****

**Práctica 6. Comunicación Interprocesos (IPC) en Linux y Windows**

**Grupo: 2CM9**

**Equipo:**

* **Guerra Vargas Irving Cristóbal Colón**
* **Reyes Valenzuela Alejandro**
* **Dávila Méndez Juan Gabriel**

**Competencia.**

* **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.

|  |  |
| --- | --- |
| **Llamada al Sistema** | **Descripción / Uso** |
| **int pipe(int pipefd[2]);** | Crea una tubería, un canal de datos unidireccional que se puede usar para la comunicación entre procesos. La matriz pipefd se usa para regresar dos descriptores de archivo que hacen referencia a los extremos de la tubería. pipefd [0] se refiere al final de lectura de la tubería. pipefd [1] se refiere a la escritura final de la tubería. En caso de éxito, se devuelve cero. En caso de error, se devuelve -1. |
| **int shmget(key\_t *key*, size\_t *size*, int *shmflg*);** | Devuelve el identificador del segmento de memoria compartida asociado con el valor de la clave de argumento. Se crea un nuevo segmento de memoria compartida, con un tamaño igual al valor del tamaño redondeado a un múltiplo de PAGE\_SIZE, si key tiene el valor IPC\_PRIVATE o key no es IPC\_PRIVATE, no existe un segmento de memoria compartida correspondiente a la clave, y se especifica IPC\_CREAT en shmflg. |
| **void \*shmat(int *shmid*, const void \**shmaddr*, int *shmflg*);** | La función shmat () une el segmento de memoria compartida asociado con el identificador de memoria compartida especificado por shmid en el espacio de direcciones del proceso de llamada.  El segmento se adjunta a la dirección especificada por uno de los siguientes criterios:  Si shmaddr es un puntero nulo, el segmento se adjunta a la primera dirección disponible seleccionada por el sistema.  Si shmaddr no es un puntero nulo y (shmflg & SHM\_RND) no es cero, el segmento se adjunta a la dirección dada por (shmaddr - ((uintptr\_t) shmaddr% SHMLBA)). El carácter '%' es el operador de resto de lenguaje C.  Si shmaddr no es un puntero nulo y (shmflg & SHM\_RND) es 0, el segmento se adjunta a la dirección dada por shmaddr. El segmento se adjunta para leer si (shmflg y SHM\_RDONLY) es distinto de cero y el proceso de llamada tiene permiso de lectura; de lo contrario, si es 0 y el proceso de llamada tiene permiso de lectura y escritura, el segmento se adjunta para lectura y escritura.  Una vez completada con éxito, shmat () incrementará el valor de shm\_nattch en la estructura de datos asociada con la ID de memoria compartida del segmento de memoria compartida conectado y devolverá la dirección de inicio del segmento. De lo contrario, el segmento de memoria compartida no se adjuntará, shmat () devolverá -1. |

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,sizeogf(bufer));

printf("Se recibió; %s\n",bufer);

}

}

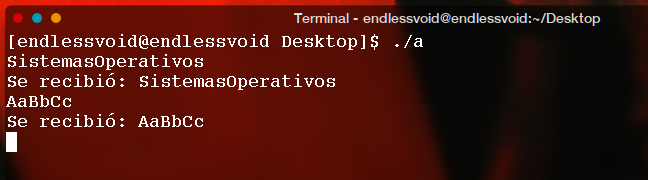
while(VALOR){

gets(bufer);

write(desc\_arch[1],bufer,strlen(bufer)+1);

}

}



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

***(Las capturas se encuentran en la sección de Windows)***

#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};

/\* Obtención de información para la inicialización del proceso hijo \*/

GetStartupInfo (&siHijo);

/\* Creación de la tubería sin nombre \*/

CreatePipe (&hLecturaPipe, &hEscrituraPipe, &pipeSeg, 0);

/\* Escritura en la tubería 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, /\* Hereda el proceso hijo los manejadores de la tubería del padre \*/

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.c \*/

#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 desde 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;

}

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 10 x 10 a multiplicar por parte del hijo, mientras tanto el proceso hijo creará un hijo de él, al cual enviará dos matrices de 10 x 10 a sumar en el proceso hijo creado, nuevamente el 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.

***(Se encuentra más adelante).***

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 c;

int shmid;

key\_t llave;

char \*shm, \*s;

llave = 5678;

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)) == (char \*) -1) {

perror("Error al enlazar la memoria compartida: shmat");

exit(-1);

}

s = shm;

for (c = 'a'; c <= 'z'; c++)

\*s++ = c;

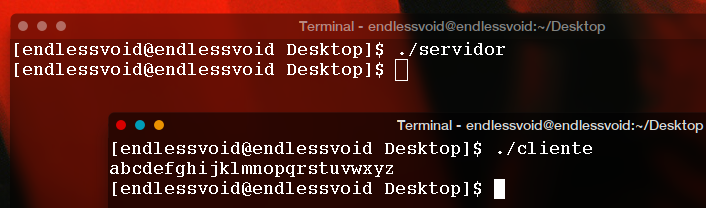
\*s = '\0';

while (\*shm != '\*')

sleep(1);

exit(0);

}



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

#include <windows.h> /\* Cliente de la memoria compartida \*/

#include <stdio.h>

#define TAM\_MEM 27 /\*Tamaño de la memoria compartida en bytes \*/

int main(void)

{

HANDLE hArchMapeo;

char \*idMemCompartida = "MemoriaCompatida";

char \*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);

