaje,sizeof(mensaje),&leidos,NULL);
    printf("Mensaje recibido del proceso padre: %s\n",mensaje);
    CloseHandle(hStdIn);
    printf("Termina el proceso Hijo, continua el proceso Padre\n");
    return 0;
}

```

```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Versión 10.0.15063]
(c) 2017 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Sergi>cd Desktop
C:\Users\Sergi\Desktop>cd Punto3
C:\Users\Sergi\Desktop\Punto3>Padre.exe Hijo.exe
Mensaje recibido del proceso padre: Tuberias en Windows
Termina el proceso Hijo, continua el proceso Padre
Mensaje recibido en el proceso hijo, termina el proceso padre
C:\Users\Sergi\Desktop\Punto3>

```

Pantalla 2. Compilacion en la terminal del programa

Observación

En estos 2 programas pudimos observar que a travez de tuberías se crean procesos uno padre y otro hijo, los cuales van a crear archivos controlados internamente por el sistema, de tal modo que, sirven como buferes los cuales va a almacenar los datos a comunicar y en este caso fue un mensaje de Tuberías en Windows.

4. Programe una aplicación que cree un proceso hijo a partir de un proceso padre, el proceso padre enviará al proceso hijo, a través de una tubería, dos matrices de 15 x 15 a multiplicar por parte del hijo, mientras tanto el proceso hijo creará un hijo de él, al cual enviará dos matrices de 15 x 15 a sumar en el proceso hijo creado, nuevamente el envío de estos valores será a través de una tubería. Una vez calculado el resultado de la suma, el proceso hijo del hijo devolverá la matriz resultante a su abuelo (vía tubería). A su vez, el proceso hijo devolverá la matriz resultante de la multiplicación que realizó a su padre. Finalmente, el proceso padre obtendrá la matriz inversa de cada una de las matrices recibidas y el resultado lo guardará en un archivo para cada matriz inversa obtenida. Programe esta aplicación tanto para Linux como para Windows utilizando las tuberías de cada sistema operativo.

Sección Linux

```
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<stdlib.h>
#define VALOR 1

void generar(int arr[][15],int val,int filas);
int main(void)
{
    int i,l,j,a[2],b[2],c[2],d[2];
    int
matrizA[15][15],matrizB[15][15],matrizC[15][15],matrizD[15][15],suma[15][
15];
    int
matrizE[15][15],matrizF[15][15],matrizG[15][15],matrizH[15][15],multi[15]
[15];
    int suma_res[15][15],multi_res[15][15];
    char bufer[50], nombre_suma[19] = "resultados.txt";

    FILE *fichero,*fp;

    generar(matrizA,2,15);
    generar(matrizB,3,15);

    if(pipe(a)!=0)
        exit(1);
    if(pipe(c)!=0)
        exit(1);
    if(pipe(d)!=0)
        exit(1);

    if(fork()==0)
    {
        read(a[0],matrizC,sizeof(matrizC));
        printf("Se recibe: \n\n");
        for(i=0;i<15;i++)
        {
```

```

        for(j=0;j<15;j++)
        {
            printf(" %d",matrizC[i][j]);
        }
        printf("\n");
    }
    read(a[0],matrizD,sizeof(matrizD));
    printf("\nSe recibe tambien: \n\n");

    for(i=0;i<15;i++)
    {
        for(j=0;j<15;j++)
        {
            printf(" %d",matrizD[i][j]);
        }
        printf("\n");
    }

    printf("\n\tEl hijo uno realizando la suma de matrices:\n");
    for (i=0;i<15;i++)
    {
        for(j=0;j<15;j++)
        {
            printf("
%d",suma[i][j]=(matrizC[i][j]+matrizD[i][j]));
        }
        printf("\n");
    }

    write(c[1],suma,sizeof(suma));

    generar(matrizD,2,15);
    generar(matrizE,3,15);

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

    write(b[1],matrizD,sizeof(matrizD));
    write(b[1],matrizE,sizeof(matrizE));

    if(fork()==0)
    {

        printf("\n El hijo dos recibiendo: \n\n");
        read(b[0],matrizF,sizeof(matrizF));
        read(b[0],matrizG,sizeof(matrizG));

        printf("Se recibe: \n\n");
        printf("\n\t Primera matriz: \n");

        for(i=0;i<15;i++)
        {
            for(j=0;j<15;j++)
            {
                printf(" %d",matrizF[i][j]);
            }
        }
    }

```

```

        }
        printf("\n");
    }
    printf("\n\t Segunda matriz: \n");
    for(i=0;i<15;i++)
    {
        for(j=0;j<15;j++)
        {
            printf(" %d",matrizG[i][j]);
        }
        printf("\n");
    }
    printf("\n\tEl segundo hijo realizando la
multiplicacion:\n");
    for(i=0;i<15;i++)
    {
        for(j=0;j<15;j++)
        {
            printf("
%d",multi[i][j]=(matrizF[i][j]*matrizG[i][j]*15));
        }
        printf("\n");
    }
    write(d[1],multi,sizeof(multi));
}

}

write(a[1],matrizA,sizeof(matrizA));
write(a[1],matrizB,sizeof(matrizB));

read(c[0],suma_res,sizeof(suma_res));
read(d[0],multi_res,sizeof(multi_res));
printf("\n:D El resultado de la suma es:\n");
for (i=0;i<15;i++)
{
    printf("\t");
    for(j=0;j<15;j++)
    {
        printf(" %d",suma_res[i][j]);
    }
    printf("\n");
}

fichero = fopen ("resultados.txt", "a+");
printf( "\nFichero: %s -> ", nombre_suma );
if(fichero)
{

    printf( "\ncreado (ABIERTO)\n" );
    fputs("\n Resultado suma:\n\n",fichero);
    for(i=0;i<15;i++){
        for(j=0;j<15;j++)
        {
            fprintf(fichero,"%d ",(suma_res[i][j]));
        }
    }
}

```

```

        fprintf(fichero, "\n");
    }
    fputs("\n Resultado multiplicacion:\n\n", fichero);
    for(i=0; i<15; i++) {
        for(j=0; j<15; j++)
        {
            fprintf(fichero, "%d ", (multi_res[i][j]));
        }
        fprintf(fichero, "\n");
    }
}
else
{
    printf( "Error (NO ABIERTO)\n" );
}

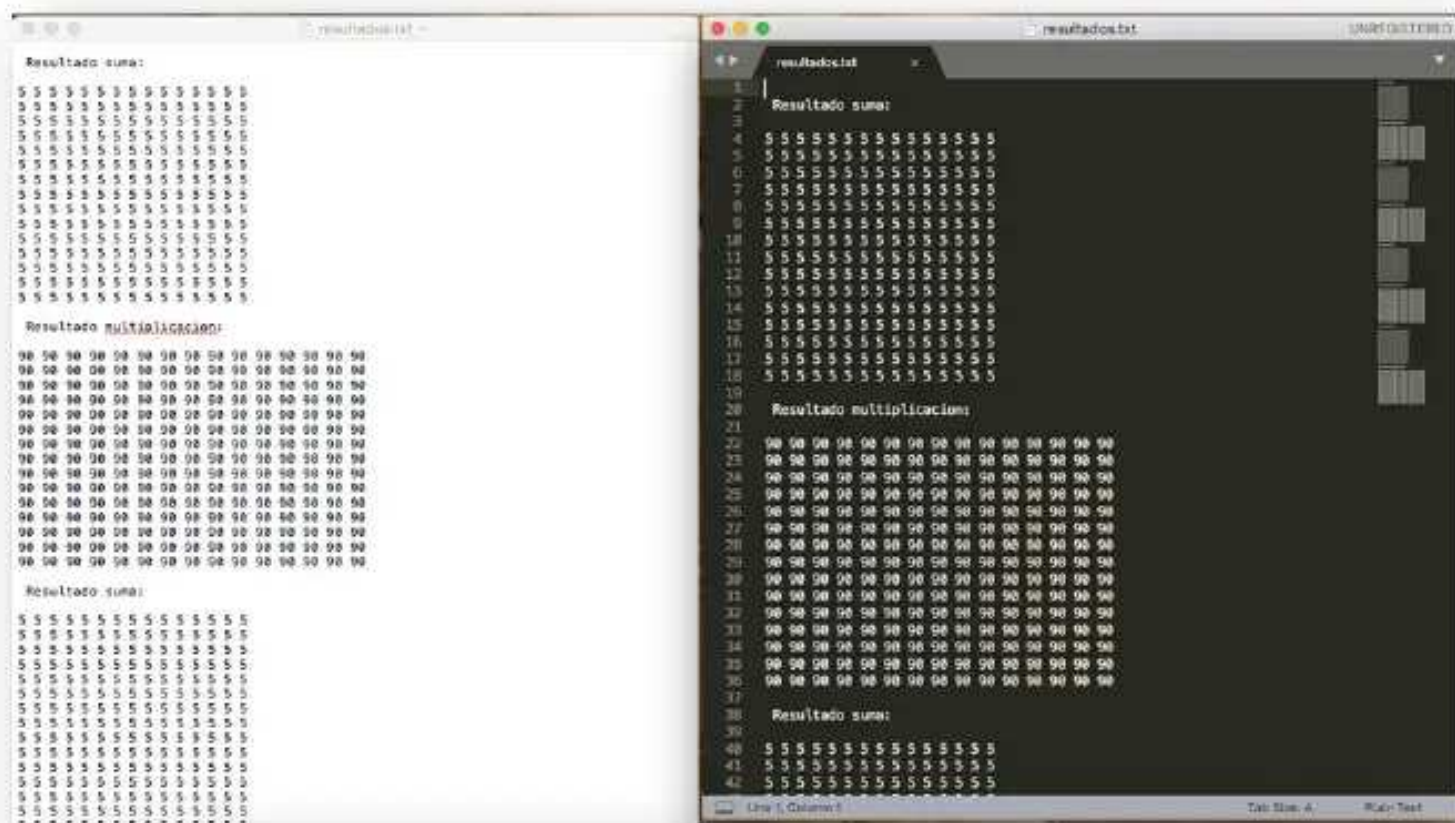
if( !fclose(fichero) )
    printf( "\nFichero cerrado\n" );
else
{
    printf( "\nError: fichero NO CERRADO\n" );
}

}

void generar(int arr[][15], int val, int filas)
{
    int j, i;
    for (i=0; i<filas; i++)
    {
        for(j=0; j<15; j++)
        {
            arr[i][j]=val;
        }
        printf("\n");
    }
}

```





Pantalla 5. Resultados de la compilación del programa 4

Sección Windows