}

#include <windows.h> /\* Servidor de la memoria compartida \*/

#include <stdio.h> /\* (ejecutar el servidor antes de ejecutar el cliente)\*/

#define TAM\_MEM 27 /\*Tamaño de la memoria compartida en bytes \*/

int main(void)

{

HANDLE hArchMapeo;

char \*idMemCompartida = "MemoriaCompatida";

char \*apDatos, \*apTrabajo, c;

if((hArchMapeo=CreateFileMapping(

INVALID\_HANDLE\_VALUE, // usa memoria compartida

NULL, // seguridad por default

PAGE\_READWRITE, // acceso lectura/escritura a la memoria

0, // tamaño maxixmo parte alta de un DWORD

TAM\_MEM, // tamaño maxixmo 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 creo 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);

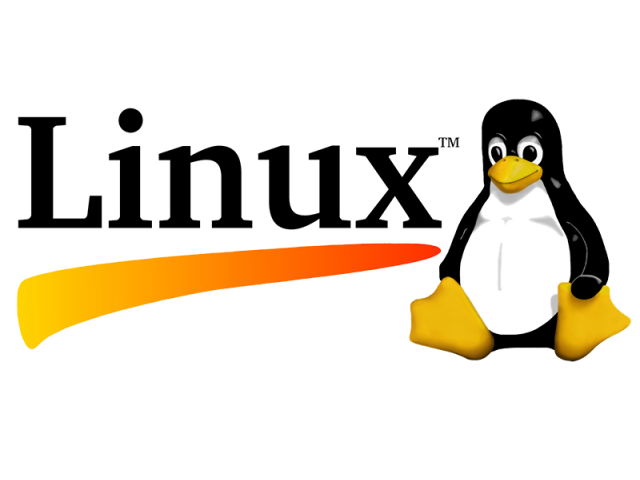
}

***(Las capturas se encuentran en la sección de Windows).***

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 para Linux como para Windows utilizando la memoria compartida de cada sistema operativo.

***(Se encuentra más adelante).***

**SECCIÓN LINUX**

****

**Programa del punto 4 para linux (utilizando tuberías).**

NOTA: Se usaron matrices 3x3 debido a que es tedioso introducir 200 valores y que además las sumas y multiplicaciones de estas sean linealmente independientes. Sin embargo, se probaron los programas con matrices de 10x10, aunque estas no son linealmente independientes, con el objetivo de verificar la suma y multiplicación.

NOTA 2: El archivo donde se encuentran todas las funciones implementadas (shmget, pipe, write, etc…) se encuentra antes de la sección para Windows.

**Matrices de 3x3 utilizadas (ambas son linealmente independientes, la primera se usó para multiplicarse consigo misma, así como la segunda, pero sumada dos veces):**

1 2 3 1 2 3

4 5 6 4 5 6

7 9 8 7 8 5

**Programa que suma y multiplica utilizando tuberías:**

#include <stdlib.h>

#include <stdio.h> //El hijo multiplica y el nieto las suma

#include "operadores.h"

#define SIZ\_MAX 3

int main()

{

float matrizA[SIZ\_MAX][SIZ\_MAX],matrizB[SIZ\_MAX][SIZ\_MAX],InvA[SIZ\_MAX][SIZ\_MAX],InvB[SIZ\_MAX][SIZ\_MAX];

int matrA[2],matrB[2], resSSum[2], resMMult[2];

printf("Inserte datos (A):\n");

InsEnMat(matrizA,SIZ\_MAX,SIZ\_MAX); //Dialogos para insertar MatrizA

printf("Inserte datos (B):\n");

InsEnMat(matrizB,SIZ\_MAX,SIZ\_MAX); //Dialogos para insertar MatrizA

pipe(matrA); pipe(matrB); pipe(resSSum); pipe (resMMult);

printf("Las dos matrices se comparten.\n");

Matriz2Tuberia(matrizA,matrA);

Matriz2Tuberia(matrizB,matrB);

if(fork()==0) //HIJO 1

{

float resMult[SIZ\_MAX][SIZ\_MAX],matrizAHijo[SIZ\_MAX][SIZ\_MAX],matrizBHijo[SIZ\_MAX][SIZ\_MAX];

Tuberia2Matriz(matrA,matrizAHijo);

Tuberia2Matriz(matrB,matrizBHijo);

Multiplicar(matrizAHijo,matrizBHijo,resMult);

Matriz2Tuberia(resMult,resMMult); //El resultado de la suma se guarda en el descriptor resMMult

printf("[hijo] Inserte datos (A):\n");

InsEnMat(matrizAHijo,SIZ\_MAX,SIZ\_MAX); //Dialogos para insertar MatrizA

printf("[hijo] Inserte datos (B):\n");

InsEnMat(matrizBHijo,SIZ\_MAX,SIZ\_MAX); //Dialogos para insertar MatrizA

Matriz2Tuberia(matrizAHijo,matrA); //Escribir la nueva matriz A al descriptor

Matriz2Tuberia(matrizBHijo,matrB); //Escribir matrizB al otro descriptor

if(fork()==0)

{

float resSum[SIZ\_MAX][SIZ\_MAX],matANieto[SIZ\_MAX][SIZ\_MAX],matBNieto[SIZ\_MAX][SIZ\_MAX];

Tuberia2Matriz(matrA,matANieto); //Leer y guardar en una matriz desde la tuberia

Tuberia2Matriz(matrB,matBNieto); //Leer y guardar en una matriz desde la tuberia

SumaOResta(matANieto,matBNieto,resSum,1); //sumarlas

Matriz2Tuberia(resSum,resSSum); //guardarlas en el otro descriptor (matA es para la multiplicacion)

exit(0);