```
/*Programa Padre*/
#include "windows.h"
#include "stdio.h"
#include "string.h"

char mensaje[] = "8,5,6,4,1,8,2,7,3,3,";

char matriz12[] = "7,5,5,3,3,2,3,1,2,9,";
char matriz13[] = "3,5,4,2,2,1,4,7,6,7,";
char matriz14[] = "2,4,5,8,7,1,2,6,5,4,";
char matriz15[] = "8,8,1,2,7,5,1,5,5,1,";
char matriz16[] = "5,6,7,5,6,4,2,4,9,5,";
char matriz17[] = "6,4,9,8,6,2,5,6,5,3,";
char matriz18[] = "3,5,6,2,8,4,3,4,8,1,";
char matriz19[] = "1,7,2,4,8,4,9,5,6,3,";
char matriz110[] = "6,1,4,5,8,8,8,5,7,4,";

char matriz21[] = "6,4,2,5,6,3,2,6,5,7,";
char matriz22[] = "2,3,6,1,4,2,1,4,2,5,";
char matriz23[] = "4,8,7,4,3,7,2,6,3,3,";
char matriz24[] = "8,6,7,6,4,7,8,4,1,2,";
char matriz25[] = "1,3,9,4,6,8,8,8,6,9,";
char matriz26[] = "7,4,8,5,3,8,6,9,4,5,";
char matriz27[] = "8,7,5,7,4,8,2,2,9,8,";
char matriz28[] = "1,2,6,8,8,2,4,6,2,8,";
char matriz29[] = "9,2,6,1,3,7,2,3,4,1,";
char matriz210[] = "5,3,5,3,4,2,4,8,2,5";

int main() {
    //char mensaje[]="1,2,3,4,5,6,7,8";
    strcat(mensaje,matriz12);
    strcat(mensaje,matriz13);
    strcat(mensaje,matriz14);
    strcat(mensaje,matriz15);
    strcat(mensaje,matriz16);
    strcat(mensaje,matriz17);
    strcat(mensaje,matriz18);

    strcat(mensaje,matriz19);
    strcat(mensaje,matriz110);
    strcat(mensaje,matriz21);
    strcat(mensaje,matriz22);
    strcat(mensaje,matriz23);
    strcat(mensaje,matriz24);
    strcat(mensaje,matriz25);
    strcat(mensaje,matriz26);
    strcat(mensaje,matriz27);
    strcat(mensaje,matriz28);
    strcat(mensaje,matriz29);
    strcat(mensaje,matriz210);
    DWORD escritos;
    HANDLE hLecturaPipe,hEscrituraPipe;
    PROCESS_INFORMATION piHijo;
    STARTUPINFO siHijo;
    SECURITY_ATTRIBUTES pipeSeg={sizeof(SECURITY_ATTRIBUTES),NULL,TRUE};
```



```

    /*Obtencion de informacion para la inacializacion del proceso hijo*/
    GetStartupInfo (&siHijo);
    /*Creacion de la tuberia sin nombre*/
    CreatePipe (&hLecturaPipe,&hEscrituraPipe,&pipeSeg,0);
    /*Escritura en la tuberia sin nombre*/
    WriteFile (hEscrituraPipe,mensaje,strlen(mensaje)+1,&escritos,NULL);

    siHijo.hStdInput = hLecturaPipe;
    siHijo.hStdError = GetStdHandle(STD_ERROR_HANDLE);
    siHijo.hStdOutput = GetStdHandle(STD_OUTPUT_HANDLE);
    siHijo.dwFlags = STARTF_USESTDHANDLES;

    CreateProcess (NULL,"Hijo1",NULL,NULL,TRUE,0,NULL,NULL,&siHijo,&piHijo);

    WaitForSingleObject (piHijo.hProcess,INFINITE);
    printf("Mensaje recibido en el proceso hijo, termina el proceso
padre\n");
    CloseHandle (hLecturaPipe);
    CloseHandle (hEscrituraPipe);
    CloseHandle (piHijo.hThread);
    CloseHandle (piHijo.hProcess);
    return 0;
}

```

```

/*Programa hijo*/
#include "windows.h"
#include "stdio.h"
#include "string.h"

char mensaje2[] = "6,4,2,5,6,3,2,6,5,7,";
char matriz22[] = "2,3,6,1,4,2,1,4,2,5,";
char matriz23[] = "4,8,7,4,3,7,2,6,3,3,";
char matriz24[] = "8,6,7,6,4,7,8,4,1,2,";
char matriz25[] = "1,3,9,4,6,8,8,8,6,9,";
char matriz26[] = "7,4,8,5,3,8,6,9,4,5,";
char matriz27[] = "8,7,5,7,4,8,2,2,9,8,";
char matriz28[] = "1,2,6,8,8,2,4,6,2,8,";
char matriz29[] = "9,2,6,1,3,7,2,3,4,1,";
char matriz210[] = "5,3,5,3,4,2,4,8,2,5";

char matriz11[] = "8,5,6,4,1,8,2,7,3,3,";
char matriz12[] = "7,5,5,3,3,2,3,1,2,9,";
char matriz13[] = "3,5,4,2,2,1,4,7,6,7,";
char matriz14[] = "2,4,5,8,7,1,2,6,5,4,";
char matriz15[] = "8,8,1,2,7,5,1,5,5,1,";
char matriz16[] = "5,6,7,5,6,4,2,4,9,5,";
char matriz17[] = "6,4,9,8,6,2,5,6,5,3,";
char matriz18[] = "3,5,6,2,8,4,3,4,8,1,";

char matriz19[] = "1,7,2,4,8,4,9,5,6,3,";
char matriz110[] = "6,1,4,5,8,8,8,5,7,4,";

void Multiplicar(int matriz[10][10],int matriz2[10][10]);
int CreandoTuberia(char mensaje[]);
int C[10][10];

////////////////////////////////////////
int const Tam=100;
void PideDatos(int k,int Dim, float Sist[][Tam]);
void Invierte(int Dim, float Sist[][Tam], float Inv[][Tam]);
void Inversa();
////////////////////////////////////////

int main(){
    char mensaje[450];
    //char mensaje2[] = "9,8,7,6,5,4,3,2";
    strcat(mensaje2,matriz22);
    strcat(mensaje2,matriz23);
    strcat(mensaje2,matriz24);
    strcat(mensaje2,matriz25);
    strcat(mensaje2,matriz26);
    strcat(mensaje2,matriz27);
    strcat(mensaje2,matriz28);
    strcat(mensaje2,matriz29);
    strcat(mensaje2,matriz210);
    strcat(mensaje2,matriz11);
    strcat(mensaje2,matriz12);
    strcat(mensaje2,matriz13);
    strcat(mensaje2,matriz14);
    strcat(mensaje2,matriz15);

    strcat(mensaje2,matriz16);
    strcat(mensaje2,matriz17);

```

```

strcat(mensaje2,matriz18);
strcat(mensaje2,matriz19);
strcat(mensaje2,matriz110);

int matriz[10][10];
int matriz2[10][10];
int i,j;
int k=0;
DWORD leidos;
HANDLE hStdln=GetStdHandle(STD_INPUT_HANDLE);
SECURITY_ATTRIBUTES pipeSeg={sizeof(SECURITY_ATTRIBUTES),NULL,TRUE};
/*Lectura de la tubería sin nombre*/
ReadFile(hStdln,mensaje,sizeof(mensaje),&leidos,NULL);
printf("Mensaje recibido del proceso padre: %s\n",mensaje);
for(i=0;i<20;i++)
{
    for(j=0;j<20;j++)
    {
        if(i<10&&j<10){
            if(k==0){
                matriz[i][j] = atoi(strtok(mensaje","));
                k++;
            }
            else
                matriz[i][j] = atoi(strtok(NULL","));
        }
        if(i>9&&j>9){
            matriz2[i-10][j-10] = atoi(strtok(NULL","));
        }
    }
}
Multiplicar(matriz,matriz2);
printf("\n");
for(i=0;i<10;i++)
{
    for(j=0;j<10;j++)
    {
        printf("\t %d",C[i][j]);
    }
    printf("\n");
}
Inversa();
CreandoTuberia(mensaje2);

CloseHandle(hStdln);
printf("Termina el proceso Hijo, continua el proceso Padre\n");
return 0;
}

void Multiplicar(int matriz[10][10],int matriz2[10][10])
{
    int i,j,k;
    //OPERACION DE MULTIPLICACION
    for (i=0;i<10;i++)
    {
        for (j=0;j<10;j++)
        {

```

```

        C[i][j]=0;
        for (k=0;k<10;k++)
        {
            C[i][j]=C[i][j]+matriz[i][k]*matriz2[k][j];
        }
    }
}

int CreandoTuberia(char mensaje[]){
    DWORD escritos2;
    HANDLE hLecturaPipe2,hEscrituraPipe2;
    PROCESS_INFORMATION piHijo2;
    STARTUPINFO siHijo2;
    SECURITY_ATTRIBUTES pipeSeg2={sizeof(SEcurity_ATTRIBUTES),NULL,TRUE};
    /*Obtencion de informacion para la inacializacion del proceso nieto*/
    GetStartupInfo(&siHijo2);
    /*Creacion de la tuberia sin nombre*/
    CreatePipe(&hLecturaPipe2,&hEscrituraPipe2,&pipeSeg2,0);
    /*Escritura en la tuberia sin nombre*/
    WriteFile(hEscrituraPipe2,mensaje,strlen(mensaje)+1,&escritos2,NULL);

    siHijo2.hStdInput = hLecturaPipe2;
    siHijo2.hStdError = GetStdHandle(STD_ERROR_HANDLE);
    siHijo2.hStdOutput = GetStdHandle(STD_OUTPUT_HANDLE);
    siHijo2.dwFlags = STARTF_USESTDHANDLES;

    CreateProcess(NULL,"Nieto1",NULL,NULL,TRUE,0,NULL,NULL,&siHijo2,&piHijo2);
    ;

    WaitForSingleObject(piHijo2.hProcess,INFINITE);
    printf("Mensaje recibido en el proceso Nieto, termina el proceso
Hijo\n");
    CloseHandle(hLecturaPipe2);
    CloseHandle(hEscrituraPipe2);
    CloseHandle(piHijo2.hThread);
    CloseHandle(piHijo2.hProcess);
    return 0;
}

//Matriz Inversa
void Inversa()
{
    int Dimension=10,k=1;
    FILE *fichero;
    int i,j;
    float Sistema[Tam][Tam],Inversa[Tam][Tam];
    PideDatos(k,Dimension,Sistema);
    Invierte(Dimension,Sistema,Inversa);
    fichero = fopen("Resultado.txt","w");
    for(i=0;i<10;i++)
    {
        for(j=0;j<10;j++)
        {
            fprintf(fichero,".1f\n",Inversa[i][j]);
        }
    }
}

```

```

        fclose(fichero);
        printf("El resultado ha sido almacenado en el archivo Resultado\n");
    return;
}

void PideDatos(int k,int Dim,float Sist[][Tam])
{
    int i,j;
    for(i=1;i<=Dim;i++) for(j=1;j<=Dim;j++){
        Sist[i][j] = C[i-1][j-1];
    }
}

void Invierte(int Dim, float Sist[][Tam], float Inv[][Tam])
{
    int NoCero,Col,C1,C2,A;
    float Pivote,V1,V2;

    /*Se inicializa la matriz inversa, como la matriz identidad:*/
    for(C1=1;C1<=Dim;C1++) for(C2=1;C2<=Dim;C2++)
        if (C1==C2) Inv[C1][C2]=1; else Inv[C1][C2]=0;

    for(Col=1;Col<=Dim;Col++){
        NoCero=0;A=Col;
        while(NoCero==0){
            if((Sist[A][Col]>0.0000001)||((Sist[A][Col]<-0.0000001))){
                NoCero=1;
            }
            else A++;
        }
        Pivote=Sist[A][Col];
        for(C1=1;C1<=Dim;C1++){
            V1=Sist[A][C1];
            Sist[A][C1]=Sist[Col][C1];
            Sist[Col][C1]=V1/Pivote;
            V2=Inv[A][C1];
            Inv[A][C1]=Inv[Col][C1];
            Inv[Col][C1]=V2/Pivote;
        }
        for(C2=Col+1;C2<=Dim;C2++){
            V1=Sist[C2][Col];
            for(C1=1;C1<=Dim;C1++){
                [ ][ ] = [ ][ ] - * [ ][ ];
                Sist C2 C1 Inv C2 C1 = Inv C2 C1 - V1 * Inv Col C1;
            }
        }
    }

    /*Aqui ya esta triangularizada, con 1s en diagonal, ahora se
    diagonaliza*/
    for(Col=Dim;Col>=1;Col--) for(C1=(Col-1);C1>=1;C1--)
    {
        V1=Sist[C1][Col];
        for(C2=1;C2<=Dim;C2++){
            Sist[C1][C2]=Sist[C1][C2]-V1*Sist[Col][C2];
            Inv[C1][C2]=Inv[C1][C2]-V1*Inv[Col][C2];
        }
    }
}

```

```

/*Programa Nieto*/
#include "windows.h"
#include "stdio.h"
#include "string.h"

void Sumar(int matriz[10][10],int matriz2[10][10]);
int C[10][10];

////////////////////////////////////
int const Tam=100;
void PideDatos(int k,int Dim, float Sist[][Tam]);
void Invierte(int Dim, float Sist[][Tam], float Inv[][Tam]);
void Inversa();
////////////////////////////////////

int main() {
    char mensaje2[450];
    int matriz[10][10];
    int matriz2[10][10];
    int i,j;
    int k=0;
    DWORD leidos;
    HANDLE hStdln=GetStdHandle(STD_INPUT_HANDLE);
    SECURITY_ATTRIBUTES pipeSeg={sizeof(SECURITY_ATTRIBUTES),NULL,TRUE};
    /*Lectura de la tubería sin nombre*/
    ReadFile(hStdln,mensaje2,sizeof(mensaje2),&leidos,NULL);
    printf("Mensaje recibido del proceso Hijo: %s\n",mensaje2);
    for(i=0;i<20;i++)
    {
        for(j=0;j<20;j++)
        {
            if(i<10&&j<10){
                if(k==0){
                    matriz[i][j] = atoi(strtok(mensaje2,","));
                    k++;
                }
                else
                    matriz[i][j] = (atoi(strtok(NULL,",")));
            }
            if(i>9&&j>9){
                matriz2[i-10][j-10] = atoi(strtok(NULL,","));
            }
        }
    }
    Sumar(matriz,matriz2);
    printf("\n");
    for(i=0;i<10;i++)
    {
        for(j=0;j<10;j++)
        {
            printf("\t %d",C[i][j]);
        }
        printf("\n");
    }
    Inversa();
}

```

```

        CloseHandle(hStdIn);
        printf("Termina el proceso Nieto, continua el proceso Hijo\n");
        return 0;
    }

void Sumar(int matriz[10][10],int matriz2[10][10])
{
    for(i=0;i<10;i++)
    {
        for(j=0;j<10;j++)
        {
            C[i][j] = matriz[i][j]+matriz2[i][j];
        }
    }
}

//Matriz Inversa
void Inversa()
{
    int Dimension=10,k=1;
    FILE *fichero;
    int i,j;
    float Sistema[Tam][Tam],Inversa[Tam][Tam];
    PideDatos(k,Dimension,Sistema);
    Invierte(Dimension,Sistema,Inversa);
    fichero = fopen("Resultado.txt","w");
    for(i=0;i<10;i++)
    {
        for(j=0;j<10;j++)
        {
            fprintf(fichero,"%1f\n",Inversa[i][j]);
        }
    }
    fclose(fichero);
    printf("El resultado ha sido almacenado en el archivo Resultado\n");
    return;
}

void PideDatos(int k,int Dim,float Sist[][Tam])
{
    int i,j;
    for(i=1;i<=Dim;i++) for(j=1;j<=Dim;j++){
        Sist[i][j] = C[i-1][j-1];
    }
}

void Invierte(int Dim, float Sist[][Tam], float Inv[][Tam])
{
    int NoCero,Col,C1,C2,A;
    float Pivote,V1,V2;

    /*Se inicializa la matriz inversa, como la matriz identidad:*/
    for(C1=1;C1<=Dim;C1++) for(C2=1;C2<=Dim;C2++)
        if (C1==C2) Inv[C1][C2]=1; else Inv[C1][C2]=0;

    for(Col=1;Col<=Dim;Col++){

```

```

NoCero=0;A=Col;
while(NoCero==0){
    if((Sist[A][Col]>0.0000001)||((Sist[A][Col]<-0.0000001))){
        NoCero=1;}
    else A++;}
Pivote=Sist[A][Col];
for(C1=1;C1<=Dim;C1++){
    V1=Sist[A][C1];
    [ ][ ]=[ ][ ];
    Sist[A][C1]=V1/Pivote;
    V2=Inv[A][C1];
    Inv[A][C1]=Inv[Col][C1];
    Inv[Col][C1]=V2/Pivote;
}
for(C2=Col+1;C2<=Dim;C2++){
    V1=Sist[C2][Col];
    for(C1=1;C1<=Dim;C1++){
        Sist[C2][C1]=Sist[C2][C1]-V1*Sist[Col][C1];
        Inv[C2][C1]=Inv[C2][C1]-V1*Inv[Col][C1];}
}}

/*Aqui ya esta triangularizada, con 1s en diagonal, ahora se
diagonaliza*/
for(Col=Dim;Col>=1;Col--) for(C1=(Col-1);C1>=1;C1--)
{
    V1=Sist[C1][Col];
    for(C2=1;C2<=Dim;C2++){
        Sist[C1][C2]=Sist[C1][C2]-V1*Sist[Col][C2];
        Inv[C1][C2]=Inv[C1][C2]-V1*Inv[Col][C2];
    }}
}

```



```

C:\WINDOWS\system32\cmd.exe
C:\Users\Sergi\Desktop\Punto4>Padre1.exe Hijo1.exe Nieto1.exe
Mensaje recibido del proceso padre: 8,5,6,4,1,8,2,7,3,3,7,5,5,3,3,2,3,1,2,9,3,5,4,2,2,3,4,7,6,7,2,4,5,8,7,1,2,6,5,4,8,8,
1,2,7,5,1,5,5,1,5,6,7,5,6,4,2,4,9,5,6,4,8,8,6,2,5,6,5,3,3,5,6,2,8,4,3,4,8,1,1,7,2,4,8,4,9,5,6,3,6,1,4,5,8,8,5,7,4,6,4,
2,5,6,3,2,6,5,7,2,3,6,1,4,2,1,4,2,5,4,8,7,4,3,7,2,8,3,3,8,6,7,6,4,7,8,4,1,2,1,3,9,4,6,8,8,6,9,7,4,8,5,3,8,6,9,4,5,8,7,
5,7,4,8,2,2,9,8,1,2,6,8,8,2,4,6,2,8,9,2,6,1,3,7,2,3,4,3,5,3,5,3,4,2,4,8,2,5

236 107 274 219 217 233 171 279 160 246
281 172 221 158 175 185 139 236 144 211
297 156 237 172 101 187 131 219 143 214
285 181 286 194 287 248 198 242 145 222
289 147 258 172 211 212 164 258 164 249
270 213 329 203 230 288 280 295 187 252
270 249 331 249 248 302 214 289 280 277
289 175 288 172 196 261 169 244 178 229
242 199 316 234 222 285 195 252 214 281
310 237 349 264 253 342 239 319 247 312

El resultado ha sido almacenado en el archivo Resultado
Mensaje recibido del proceso Hijo: 6,4,2,5,6,3,2,6,5,7,2,3,6,1,4,2,1,4,2,5,4,8,7,4,3,7,2,6,3,3,8,6,7,6,4,7,8,4,1,2,1,3,0
4,6,8,8,8,6,9,7,4,8,5,3,8,6,9,4,5,8,7,5,7,4,8,2,2,0,8,1,2,6,8,8,2,4,6,2,8,9,2,6,1,3,7,2,3,4,1,5,3,5,3,4,2,4,8,2,58,5,6,
4,1,8,2,7,8,3,7,5,5,3,3,2,3,1,2,9,3,5,4,2,2,1,4,7,6,7,2,4,5,8,7,1,2,6,5,4,8,8,1,2,7,5,1,5,5,1,5,6,7,5,6,4,2,4,9,5,6,4,9,
8,6,2,5,6,5,3,3,5,6,2,8,4,3,4,8,1,1,7,2,4,8,4,0,5,6,3,6,1,4,5,8,8,8,5,7,4,