}

wait(0); exit(0);

}

wait(0);

system("clear");

Tuberia2Matriz(resMMult,matrizA);

Tuberia2Matriz(resSSum,matrizB);

printf("Resultado multiplicacion: \n");

ImpMat(matrizA,NULL,1,1,0);

printf("Resultado suma: \n");

ImpMat(matrizB,NULL,1,1,0);

if(determ(matrizA,SIZ\_MAX)!=0)

{

TreInversaCaller(matrizA,InvA,SIZ\_MAX);

printf("Inversa de la multiplicacion: \n");

ImpMat(InvA,"Inversa de la multiplicacionTub.txt",0,0,0);

}

else

printf("[ERROR] La multiplicacion no fue L.I.\n");

if(determ(matrizB,SIZ\_MAX)!=0)

{

TreInversaCaller(matrizB,InvB,SIZ\_MAX);

printf("Inversa de la suma:\n");

ImpMat(InvB,"Inversa de la sumaTub.txt",0,0,0);

}

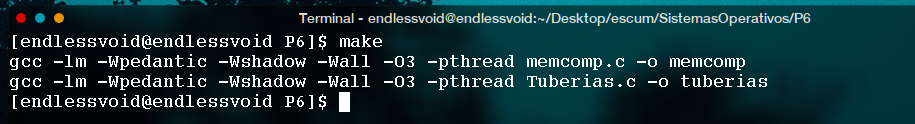
else

printf("[ERROR] La suma no fue L.I.\n");

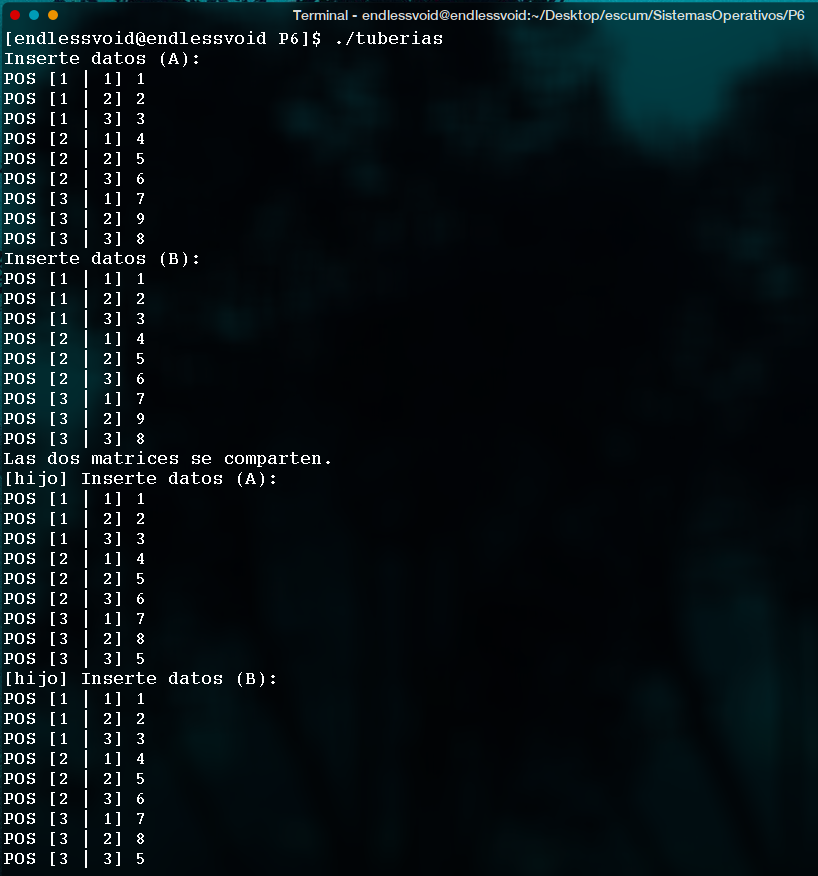
}

**Capturas de programa que utiliza tuberías:**

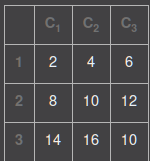
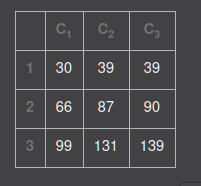
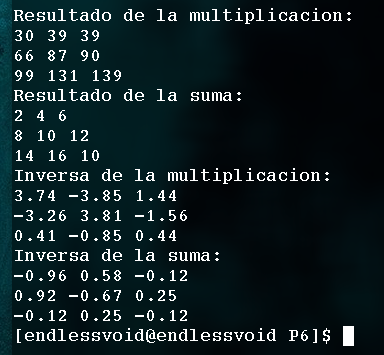
Compilación (de este programa y el siguiente):



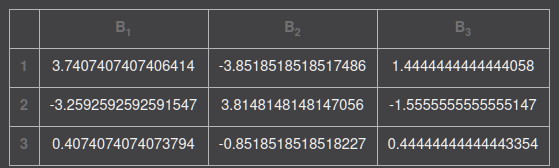
Ingreso de datos:

****

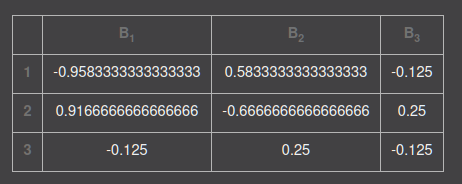
Resultados (del programa, así como la comprobación utilizando un sitio web de la multiplicación, de la suma, así como de las inversas de la multiplicación y suma):