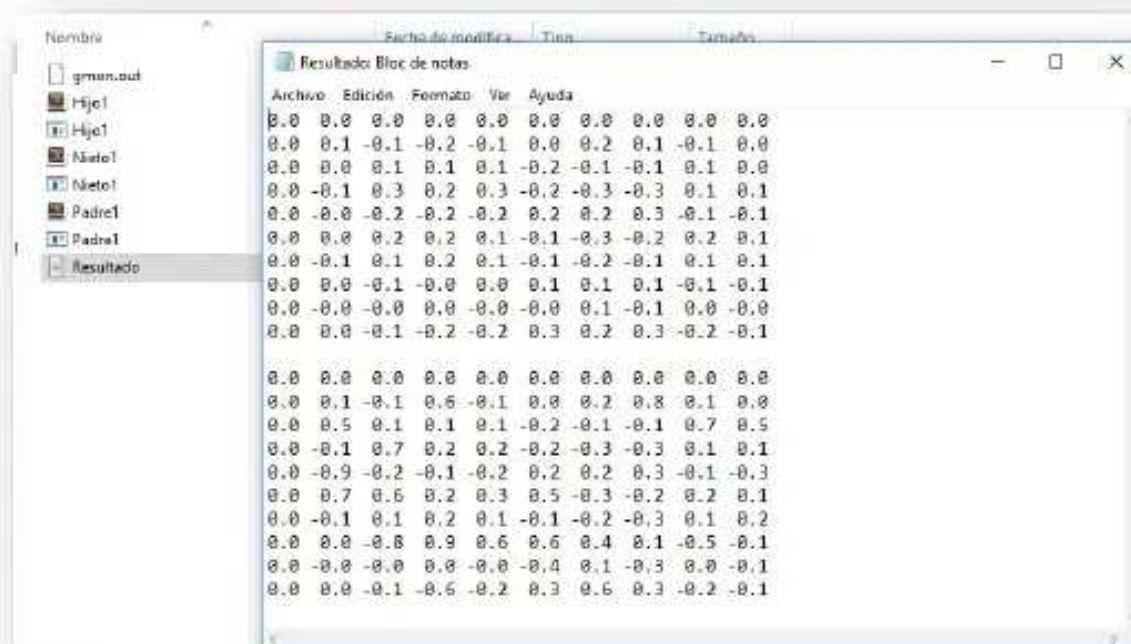
11 18 6 6 14 5 9 9 8 14
7 8 9 4 6 5 2 6 11 8
9 12 9 8 4 11 9 12 10 5
12 11 15 13 5 9 14 9 5 19
9 4 11 11 11 9 13 13 7 14
13 11 13 11 7 10 10 18 9 11
12 16 13 13 6 13 8 7 12 11
6 8 8 16 12 5 8 14 3 9
16 4 10 9 7 16 7 0 7 7
6 7 10 11 12 10 9 15 6 58

El resultado ha sido almacenado en el archivo Resultado
Termina el proceso Nieto, continua el proceso Hijo
Mensaje recibido en el proceso Nieto, termina el proceso Hijo
Termina el proceso Hijo, continua el proceso Padre
Mensaje recibido en el proceso Hijo, termina el proceso padre
C:\Users\Sergi\Desktop\Punto4>

```

Pantalla 6. Compilación en la terminal del programa 4.

El resultado de la matriz inversa se almacena en un archivo llamado Resultado.txt
Aquí se muestra el resultado de las matrices inversas:



Pantalla 7. Resultados de la compilación del programa 4.

5. Capture, compile y ejecute los siguientes programas para Linux. Observe su funcionamiento.

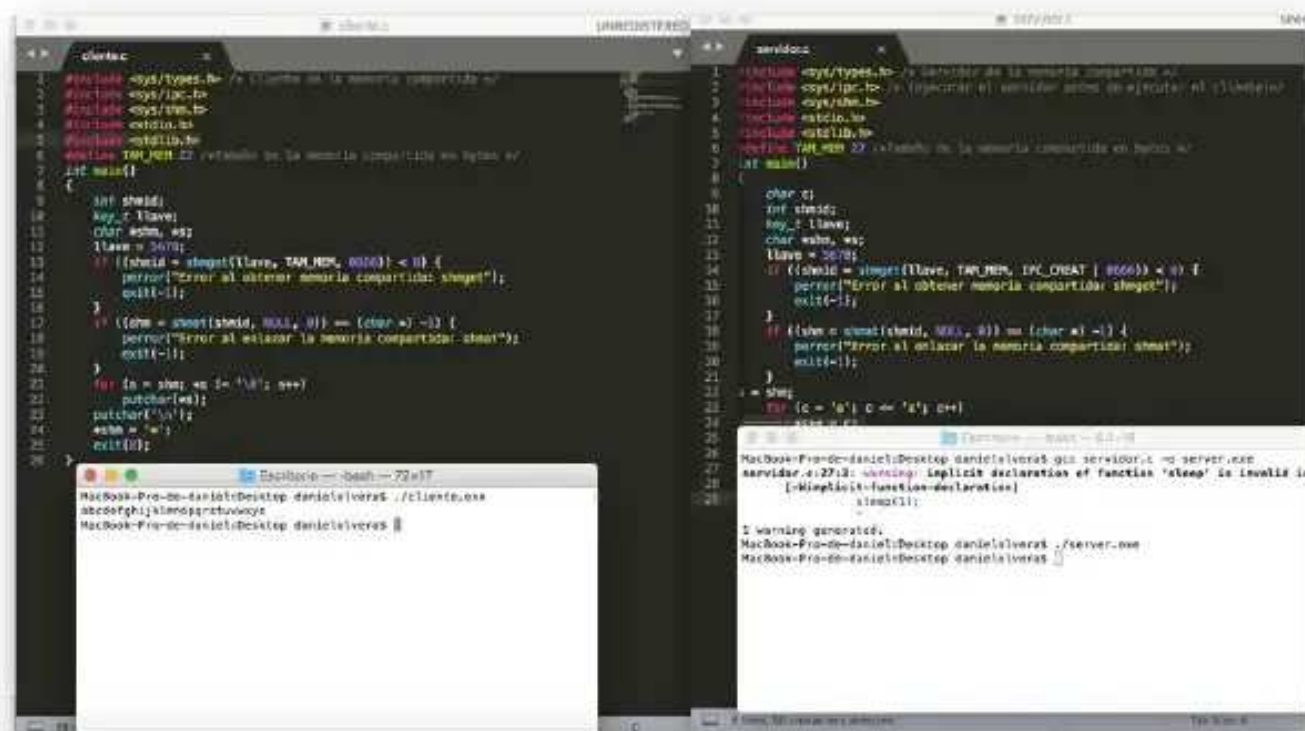
```
#include <sys/types.h> /* Cliente de la memoria compartida */
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#define TAM_MEM 27 /*Tamaño de la memoria compartida en bytes */
int main()
{
    int shmid;
    key_t llave;
    char *shm, *s;
    llave = 5678;
    if ((shmid = shmget(llave, TAM_MEM, 0666)) < 0) {
        perror("Error al obtener memoria compartida: shmget");
        exit(-1);
    }
    if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
        perror("Error al enlazar la memoria compartida: shmat");
        exit(-1);
    }
    for (s = shm; *s != '\\0'; s++)
        putchar(*s);
    putchar('\\n');
    *shm = '*';
    exit(0);
}
```

```

#include <sys/types.h> /* Servidor de la memoria compartida */
#include <sys/ipc.h> /* (ejecutar el servidor antes de ejecutar el
cliente)*/
#include <sys/shm.h>
#include <stdio.h>
#define TAM_MEM 27 /*Tamaño de la memoria compartida en bytes */
int main()
{
    char s[27];
    key_t llave;
    char *shm, *s;
    llave = 5678;
    if ((shm = shmget(llave, TAM_MEM, IPC_CREAT | 0666)) < 0) {
        perror("Error al obtener memoria compartida: shmget");
        exit(-1);
    }
    if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
        perror("Error al enlazar la memoria compartida: shmat");
        exit(-1);
    }
    s = shm;
    for (c = 'a'; c <= 'z'; c++)
        *s++ = c;

    *s = '\0';
    while (*shm != '*')
        sleep(1);
    exit(0);
}

```



Pantalla 8. Compilación en la terminal del programa 5.

Observación

Este programa pude observar que al declarar una llave podemos acceder a una cierta región de memoria para que cuando exista un servidor de memoria compartida cualquier proceso que obtenga esa llave pueda comunicarse con el otro proceso.

6. Capture, compile y ejecute los siguientes programas para Windows. Observe su funcionamiento.

```
/*Programa Servidor*/
#include <windows.h> /*Cliente de la memoria compartida*/
#include <stdio.h>
#include <unistd.h>
#define TAM_MEM 27 /*Tamaño de la memoria compartida en bytes*/

int main(void){
    HANDLE hArchMapeo;
    char * idMemCompartida = "MemoriaCompartida";
    char * apDatos, *apTrabajo, c;
    if((hArchMapeo = CreateFileMapping(
        INVALID_HANDLE_VALUE, //usa la memoria compartida
        NULL, //seguridad por default
        PAGE_READWRITE, //acceso lectura/escritura a la memoria
        0, //tamaño maximo parte alta de un DWORD
        TAM_MEM, //tamaño maximo parte baja de un DWORD
        idMemCompartida) //Identificador de la memoria compartida
    ) == NULL)
    {
        printf("No se mapeo la memoria compartida: (%i)\n",
            GetLastError());
        exit(-1);
    }
    if((apDatos = (char *)MapViewOfFile(hArchMapeo, //Manejador del mapeo
        FILE_MAP_ALL_ACCESS, //Permiso de lectura/escritura en la memoria
        0,
        0,
        TAM_MEM)) == NULL)
    {
        printf("No se accedio a la memoria compartida: (%i)\n",
            GetLastError());
        CloseHandle(hArchMapeo);
        exit(-1);
    }
    apTrabajo = apDatos;
    for(c = 'a' ; c <= 'z'; c++)
        *apTrabajo++ = c;
    *apTrabajo = '\0';
    while(*apDatos != '*')
        sleep(1);
    UnmapViewOfFile(apDatos);
    CloseHandle(hArchMapeo);
    exit(0);
}
```

```

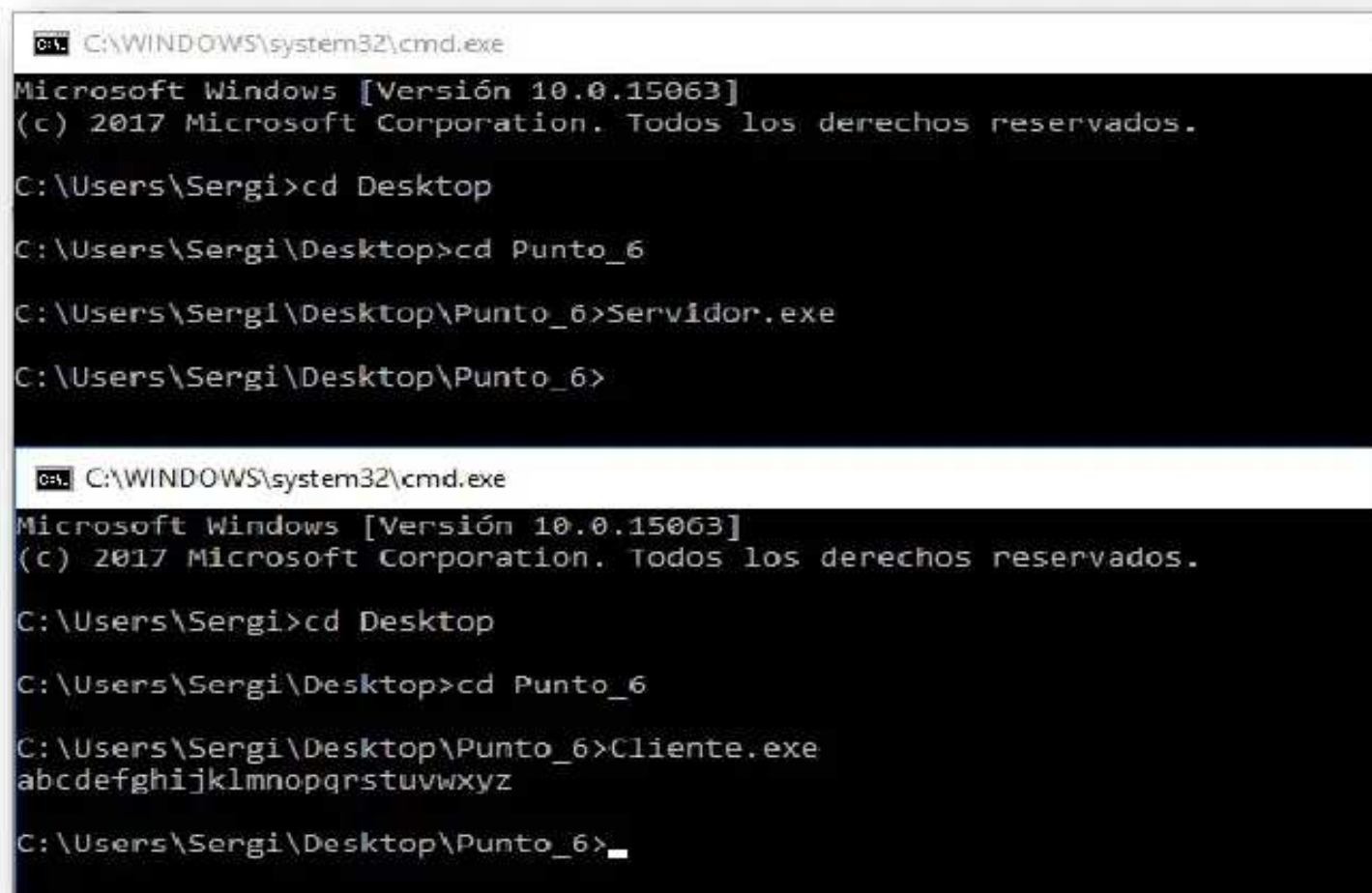
/*Programa Cliente*/
#include <windows.h>  /*Cliente de la memoria compartida*/
#include <stdio.h>
#include <unistd.h>
#define TAM_MEM 27  /*Tamaño de la memoria compartida en bytes*/

int main(void){
    HANDLE hArchMapeo;

    char *idMemCompartida = "MemoriaCompartida";
    apDatos, apTrabajo, c;
    if((hArchMapeo = OpenFileMapping(
        FILE_MAP_ALL_ACCESS, //Acceso lectura/escritura de la memoria
        compartida
        FALSE,                //No se hereda el nombre
        idMemCompartida)      //Identificador de la memoria compartida
        ) == NULL)
    {
        printf("No se abrio archivo de mapeo de la memoria compartida:
        (%i)\n", GetLastError());
        exit(-1);
    }
    if((apDatos = (char *)MapViewOfFile(hArchMapeo, //Manejador del mapeo
        FILE_MAP_ALL_ACCESS, //Permiso de lectura/escritura en la memoria
        0,
        0,
        TAM_MEM)) == NULL)
    {
        printf("No se accedio a la memoria compartida: (%i)\n",
        GetLastError());
        CloseHandle(hArchMapeo);
        exit(-1);
    }
    for(apTrabajo = apDatos; *apTrabajo != '\0'; apTrabajo++)

        putchar(*apTrabajo);
        putchar('\n');
        *apDatos = '*';
        UnmapViewOfFile(apDatos);
        CloseHandle(hArchMapeo);
    } exit(0);
}

```



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Versión 10.0.15063]
(c) 2017 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Sergi>cd Desktop
C:\Users\Sergi\Desktop>cd Punto_6
C:\Users\Sergi\Desktop\Punto_6>Servidor.exe
C:\Users\Sergi\Desktop\Punto_6>

C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Versión 10.0.15063]
(c) 2017 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Sergi>cd Desktop
C:\Users\Sergi\Desktop>cd Punto_6
C:\Users\Sergi\Desktop\Punto_6>Cliente.exe
abcdefghijklmnopqrstuvwxyz
C:\Users\Sergi\Desktop\Punto_6>_
```

Pantalla 9. Compilación en la terminal del programa 6.

Observación

En estos dos programas observamos que a partir de direcciones de memoria es posible una comunicación entre servidor y cliente, de tal modo que cada dato que se requiera se debe de tener un identificador de memoria el cual va a ser el que obtendra los datos que se requieren de hecho tambien la memoria compratida nos ayuda a que cualquier procesos que sea creado pueda acceder a cierta region de memoria que pida.

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

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>

typedef struct
{
    int shmid;
    int filas, columnas;
    float **coef;
}matriz;

matriz *crear_matriz(int filas, int columnas)
{
    int shmid,i;
    matriz *m;

    shmid=shmget(IPC_PRIVATE,sizeof(matriz)+filas*sizeof(float*))+filas*columnas*sizeof(float),IPC_CREAT|0600);
    if(shmid==-1)
    {
        perror("crear_matriz(shmget)");
        exit(-1);
    }
    if((m=(matriz*)shmat(shmid,0,0))==(matriz*)-1)
    {
        perror("crear_matriz (shmid)");
        exit(-1);
    }
    m->shmid=shmid;
    m->filas=filas;
    m->columnas=columnas;
    m->coef=(float**) &m->coef+sizeof(float**);
    for(i=0;i<filas;i++)
        m->coef[i]=(float*) &m->coef[filas]+i*columnas*sizeof(float);
    return m;
}

matriz *leer_matriz()
{
    int filas,columnas,i,j;
    matriz *m;
    printf("Filas: ");
    scanf("%d",&filas);
    printf("Columnas: ");
    scanf("%d",&columnas);
    m=crear_matriz(filas,columnas);
```



```

        for(i=0;i<filas;i++)
            for(j=0;j<columnas;j++)
                scanf("%f",&m->coef[i][j]);

        return m;
    }

matriz *opers_matrices(matriz *a,matriz *b)
{
    int semid;
    matriz *c;
    int pid;
    if((a->columnas) != (b->filas))
        return NULL;
    c=crear_matriz(a->filas,b->columnas);
    semid=semget(IPC_PRIVATE,1,IPC_CREAT|0600);
    if(semid==-1)
    {
        perror("multiplicar_matrices (semget)");
        exit(-1);
    }
    if((pid=fork())==0)
    {
        int i,j,k;
        FILE *pf;
        if((pf=fopen("matrizm","wt"))==NULL)
        {
            puts("Error en apertura");
            exit(-1);
        }
        for (i=0;i<a->filas;i++)
            for (j=0;j<b->columnas;j++)
            {
                c->coef[i][j]=0;
                for (k=0;k<b->columnas;k++)
                    c->coef[i][j]=c->coef[i][j]+a->coef[i][k]*b->coef[k][j];
            }
        for(i=0;i<a->filas;i++)
        {
            for(j=0;j<b->columnas;j++)
                fprintf(pf,"%f ",c->coef[i][j]);
            fprintf(pf,"\n");
        }
        fclose(pf);
    }
    else
    {
        int i,j,k;
        FILE *pf;
        if((a->columnas) != (b->filas))
            return NULL;
        if((pf=fopen("matrizs","wt"))==NULL)
        {
            puts("Error en apertura");
            exit(-1);
        }
    }
}

```

```

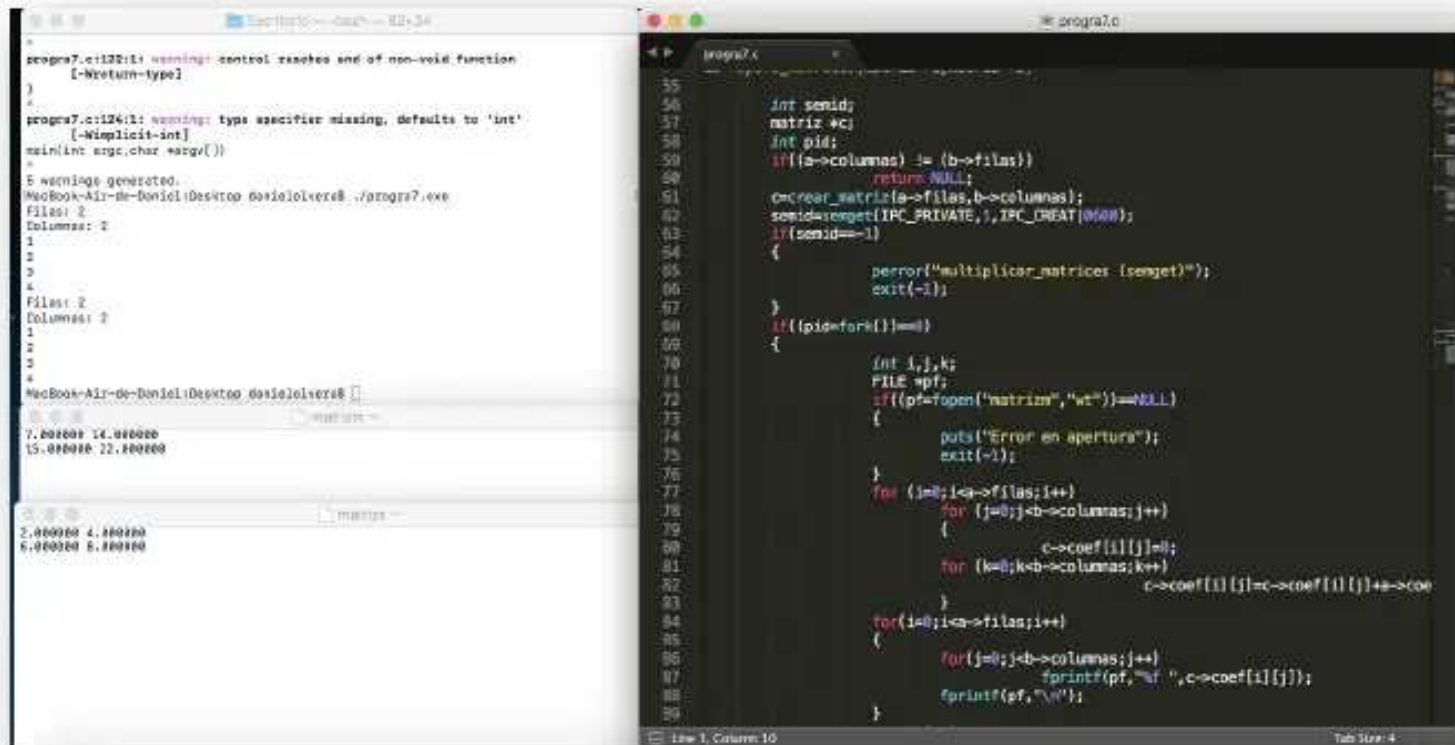
        for (i=0;i<a->filas;i++)
            for (j=0;j<b->columnas;j++)
            {
                for (k=0;k<b->columnas;k++)
                    c->coef[i][j]=a-
>coef[i][j]+b->coef[i][j];
            }
        for(i=0;i<a->filas;i++)
        {
            for(j=0;j<b->columnas;j++)
                fprintf(pf,"%f
",c->coef[i][j]);

            fprintf(pf,"\n");
        }
        fclose(pf);
    }
    return c;
}

destruir_matriz(matriz *m)
{
    shmctl(m->shmid,IPC_RMID,0);
}

main(int argc,char *argv[])
{
    int numproc;
    matriz *a, *b, *c;
    a=leer_matriz();
    b=leer_matriz();
    c=opers_matrices(a,b);
    destruir_matriz(a);
    destruir_matriz(b);
    destruir_matriz(c);
}

```