****

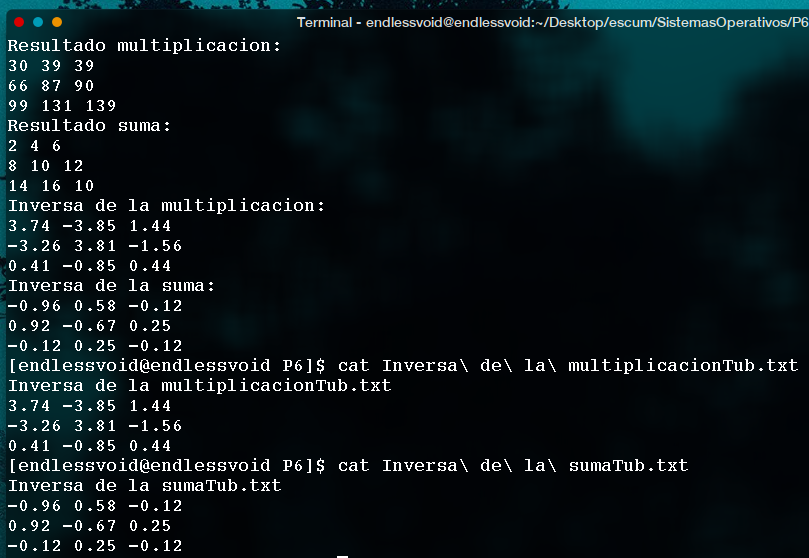
Inversa multiplicación:

****

Inversa suma:

****

Comprobación de que los archivos de las inversas estén en disco con los contenidos correctos:

****

**Programa que realiza suma y multiplicación de matrices utilizando memoria compartida para Linux (punto 7):**

**#include <stdlib.h>**

**#include <stdio.h>**

**#include "operadores.h"**

**#define SIZ\_MAX 3**

**int main()**

**{**

**float matrizA[SIZ\_MAX][SIZ\_MAX],matrizB[SIZ\_MAX][SIZ\_MAX],resultadoSum[SIZ\_MAX][SIZ\_MAX],resultadoMult[SIZ\_MAX][SIZ\_MAX],InvA[SIZ\_MAX][SIZ\_MAX],InvB[SIZ\_MAX][SIZ\_MAX];**

**printf("Inserte datos (A):\n");**

**InsEnMat(matrizA,SIZ\_MAX,SIZ\_MAX); //Dialogos para insertar MatrizA**

**CompartirMat(matrizA,1234); //Compartir "matrizA"**

**printf("Inserte datos (B):\n");**

**InsEnMat(matrizB,SIZ\_MAX,SIZ\_MAX); //Dialogos para insertar MatrizB**

**CompartirMat(matrizB,1235); //Compartir "matrizB"**

**if(fork()==0) //HIJO 1**

**{**

**float A[SIZ\_MAX][SIZ\_MAX],B[SIZ\_MAX][SIZ\_MAX],resMult[SIZ\_MAX][SIZ\_MAX],matHijoA[SIZ\_MAX][SIZ\_MAX],matHijoB[SIZ\_MAX][SIZ\_MAX];**