```
progra7.c:125:11: warning: control reaches end of non-void function [-Wreturn-type]
}
progra7.c:126:11: warning: type specifier missing, defaults to 'int' [-Wimplicit-int]
main(int argc, char *argv[])
^
5 warnings generated.
MacBook-Air-De-Daniel:Desktop daniel@vers8 ~ % ./progra7.exe
Files: 2
Columns: 2
1
2
3
4
Files: 2
Columns: 2
1
2
3
4
MacBook-Air-De-Daniel:Desktop daniel@vers8 ~ %
7.000000 14.000000
15.000000 22.000000

progra7.c
35
36 int sendid;
37 matriz *c;
38 int pid;
39 if((a->columns) != (b->filas))
40     return NULL;
41 crear_matriz(a->filas, b->columns);
42 sendid = get(IPC_PRIVATE, 1, IPC_CREAT | 0600);
43 if(sendid == -1)
44 {
45     perror("multiplicar matrices (sendid)");
46     exit(-1);
47 }
48 if((pid = fork()) == 0)
49 {
50     int i, j, k;
51     FILE *pf;
52     if((pf = fopen("matrizm", "wt")) == NULL)
53     {
54         puts("Error en apertura");
55         exit(-1);
56     }
57     for (i = 0; i < a->filas; i++)
58     {
59         for (j = 0; j < b->columns; j++)
60         {
61             c->coef[i][j] = 0;
62             for (k = 0; k < b->columns; k++)
63                 c->coef[i][j] = c->coef[i][j] + a->coef[i][k] * b->coef[k][j];
64         }
65     }
66     for (i = 0; i < a->filas; i++)
67     {
68         for (j = 0; j < b->columns; j++)
69             fprintf(pf, "%f ", c->coef[i][j]);
70         fprintf(pf, "\n");
71     }
72 }
```

Pantalla 10. Compilación en la terminal del programa 7.

Sección Windows

```
/*Programa Servidor*/
#include <windows.h> /*Cliente de la memoria compartida*/
#include <stdio.h>
#include <unistd.h>
#define TAM_MEM 27 /*Tamaño de la memoria compartida en bytes*/

int Matriz_Resultado_Suma[10][10];
int Matriz_Resultado_Multiplicacion[10][10];
int const Tam=100;
void Inversa();
void PideDatos(int k,int Dim, float Sist[][Tam]);
void Invierte(int Dim, float Sist[][Tam], float Inv[][Tam]);

void Inversa(){
    int Dimension=10,k=1;
    FILE *fichero;
    int i,j;
    float Sistema[Tam][Tam],Inversa[Tam][Tam];
    PideDatos(k,Dimension,Sistema);
    Invierte(Dimension,Sistema,Inversa);
    fichero = fopen("Resultado.txt","w");

    for(i=0;i<10;i++)
        for(j=0;j<10;j++)
        {
            fprintf(fichero,"%1f\n",Inversa[i][j]);
        }
    fclose(fichero);
    printf("El resultado ha sido almacenado en el archivo Resultado\n");
    return;
}

void PideDatos(int k,int Dim,float Sist[][Tam]){
    int i,j;
    for(i=1;i<=Dim;i++){
        for(j=1;j<=Dim;j++){
            Sist[i][j] = Matriz_Resultado_Multiplicacion[i-1][j-1];
        }
    }
}

void Invierte(int Dim, float Sist[][Tam], float Inv[][Tam]){
    int NoCero,Col,C1,C2,A;
    float Pivote,V1,V2;

    /*Se inicializa la matriz inversa, como la matriz identidad:*/
    for(C1=1;C1<=Dim;C1++) for(C2=1;C2<=Dim;C2++)
        if (C1==C2) Inv[C1][C2]=1; else Inv[C1][C2]=0;

    for(Col=1;Col<=Dim;Col++){
        NoCero=0;A=Col;
        while(NoCero==0){
            if((Sist[A][Col]>0.000001)||((Sist[A][Col]<-0.000001))){
                NoCero=1;
            }
        }
    }
}
```

```

        else A++;}
Pivote=Sist[A][Col];
for(C1=1;C1<=Dim;C1++){
    V1=Sist[A][C1];
    Sist[A][C1]=Sist[Col][C1];
    Sist[Col][C1]=V1/Pivote;
    V2=Inv[A][C1];
    Inv[A][C1]=Inv[Col][C1];
}
Inv[Col][C1]=V2/Pivote;
for(C2=Col+1;C2<=Dim;C2++){
    V1=Sist[C2][Col];
    for(C1=1;C1<=Dim;C1++){
        Sist[C2][C1]=Sist[C2][C1]-V1*Sist[Col][C1];
        Inv[C2][C1]=Inv[C2][C1]-V1*Inv[Col][C1];}
    }
}

/*Aqui ya esta triangularizada, con 1s en diagonal, ahora se
diagonaliza*/
for(Col=Dim;Col>=1;Col--) for(C1=(Col-1);C1>=1;C1--)
{
    V1=Sist[C1][Col];
    for(C2=1;C2<=Dim;C2++){
        Sist[C1][C2]=Sist[C1][C2]-V1*Sist[Col][C2];
        Inv[C1][C2]=Inv[C1][C2]-V1*Inv[Col][C2];
    }
}
}

int main(void){
HANDLE hArchMapeo, hArchMapeo2, hArchMapeo5;
char * idMemCompartida = "MemoriaCompartida";
char * idMemCompartida2 = "MemoriaCompartida2";
char * idMemCompartida5 = "MemoriaCompartida5";
char * apDatos, *apTrabajo, c, *apDatos1, *apTrabajo1;
char *apDatos5, *apTrabajo5;
char MatrizAux[10][10], MatrizAux2[10][10];

int contador5=0, contador2=0, contador3 = 0, contador4 = 0;
char Matriz_A[10][10]={
    {'1','2','3','4','5','1','2','3','4','5'},
    {'7','5','5','3','3','5','4','3','2','1'},
    {'3','5','4','2','2','1','4','7','6','7'},
    {'2','4','5','8','7','1','2','6','5','4'},
    {'8','8','1','2','7','5','1','5','5','1'},
    {'5','6','7','5','6','4','2','4','9','5'},
    {'6','4','9','8','6','2','5','6','5','3'},
    {'3','5','6','2','8','4','3','4','8','1'},
    {'1','7','2','4','8','4','9','5','6','3'},
    {'6','1','4','5','8','8','8','5','7','4'},
};

char Matriz_B[10][10]={
    {'6','4','2','5','6','3','2','6','5','7'},
    {'2','3','6','1','4','2','1','4','2','5'},

```

```

        {'4','8','7','4','3','7','2','6','3','3'},
        {'8','6','7','6','4','7','8','4','1','2'},
        {'1','3','9','4','6','8','8','8','6','9'},
        {'7','4','8','5','3','8','6','9','4','5'},
        {'8','7','5','7','4','8','2','2','9','8'},
        {'1','2','6','8','8','2','4','6','2','8'},
        {'9','2','6','1','3','7','2','3','4','1'},
        {'5','3','5','3','4','2','4','8','2','5'},

//Envio de datos al cliente
if((hArchMapeo = CreateFileMapping(
    INVALID_HANDLE_VALUE, //usa la memoria compartida
    NULL, //seguridad por default
    PAGE_READWRITE, //acceso lectura/escritura a la memoria
    0, //tamaño maximo parte alta de un DWORD
    TAM_MEM, //tamaño maximo parte baja de un DWORD
    idMemCompartida) //Identificador de la memoria compartida
) == NULL)
{
    printf("No se mapeo la memoria compartida: (%i)\n", GetLastError());
    exit(-1);
}

//Manejador de mapeo para envio de datos al cliente
if((apDatos = (char *)MapViewOfFile(hArchMapeo,
    FILE_MAP_ALL_ACCESS, //Permiso de lectura/escritura en la memoria
    0,
    0,
    TAM_MEM)) == NULL)
{
    printf("No se accedio a la memoria compartida: (%i)\n", GetLastError());
    CloseHandle(hArchMapeo);
    exit(-1);
}

//Enviar dos matrices al cliente
apTrabajo = apDatos;
for(int i = 0 ; i < 10; i++){
    for(int j = 0; j < 10; j++)
        *apTrabajo++ = Matriz_A[i][j];
}
for(int i = 0 ; i < 10; i++){
    for(int j = 0; j < 10; j++)
        *apTrabajo++ = Matriz_B[i][j];
}
*apTrabajo = '\0';
while(*apDatos != '*')
    sleep(1);
UnmapViewOfFile(apDatos);
CloseHandle(hArchMapeo);

//Recive la matriz multiplicacion desde el cliente

if((hArchMapeo2 = OpenFileMapping(
    FILE_MAP_ALL_ACCESS, //Acceso lectura/escritura de la memoria compartida
    FALSE, //No se hereda el nombre
    idMemCompartida2) //Identificador de la memoria compartida

```

```

        ) == NULL)
    {
        printf("No se abrio archivo de mapeo de la memoria compartida: (%i)\n", GetLastError());
        exit(-1);
    }

    if((apDatos1 = (char *)MapViewOfFile(hArchMapeo2, //Manejador del mapeo
        FILE_MAP_ALL_ACCESS, //Permiso de lectura/escritura en la memoria
        0,
        0,
        TAM_MEM)) == NULL)
    {
        printf("No se accedio a la memoria compartida: (%i)\n", GetLastError());
        CloseHandle(hArchMapeo2);
        exit(-1);
    }

    for(apTrabajo1 = apDatos1; *apTrabajo1 != '\0'; apTrabajo1++){
        //obteniendo los datos de la matriz resultado
        if(contador < 10){
            Matriz_Resultado_Multiplicacion[contador][contador2++]
                = *apTrabajo1;

            if(contador2 == 10){
                contador += 1;
                contador2 = 0;
            }
        }
    }

    for(int i = 0; i < 10; i++){
        printf("\n");
        for(int j = 0; j < 10; j++){
            printf("%d ", Matriz_Resultado_Multiplicacion[i][j]);
        }
        for(int i = 0; i < 10; i++){
            for(int j = 0; j < 10; j++){
                MatrizAux[i][j] = Matriz_A[i][j] - 48;
            }
        }
        for(int i = 0; i < 10; i++){
            for(int j = 0; j < 10; j++){
                MatrizAux2[i][j] = Matriz_B[i][j] - 48;
            }
        }
        for(int i = 0; i < 10; i++){
            for(int j = 0; j < 10; j++){
                Matriz_Resultado_Suma[i][j] = 0;
                for(int k = 0; k < 10; k++){
                    Matriz_Resultado_Suma[i][j] =
                        Matriz_Resultado_Suma[i][j] +
                        (MatrizAux[i][k] * MatrizAux2[k][j]);
                }
            }
        }

        for(int i = 0; i < 10; i++){
            printf("\n");
        }
    }

```

```

        for(int j = 0; j < 10; j++)
            printf("%d ", Matriz_Resultado_Suma[i][j]);
    }
    *apDatos1 = '*';
    UnmapViewOfFile(apDatos1);
    CloseHandle(hArchMapeo2);
    printf("\n\n");
    Inversa(Matriz_Resultado_Multiplicacion, Matriz_Resultado_Suma);
}
exit(0);

```