**ObtenerMatCompartida(1234,A); //Obtiene la matriz A y la guarda en "A"**

**ObtenerMatCompartida(1235,B); //Obtiene la matriz A y la guarda en "B"**

**Multiplicar(A,B,resMult);**

**CompartirMat(resMult,1236);**

**printf("[hijo] Inserte datos (A)\n");**

**InsEnMat(matHijoA,SIZ\_MAX,SIZ\_MAX); //Dialogos para insertar "matHijoA"**

**CompartirMat(matHijoA,1237); //Compartirlo con llave 1237**

**printf("[hijo] Inserte datos (B)\n");**

**InsEnMat(matHijoB,SIZ\_MAX,SIZ\_MAX); // Dialogos**

**CompartirMat(matHijoB,1238); //Compartir con llave 1238**

**if(fork()==0)**

**{**

**float reeeeSuma[SIZ\_MAX][SIZ\_MAX],aaa[SIZ\_MAX][SIZ\_MAX],bbb[SIZ\_MAX][SIZ\_MAX];**

**ObtenerMatCompartida(1237,aaa); //Obtener lo que este con la llave 1237 y guardarlo en aaa**

**ObtenerMatCompartida(1238,bbb); //Obtener y guardar en bbb**

**SumaOResta(aaa,bbb,reeeeSuma,1);**

**CompartirMat(reeeeSuma,1239);**

**exit(0);**

**}**

**wait(0);**

**exit(0);**

**}**

**wait(0);**

**system("clear");**

**ObtenerMatCompartida(1236,resultadoMult);**

**ObtenerMatCompartida(1239,resultadoSum);**

**printf("Resultado de la multiplicacion:\n");**

**ImpMat(resultadoMult,NULL,1,1,0);**

**printf("Resultado de la suma:\n");**

**ImpMat(resultadoSum,NULL,1,1,0);**

**if(determ(resultadoMult,SIZ\_MAX)!=0)**

**{**

**TreInversaCaller(resultadoMult,InvA,SIZ\_MAX);**

**printf("Inversa de la multiplicacion: \n");**

**ImpMat(InvA,"Inversa de la multiplicacionMem .txt",0,0,0);**

**}**

**else**

**printf("[ERROR] La multiplicacion no fue L.I.\n");**

**if(determ(resultadoSum,SIZ\_MAX)!=0)**

**{**

**TreInversaCaller(resultadoSum,InvB,SIZ\_MAX);**

**printf("Inversa de la suma:\n");**

**ImpMat(InvB,"Inversa de la sumaMem.txt",0,0,0);**

**}**

**else**

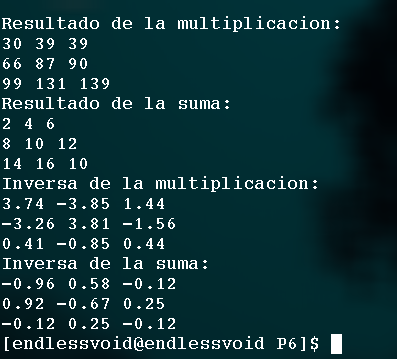
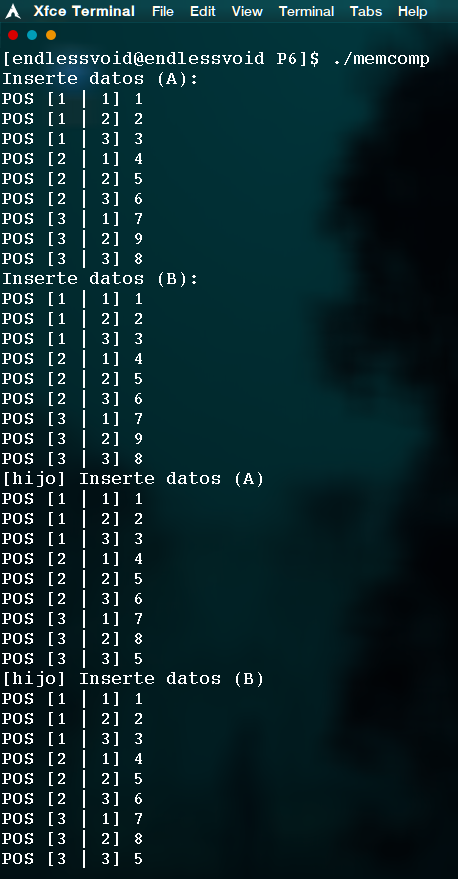
**printf("[ERROR] La suma no fue L.I.\n");**

**for(int i=1234;i<1240;i++)**

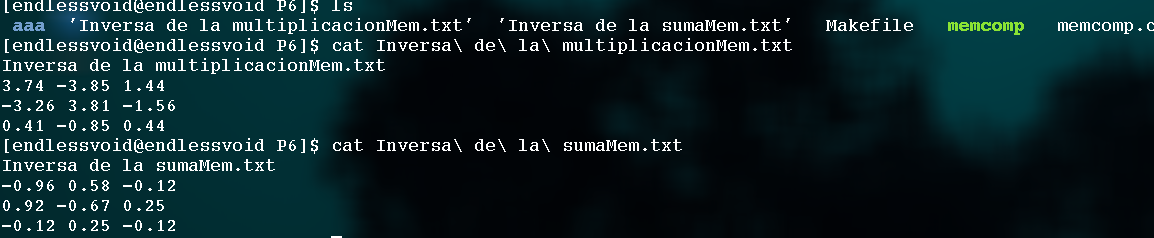
**EliminarDeMemoria(i); //Eliminar de memoria las matrices**

**}**

Capturas de pantalla (ejecución y resultados):

****

Comprobación de que los archivos de las inversas estén en disco con los contenidos correctos:

****

Matrices de 10x10 (una utilizada para la multiplicación y la otra para la suma):

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2

2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4

3 3 3 3 3 3 3 3 3 3 5 5 5 5 5 6 6 6 6 6

4 4 4 4 4 4 4 4 4 4 7 7 7 7 7 8 8 8 8 8

5 5 5 5 5 5 5 5 5 5 9 9 9 9 9 10 10 10 10 10

6 6 6 6 6 6 6 6 6 6 9 9 9 9 9 8 8 8 8 8

7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 6 6 6 6 6

8 8 8 8 8 8 8 8 8 8 5 5 5 5 5 4 4 4 4 4

9 9 9 9 9 9 9 9 9 9 3 3 3 3 3 2 2 2 2 2

10 10 10 10 10 10 10 10 10 10 1 1 1 1 1 2 2 2 2 2

Salida de los dos programas con matrices anteriores que son **LINEALMENTE DEPENDIENTES (no invertibles)** de 10x10 (ambos programas tienen la misma salida):

****

**Archivo de cabecera donde se encuentran implementadas todas las funciones utilizadas en los dos programas anteriores (ubicándose hasta abajo las implementadas en el programa):**

**#include<stdio.h>**

**#include<curses.h>**

**#include<math.h>**

**#include<string.h>**

**#include<pthread.h>**

**#include<sys/shm.h>**

**#include<sys/wait.h>**

**#include<unistd.h>**

**#include<sys/syscall.h>**

**#define SIZ\_MAX 10**

**float determ(float a[SIZ\_MAX][SIZ\_MAX], float tam);**

**// int < determ(matrizA, tamMatriz)**

**void TreInversaCaller(float [SIZ\_MAX][SIZ\_MAX], float dest[SIZ\_MAX][SIZ\_MAX], float);**

**// void < TreInversaCaller(matrizOrigen, matrizDestino, tamMatriz)**

**void treinversa(float [][SIZ\_MAX], float [][SIZ\_MAX], float dest[][SIZ\_MAX], float); //llamada indirecta**

**void ImpMat(float matrix[SIZ\_MAX][SIZ\_MAX], char\* nombre, int nodecim, int nofile, int nomostrar);**

**// void < ImpMatriz (matrizAImprimir, TamFila, TamColumna, EtiquetaParaArchivo, NoMostrarDecimales(1), NoImprimirA-Archivo(1), NoMostrar(1))**

**void InsEnMat(float matrix[SIZ\_MAX][SIZ\_MAX], int row, int column);**

**// void < InsertarEnMatriz (matrizAInsertar, TamFila, TamColumna)**

**void SumaOResta (float matrix[SIZ\_MAX][SIZ\_MAX], float matrix2[SIZ\_MAX][SIZ\_MAX], float matrixDest[SIZ\_MAX][SIZ\_MAX], int pain);**

**// void < SumaOResta (matrizA, matrizB, matrizDestino,FlagSuma(1))**

**void Transpuesta(float matrix[SIZ\_MAX][SIZ\_MAX], float matrixDest[SIZ\_MAX][SIZ\_MAX], int row, int column);**

**// void < transpuesta(matrizOrigen, matrizDestino, TamFila, TamColumna)**

**void Multiplicar(float matrix[SIZ\_MAX][SIZ\_MAX], float matrix2[SIZ\_MAX][SIZ\_MAX], float matrixDest[SIZ\_MAX][SIZ\_MAX]);**

**// void < Multiplicar(matrizA, matrizB, matrizDestino)**

**void CargarEnArreglo(char \*NombreArchivo, float matrix[SIZ\_MAX][SIZ\_MAX],int tam, int skiprows);**

**// void < CargarEnArreglo("nombreArchivo",matrizDestino, TamFila, TamColumna, SaltarNLineas)**

**int ObtenerIDHilo(void );**

**// Devuelve el ID del hilo.**

**void CompartirMat (float matrizOrigen[SIZ\_MAX][SIZ\_MAX],int key);**

**//void CompartirMat (Matriz2D Origen,llave);**

**void ObtenerMatCompartida(int key, float matrizDestino[SIZ\_MAX][SIZ\_MAX]);**

**//void ObtenerMatCompartida(llave, matriz2D-Destino)**

**void CrearEspera(int key);**

**//void CrearEspera(llave)**

**void TerminarEspera(int key);**

**//void CrearEspera(llave);**

**void EliminarDeMemoria (int key);**

**//void EliminarDeMemoria (llave)**

**void CopiarMatriz(float matrizOrigen[SIZ\_MAX][SIZ\_MAX],float matrizDestino[SIZ\_MAX][SIZ\_MAX]);**

**//void CopiarMatriz(matrizOrigen,matrizDestino)**

**void Matriz2Tuberia(float matriz[SIZ\_MAX][SIZ\_MAX], int tuberia[]);**

**//void Matriz2Tuberia (matrizOrigen, apuntadorTuberia)**

**void Tuberia2Matriz(int tuberia[], float matriz[SIZ\_MAX][SIZ\_MAX]);**

**//void Tuberia2Mat(apuntadorTuberia, matrizDestino)**

**float determ(float a[SIZ\_MAX][SIZ\_MAX], float tam)**

**{**

**float s = 1, det = 0, b[SIZ\_MAX][SIZ\_MAX];**

**int i, j, m, n, c;**

**if (tam == 1)**

**{**

**return (a[0][0]);**

**}**

**else**

**{**

**det = 0;**

**for (c = 0; c < tam; c++)**

**{**

**m = 0;**

**n = 0;**

**for (i = 0;i < tam; i++)**

**{**

**for (j = 0 ;j < tam; j++)**

**{**

**b[i][j] = 0;**

**if (i != 0 && j != c)**

**{**

**b[m][n] = a[i][j];**

**if (n < (tam - 2))**

**n++;**

**else**

**{**

**n = 0;**

**m++;**

**}**

**}**

**}**

**}**

**det = det + s \* (a[0][c] \* determ(b, tam - 1));**

**s = -1 \* s;**

**}**

**}**

**return (det);**

**}**

**void TreInversaCaller(float num[SIZ\_MAX][SIZ\_MAX],float dest[SIZ\_MAX][SIZ\_MAX], float f)**

**{**

**float b[SIZ\_MAX][SIZ\_MAX], fac[SIZ\_MAX][SIZ\_MAX];**

**int p, q, m, n, i, j;**

**for (q = 0;q < f; q++)**

**{**

**for (p = 0;p < f; p++)**

**{**

**m = 0;**

**n = 0;**

**for (i = 0;i < f; i++)**

**{**

**for (j = 0;j < f; j++)**

**{**

**if (i != q && j != p)**

**{**

**b[m][n] = num[i][j];**

**if (n < (f - 2))**

**n++;**

**else**

**{**

**n = 0;**

**m++;**

**}**

**}**

**}**

**}**

**fac[q][p] = pow(-1, q + p) \* determ(b, f - 1);**

**}**

**}**

**treinversa(num, fac, dest, f);**

**}**

**void treinversa(float num[SIZ\_MAX][SIZ\_MAX], float fac[SIZ\_MAX][SIZ\_MAX], float dest[SIZ\_MAX][SIZ\_MAX], float r)**