```

/*Programa Cliente*/
#include <windows.h>  /*Cliente de la memoria compartida*/
#include <stdio.h>
#include <unistd.h>
#define TAM_MEM 27  /*Tamaño de la memoria compartida en bytes*/

int main(void){
    HANDLE hArchMapeo, hArchMapeo1, hArchMapeo2;

    HANDLE hArchMapeo3, hArchMapeo4, hArchMapeo5;
    char *idMemCompartida = "MemoriaCompartida";
    char *idMemCompartida1 = "MemoriaCompartida1";
    char *idMemCompartida2 = "MemoriaCompartida2";
    char *idMemCompartida3 = "MemoriaCompartida3";
    char *idMemCompartida4 = "MemoriaCompartida4";
    char *idMemCompartida5 = "MemoriaCompartida5";
    char *apDatos, *apTrabajo, c, *apDatos1, *apTrabajo1, *apDatos2;
    char *apTrabajo2, *apDatos3, *apTrabajo3, *apDatos4, *apTrabajo4;
    char *apDatos5, *apTrabajo5;
    int Matriz_A[10][10];
    int Matriz_B[10][10];
    int Matriz_Resultado[10][10];
    int Matriz_Resultado1[10][10];
    int Matriz_Aux[10][10];
    int contador = 0, contador2 = 0, contador3 = 0, contador4 = 0;
    int contador7 = 0, contador8 = 0, contador5 = 0, contador6 = 0;
    if((hArchMapeo = OpenFileMapping(
        FILE_MAP_ALL_ACCESS,  //Acceso lectura/escritura de la memoria compartida
        FALSE,                //No se hereda el nombre
        idMemCompartida)      //Identificador de la memoria compartida
    ) == NULL)
    {
        printf("No se abrio archivo de mapeo de la memoria compartida:
        (%i)\n", GetLastError());
        exit(-1);
    }

    if((hArchMapeo2 = CreateFileMapping(
        INVALID_HANDLE_VALUE, //usa la maemoria compartida
        NULL,                  //seguridad por dfault
        PAGE_READWRITE,       //acceso lectura/escritura a la memoria
        0,                     //tamaño máximo parte alta de un DWORD
        TAM_MEM,               //tamaño maximo parte baja de un DWORD
        idMemCompartida2)      //Identificador de la memoria compartida
    ) == NULL)
    {
        printf("No se mapeo la memoria compartida: (%i)\n", GetLastError());
        exit(-1);
    }

    if((apDatos = (char *)MapViewOfFile(hArchMapeo, //Manejador del mapeo
        FILE_MAP_ALL_ACCESS, //Permiso de lectura/escritura en la memoria
        0,
        0,
        TAM_MEM)) == NULL)
    {
        printf("No se accedio a la memoria compartida: (%i)\n", GetLastError());
        CloseHandle(hArchMapeo);
    }
}

```

```

        exit(-1);
    }

    if((apDatos2 = (char *)MapViewOfFile(hArchMapeo2, //Manejador del mapeo
        FILE_MAP_ALL_ACCESS, //Permiso de lectura/escritura en la memoria
        0,
        0,
        TAM_MEM)) == NULL)

printf("No se accedio a la memoria compartida: (%i)\n", GetLastError());
    CloseHandle(hArchMapeo2);
    exit(-1);
}

for(apTrabajo = apDatos; *apTrabajo != '\0'; apTrabajo++){
    //obteniendo los datos de la Matriz_A
    if(contador < 10){
        Matriz_A[contador][contador2++] = *apTrabajo - 48;
        if(contador2 == 10){
            contador += 1;
            contador2 = 0;
        }
    }
    //Obteniendo los datos de la Matriz_B
    else if(contador >= 10){
        Matriz_B[contador3][contador4++] = *apTrabajo - 48;
        if(contador4 == 10){
            contador3 +=1;
            contador4 = 0;
        }
    }
}

//Multiplicacion de la Matriz_A y Matriz_B
for(int i = 0; i < 10; i++){
    for(int j = 0; j < 10; j++){
        Matriz_Resultado[i][j] = 0;
        for(int k = 0; k < 10; k++)
            Matriz_Resultado[i][j] = Matriz_Resultado[i][j] +
(Matriz_A[i][k] * Matriz_B[k][j]);
    }
}

printf("Multiplicacion de Matrices");
for(int i = 0; i < 10; i++){
    printf("\n");
    for(int j = 0; j < 10; j++)
        printf("%d ", Matriz_Resultado[i][j]);
}

*apDatos = '*';
UnmapViewOfFile(apDatos);
CloseHandle(hArchMapeo);

apTrabajo2 = apDatos2;
for(int i = 0 ; i < 10; i++){
    for(int j = 0; j < 10; j++)
        *apTrabajo2++ = Matriz_A[i][j] + Matriz_B[i][j];
}
for(int i = 0 ; i < 10; i++){

```

```

        for(int j = 0; j < 10; j++)
            *apTrabajo2++ = Matriz_A[i][j];
    }
    for(int i = 0 ; i < 10; i++){
        for(int j = 0; j < 10; j++)
            *apTrabajo2++ = Matriz_B[i][j];
    }
    *apTrabajo2 = '\0';

    while(*apDatos2 != '*')
        sleep(1);
    UnmapViewOfFile(apDatos2);
    CloseHandle(hArchMapeo2);

    //Mapeo para memoria compartida Cliente Nieto
    if((hArchMapeo1 = CreateFileMapping(
        INVALID_HANDLE_VALUE, //usa la maemoria compartida
        NULL, //seguridad por dfault
        PAGE_READWRITE, //acceso lectura/escritura a la memoria
        0, //tamaño maximo parte alta de un DWORD
        TAM_MEM, //tamaño maximo parte baja de un DWORD
        idMemCompartida1) //Identificador de la memoria compartida
    ) == NULL)
    {
        printf("No se mapeo la memoria compartida: (%i)\n", GetLastError());
        exit(-1);
    }

    if((apDatos1 = (char *)MapViewOfFile(hArchMapeo1, //Manejador del mapeo
        FILE_MAP_ALL_ACCESS, //Permiso de lectura/escritura en la memoria
        0,
        0,
        TAM_MEM)) == NULL)
    {
        printf("No se accedio a la memoria compartida: (%i)\n", GetLastError());
        CloseHandle(hArchMapeo1);
        exit(-1);
    }

    apTrabajo1 = apDatos1;

    for(int i = 0 ; i < 10; i++){
        for(int j = 0; j < 10; j++)
            *apTrabajo1++ = Matriz_A[i][j];
    }
    for(int i = 0 ; i < 10; i++){
        for(int j = 0; j < 10; j++)
            *apTrabajo1++ = Matriz_B[i][j];
    }
    *apTrabajo1 = '\0';
    while(*apDatos1 != '*')
        sleep(1);
    UnmapViewOfFile(apDatos1);
    CloseHandle(hArchMapeo1);

    if((hArchMapeo4 = OpenFileMapping(
        FILE_MAP_ALL_ACCESS, //Acceso lectura/escritura de la memoria compartida
        FALSE, //No se hereda el nombre
        idMemCompartida4) //Identificador de la memoria compartida

```

```

        ) == NULL)
    {
        printf("No se abrio archivo de mapeo de la memoria compartida:
(%i)\n", GetLastError());
        exit(-1);
    }

    if((apDatos4 = (char *)MapViewOfFile(hArchMapeo4, //Manejador del mapeo
        FILE_MAP_ALL_ACCESS, //Permiso de lectura/escritura en la memoria
        0,
        0,
        TAM_MEM)) == NULL)
    {
        printf("No se accedio a la memoria compartida: (%i)\n", GetLastError());
        CloseHandle(hArchMapeo4);
        exit(-1);
    }

    for(apTrabajo4 = apDatos4; *apTrabajo4 != '\0'; apTrabajo4++){
        //obteniendo los datos de la Matriz_A
        if(contador7 < 10){
            Matriz_Resultado1[contador7][contador8++] = *apTrabajo4;
            if(contador8 == 10){
                contador7 += 1;
                contador8 = 0;
            }
        }
    }
    *apDatos4 = '*';
    UnmapViewOfFile(apDatos4);
    CloseHandle(hArchMapeo4);
    exit(0);
}

```

```

/*Programa Cliente Nieto*/
#include <windows.h>  /*Cliente de la memoria compartida*/
#include <stdio.h>
#include <unistd.h>
#define TAM_MEM 27  /*Tamaño de la memoria compartida en bytes*/

int main(void){
    HANDLE hArchMapeo, hArchMapeo4;

    char *idMemCompartida="MemoriaCompartida1";
    char *idMemCompartida4="MemoriaCompartida4";
    char *apDatos, *apTrabajo, c, *apDatos4, *apTrabajo4;
    int Matriz_A[10][10];
    int Matriz_B[10][10];
    int Matriz_Resultado[10][10];
    int contador = 0, contador2 = 0, contador3 = 0, contador4 = 0;
    if((hArchMapeo = OpenFileMapping(
        FILE_MAP_ALL_ACCESS, //Acceso lectura/escritura de la memoria compartida
        FALSE,                //No se hereda el nombre
        idMemCompartida)      //Identificador de la memoria compartida
    ) == NULL)
    {
        printf("No se abrio archivo de mapeo de la memoria compartida:
        (%i)\n", GetLastError());
        exit(-1);
    }

    if((apDatos = (char *)MapViewOfFile(hArchMapeo, //Manejador del mapeo
        FILE_MAP_ALL_ACCESS, //Permiso de lectura/escritura en la memoria
        0,
        0,
        TAM_MEM)) == NULL)
    {
        printf("No se accedio a la memoria compartida: (%i)\n",
        GetLastError());
        CloseHandle(hArchMapeo);
        exit(-1);
    }

    for(apTrabajo = apDatos; *apTrabajo != '\0'; apTrabajo++){
        //obteniendo los datos de la Matriz_A
        if(contador < 10){
            Matriz_A[contador][contador2++] = *apTrabajo;
            if(contador2 == 10){
                contador += 1;
                contador2 = 0;
            }
        }
        //Obteniendo los datos de la Matriz_B
        else if(contador >= 10){
            Matriz_B[contador3][contador4++] = *apTrabajo;
            if(contador4 == 10){
                contador3 +=1;
                contador4 = 0;
            }
        }
    }
    //Suma de la Matriz_A y Matriz_B

```