**{**

**int i, j;**

**float b[SIZ\_MAX][SIZ\_MAX], d;**

**for (i = 0;i < r; i++)**

**for (j = 0;j < r; j++)**

**b[i][j] = fac[j][i];**

**d = determ(num, r);**

**for (i = 0;i < r; i++)**

**for (j = 0;j < r; j++)**

**dest[i][j] = b[i][j] / d;**

**}**

**void InsEnMat (float matrix[SIZ\_MAX][SIZ\_MAX], int row, int column)**

**{**

**int i, j;**

**for (i=0;i<row;i++)**

**{**

**for (j=0;j<column;j++)**

**{**

**printf ("POS [%d | %d] ", i+1, j+1);**

**scanf ("%f", &matrix[i][j]);**

**}**

**}**

**}**

**void ImpMat (float matrix[SIZ\_MAX][SIZ\_MAX], char\* operacion, int nodecim, int nofile, int nomostrar)**

**{**

**int i, j;**

**FILE \*NuevoArchivo=NULL;**

**if(nofile!=1)**

**NuevoArchivo = fopen(operacion, "w");**

**if(NuevoArchivo!=NULL && nofile!=1)**

**fprintf(NuevoArchivo,"%s \n",operacion);**

**for(i=0;i<SIZ\_MAX;i++)**

**{**

**for(j=0;j<SIZ\_MAX;j++)**

**{**

**if(nodecim==1)**

**{**

**if(nomostrar!=1)**

**printf("%.0lf ", matrix[i][j]);**

**if(NuevoArchivo!=NULL && nofile!=1)**

**fprintf(NuevoArchivo,"%.0lf ",matrix[i][j]);**

**}**

**else**

**{**

**if(nomostrar!=1)**

**printf("%.2lf ", matrix[i][j]);**

**if(NuevoArchivo!=NULL && nofile!=1)**

**fprintf(NuevoArchivo,"%.2lf ",matrix[i][j]);**

**}**

**}**

**if(nomostrar!=1)**

**printf("\n");**

**if(NuevoArchivo!=NULL && nofile!=1)**

**fprintf(NuevoArchivo,"\n");**

**}**

**if(NuevoArchivo!=NULL && nofile!=1)**

**fclose(NuevoArchivo);**

**}**

**void SumaOResta (float matrix[SIZ\_MAX][SIZ\_MAX], float matrix2[SIZ\_MAX][SIZ\_MAX], float matrixDest[SIZ\_MAX][SIZ\_MAX], int pain)**

**{**

**int i;**

**int j;**

**for(i=0;i<SIZ\_MAX;i++)**

**for(j=0;j<SIZ\_MAX;j++)**

**{**

**if(pain==1)**

**matrixDest[i][j]=matrix[i][j]+matrix2[i][j];**

**else**

**matrixDest[i][j]=matrix[i][j]-matrix2[i][j];**

**}**

**}**

**void Transpuesta(float matrix[SIZ\_MAX][SIZ\_MAX], float matrixDest[SIZ\_MAX][SIZ\_MAX], int row, int column)**

**{**

**int i;**

**int j;**

**for(i=0;i<row;i++)**

**for(j=0;j<column;j++)**

**{**

**matrixDest[i][j]=matrix[j][i];**

**}**

**}**

**void Multiplicar(float matrix[SIZ\_MAX][SIZ\_MAX], float matrix2[SIZ\_MAX][SIZ\_MAX], float matrixDest[SIZ\_MAX][SIZ\_MAX])**