```

    for(int i = 0; i < 10; i++){
        for(int j = 0; j < 10; j++){
            Matriz_Resultado[i][j] = Matriz_B[i][j] + Matriz_A[i][j];
        }
    }
    //Resultado de la suma
    printf("Suma de Matrices");
    for(int i = 0; i < 10; i++){
        printf("\n");
        for(int j = 0; j < 10; j++){
            printf("%d ", Matriz_Resultado[i][j]);
        }

        *apDatos = '*';
        UnmapViewOfFile(apDatos);
        CloseHandle(hArchMapeo);

        if((hArchMapeo4 = CreateFileMapping(
            INVALID_HANDLE_VALUE, //usa la memoria compartida
            NULL, //seguridad por dfault
            PAGE_READWRITE, //acceso lectura/escritura a la memoria
            0, //tamaño máximo parte alta de un DWORD
            TAM_MEM, //tamaño máximo parte baja de un DWORD
            idMemCompartida4) //Identificador de la memoria compartida
        ) == NULL)
        {
            printf("No se mapeo la memoria compartida: (%i)\n", GetLastError());
            exit(-1);
        }

        if((apDatos4 = (char *)MapViewOfFile(hArchMapeo4, //Manejador del mapeo
            FILE_MAP_ALL_ACCESS, //Permiso de lectura/escritura en la memoria
            0,
            0,
            TAM_MEM)) == NULL)
        {
            printf("No se accedio a la memoria compartida: (%i)\n",
                GetLastError());
            CloseHandle(hArchMapeo4);
            exit(-1);
        }

        apTrabajo4 = apDatos4;
        for(int i = 0 ; i < 10; i++){
            for(int j = 0; j < 10; j++){
                *apTrabajo4++ = Matriz_Resultado[i][j];
            }
        }
        *apTrabajo4 = '\0';
        while(*apDatos4 != '*')
            sleep(1);
        UnmapViewOfFile(apDatos4);
        CloseHandle(hArchMapeo4);
        exit(0);
    }
}

```

```

C:\Users\Sergi\Desktop\Punto_7>Servidor.exe
7 6 5 9 11 4 4 9 9 12
9 8 11 4 7 7 5 7 4 6
7 33 11 6 5 8 6 13 9 10
10 10 12 14 11 8 10 10 6 6
9 11 10 6 13 13 9 13 11 10
12 10 15 10 9 12 8 13 13 10
14 11 14 15 10 10 7 8 14 11
4 7 12 10 16 6 7 10 10 9
10 9 8 5 11 11 11 8 10 4
11 4 9 8 12 10 12 13 9 9

146 120 193 125 136 164 132 171 106 153
192 171 222 172 172 205 135 213 153 210
197 156 237 172 191 187 133 219 143 214
205 181 286 194 207 240 198 242 145 222
189 147 258 172 211 212 164 250 164 249
270 213 329 283 230 280 295 187 252
270 248 331 249 248 302 214 289 200 277
209 175 288 172 196 261 169 244 170 229
242 199 316 214 222 285 195 252 214 281
310 237 349 264 253 342 239 319 247 312
El resultado ha sido almacenado en el archivo Resultado

C:\Users\Sergi\Desktop\Punto_7>Cliente.exe
Multiplicacion de Matrices
146 120 193 125 136 164 132 171 106 153
192 171 222 172 172 205 135 213 153 210
197 156 237 172 191 187 133 219 143 214
205 181 286 194 207 240 198 242 145 222
189 147 258 172 211 212 164 250 164 249
270 213 329 283 230 280 295 187 252
270 248 331 249 248 302 214 289 200 277
209 175 288 172 196 261 169 244 170 229
242 199 316 214 222 285 195 252 214 281
310 237 349 264 253 342 239 319 247 312
C:\Users\Sergi\Desktop\Punto_7>

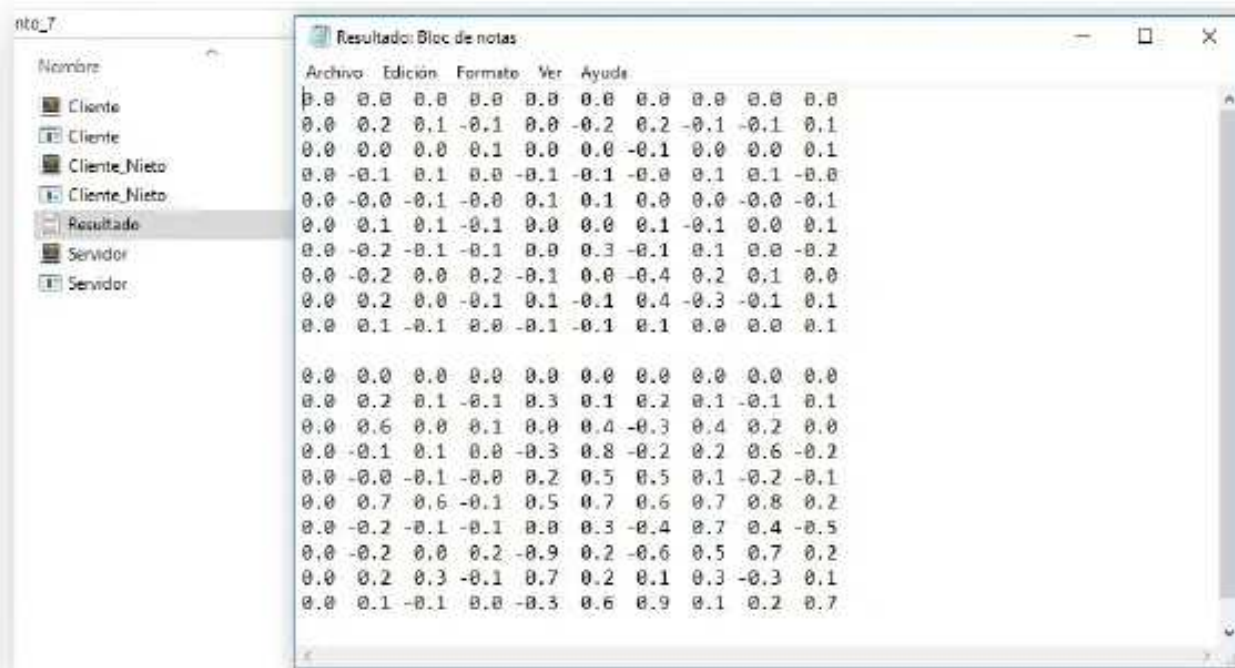
C:\Users\Sergi\Desktop\Punto_7>Cliente_Nieto.exe
Suma de Matrices
7 6 5 9 11 4 4 9 9 12
9 8 11 4 7 7 5 7 4 6
7 13 11 6 5 8 6 13 9 10
10 10 12 14 11 8 10 10 6 6
9 11 10 6 13 13 9 13 11 10
12 10 15 10 9 12 8 13 13 10
14 11 14 15 10 10 7 8 14 11
4 7 12 10 16 6 7 10 10 9
10 9 8 5 11 11 11 8 10 4
11 4 9 8 12 10 12 13 9 9
C:\Users\Sergi\Desktop\Punto_7>

```

Pantalla 11. Compilación en la terminal del programa 7 sección windows.

El resultado de las matrices inversas se guarda en un archivo llamado resultado.txt

Aquí se muestran los resultados de las matrices inversas:



Pantalla 12. Resultados de la compilación del programa 7 sección windows.

Análisis critico

Leonardo Daniel Olvera Aguila

La diferencia entre el desarrollo de aplicaciones utilizando los recursos de tuberías y memoria compartida son sumamente diferentes en Linux y Windows, resultando ser más sencillo realizarlo en el primero. el funcionamiento es igual de beneficiosos para las aplicaciones en ambos sistemas operativos. es importante el uso continuo de estos recursos en nuestras aplicaciones, lo cual mejorará nuestro conocimiento con respecto a sus diferentes usos, los cuales son amplios todavía.

Mendoza Parra Sergio

Durante el proceso de desarrollo de la práctica, fue fácil notar que es de suma importancia la aplicación de tuberías en las diferentes aplicaciones que sean desarrolladas con base en diferentes procesos, facilitando así la comunicación de estos y el envío de diferentes datos a través de las mismas tuberías, como lo es también en combinación con la memoria compartida, se mejoraría notablemente esta interacción entre los diferentes datos que usa todo el programa y los diferentes procesos.

Paz Sánchez Brandon

La mejora del programa gracias a las tuberías dejo también un poco de conocimiento extra, debido a que en el proceso de desarrollo de las aplicaciones se encontraron detalles acerca de su uso y su comportamiento, erradicando en algunas ocasiones dudas que surgieron al inicio de la práctica durante la observación de los diferentes programas ejemplo. Es importante empezar a hacer uso de estos recursos en nuestras diferentes aplicaciones.

Conclusiones

Leonardo Daniel Olvera Aguila

Con el desarrollo de los ejemplos y la aplicación que se desarrolló en esta práctica podemos darnos cuenta de la importancia que tiene el utilizar tuberías para que los procesos puedan comunicarse y realizar las tareas que se le asignan a cada uno de ellos. El trabajar con tuberías y memoria compartida en esta práctica nos ha ayudado a entender mejor cómo los procesos se pueden comunicar y las técnicas que existen, además de implementar una aplicación para cada caso, así como de llegar a la conclusión de que en qué caso o situación se puede o es necesario usar alguna de estas técnicas, que finalmente hacen que los procesos que se crean se comuniquen de diferentes formas y hagan la o las tareas que se les asignó realizar. Sobre todo, es muy importante saber que funciones debemos utilizar en cada caso y por ende ya debemos saber para qué es cada una de ellas, y cuáles de ellas funcionan para cada uno de los dos sistemas operativos que hemos ido trabajando, que son Linux y Windows.

Mendoza Parra Sergio

La implementación de la memoria compartida me parece que es una de las mejores formas de comunicación entre procesos ya que no es difícil su implementación, las líneas de código no son tantas y por la forma de cómo tratar a la memoria podemos usar lógica de apuntadores para movernos dentro del arreglo.

Paz Sánchez Brandon

Esta práctica en su momento me pareció un poco compleja dado que nunca había visto este tema, pero al investigar de como funciona las tuberías y el uso de la memoria compartida entiendo que estas dos formas de creación de procesos con soporte de comunicación es más completa para la programación concurrente. Por otro lado, note que las tuberías son unidireccionales y que dependiendo si son de nombre o sin nombre no es difícil codificarlas.