**{**

**int i, j, k;**

**for (i=0;i<SIZ\_MAX; i++)**

**{**

**for (j=0;j<SIZ\_MAX;j++)**

**{**

**matrixDest[i][j]=0;**

**for (k = 0; k < SIZ\_MAX; k++)**

**matrixDest[i][j]+=matrix[i][k]\*matrix2[k][j];**

**}**

**}**

**}**

**void LeerArchivos(char \*NombreArchivo)**

**{**

**char c; FILE \*PTRNomArch;**

**PTRNomArch = fopen(NombreArchivo, "r");**

**if (PTRNomArch == NULL)**

**{**

**printf("No se pudo abrir el archivo %s\n",NombreArchivo);**

**exit(0);**

**}**

**c = fgetc(PTRNomArch);**

**while (c != EOF)**

**{**

**printf ("%c", c);**

**c = fgetc(PTRNomArch);**

**}**

**fclose(PTRNomArch);**

**}**

**void CargarEnArreglo(char \*NombreArchivo, float matrix[SIZ\_MAX][SIZ\_MAX], int tam, int skiprows)**

**{**

**int counter=0, numberRead=0, i, j,cont=0;**

**int array[100];**

**FILE \*PTRNomArch;**

**PTRNomArch = fopen(NombreArchivo, "r");**

**if (PTRNomArch == NULL)**

**{**

**printf("No se pudo abrir el archivo %s\n",NombreArchivo);**

**exit(0);**

**}**

**char buffer[100]; //la primera linea se asume que es de 100**

**if (skiprows>0)**

**for(int rowskipped=0;rowskipped<skiprows;rowskipped++)**

**fgets(buffer, 100, PTRNomArch);**

**char arr[666];**

**char\* ptr;**

**fread(arr , 1,sizeof arr , PTRNomArch);**

**ptr = strtok(arr , " ");**

**while(ptr)**

**{**

**array[counter++] = strtol(ptr , NULL , 10);**

**++numberRead;**

**ptr = strtok(NULL , " ");**

**}**

**for(i=0;i<tam;i++)**

**for(j=0;j<tam;j++)**

**{**

**matrix[i][j]=(double)array[cont++];**

**// cont++;**

**}**

**fclose(PTRNomArch);**

**}**

**void CompartirMat (float matrizOrigen[SIZ\_MAX][SIZ\_MAX],int key)**

**{**

**float matrix[SIZ\_MAX\*SIZ\_MAX];**

**float \*matrizComp;**

**int shmid;**

**if ((shmid = shmget((key\_t) key, sizeof matrix[SIZ\_MAX\*SIZ\_MAX], IPC\_CREAT | 0666)) < 0) {**

**printf("Error al crear el segmento de memoria.");**

**exit(1);**

**}**

**if ((matrizComp = shmat(shmid, NULL, 0)) == (void \*) -1) {**

**printf("Error al obtener la memoria compartida");**

**exit(1);**

**}**

**int cont=0;**

**for(int i=0;i<SIZ\_MAX;i++) //Este ciclo sirve para**

**for(int j=0;j<SIZ\_MAX;j++) //Convertir una matriz bidimensional a una**

**{ //matriz lineal, que luego se va compartir**

**matrix[cont++]=matrizOrigen[i][j];**

**}**

**float \*auxbkp=matrizComp; //Apuntador auxiliar**

**for (int i=0;i<SIZ\_MAX\*SIZ\_MAX;i++) //Usa los valores el arreglo lineal...**

**{ //...y los guarda en el contenido del apuntador.**

**\*(auxbkp++)=matrix[i]; //Ir a la siguiente region**

**}**

**//printf("[OK/CML] key: %d\n",key);**

**}**

**void ObtenerMatCompartida(int key, float matrizDestino[SIZ\_MAX][SIZ\_MAX])**

**{**

**int shmid;**

**float \*matrizComp;**

**float matrix[SIZ\_MAX\*SIZ\_MAX];**

**if ((shmid = shmget((key\_t) key, sizeof matrix[SIZ\_MAX\*SIZ\_MAX], IPC\_CREAT | 0666)) < 0) {**

**printf("Error al crear el segmento de memoria.");**

**exit(1);**

**}**

**// printf("[INFO/ObMatCmp] key: %d\n",key);**

**if ((matrizComp = shmat(shmid, NULL, 0)) == (void \*) -1) {**

**printf("Error al obtener la memoria compartida");**

**exit(1);**

**}**

**float \*auxbkp=matrizComp; //Apuntador auxiliar**

**for(int i=0;i<SIZ\_MAX\*SIZ\_MAX;i++)**

**{**

**matrix[i]=\*(auxbkp++); //En la matriz lineal se guarda el contenido**

**} //del apuntador, y se incrementa el apuntador**

**int cont=0;**

**for(int i=0;i<SIZ\_MAX;i++) //Estos ciclos son para transformar la matriz**

**for(int j=0;j<SIZ\_MAX;j++)//lineal a una matriz bidimensional**

**{ //para facilitar el uso de la funcion.**

**matrizDestino[i][j]=matrix[cont++];**

**}**

**}**

**void CrearEspera(int key)**

**{**

**int ref[1]; //Esta funcion sirve como semaforo si es que se utiliza**

**int \*matrizComp; //Un cliente y servidor en procesos separados.**

**int shmid;**

**if ((shmid = shmget((key\_t) key, sizeof ref[1], IPC\_CREAT | 0666)) < 0) {**

**printf("Error al crear el segmento de memoria.");**

**exit(1);**

**}**

**if ((matrizComp = shmat(shmid, NULL, 0)) == (void \*) -1) {**

**printf("Error al obtener la memoria compartida");**

**exit(1);**

**}**

**printf("[INFO/WAIT] esperando unlock: %d\n",key);**

**while (\*matrizComp != key)**

**sleep(1);**

**if(\*matrizComp==key)**

**printf("[INFO/WAIT] unlock recibido: %d\n",key);**

**}**

**void TerminarEspera(int key)**

**{**

**int ref[1]; //Esta funcion sirve para controlar el semaforo**

**int \*matrizComp; //Si es que se utiliza cliente y servidor.**

**int shmid;**

**if ((shmid = shmget((key\_t) key, sizeof ref[1], IPC\_CREAT | 0666)) < 0) {**

**printf("Error al crear el segmento de memoria.");**

**exit(1);**

**}**

**if ((matrizComp = shmat(shmid, NULL, 0)) == (void \*) -1) {**

**printf("Error al obtener la memoria compartida");**

**exit(1);**

**}**

**\*matrizComp=key;**

**if(shmctl(shmid, IPC\_RMID, NULL)!=-1)**

**printf("[INFO/TE] liberando: %d\n",key);**

**else**

**printf("[ERROR] No existe/no se pudo liberar %d\n",key);**

**}**

**void EliminarDeMemoria (int key)**

**{**

**float ref[SIZ\_MAX\*SIZ\_MAX]; //Sirve para eliminar los datos que podrian**

**int shmid; //permanecer en memoria y causar conflictos.**

**if ((shmid = shmget((key\_t) key, sizeof ref[SIZ\_MAX\*SIZ\_MAX], IPC\_CREAT | 0666)) < 0) {**

**printf("Error al crear el segmento de memoria.");**

**exit(1);**

**}**

**else**

**shmctl(shmid, IPC\_RMID, NULL);**

**}**

**int ObtenerIDHilo(void ){**

**pid\_t threadid = syscall(SYS\_gettid);**

**return threadid; //Sirve para obtener el identificador del hilo**

**}**

**void CopiarMatriz(float matrizOrigen[SIZ\_MAX][SIZ\_MAX],float matrizDestino[SIZ\_MAX][SIZ\_MAX]){**

**for(int i=0;i<SIZ\_MAX;i++)**

**for(int j=0;j<SIZ\_MAX;j++)**

**matrizDestino[i][j]=matrizOrigen[i][j];**

**}**

**void Matriz2Tuberia(float matriz[SIZ\_MAX][SIZ\_MAX], int tuberia[]){**

**for(int i=0;i<SIZ\_MAX;i++) //Cada elemento de la matriz, se escribe a la tuberia**

**for(int j=0;j<SIZ\_MAX;j++) //**

**write(tuberia[1],&matriz[i][j],sizeof(matriz[i][j]));**

**}**

**void Tuberia2Matriz(int tuberia[], float matriz[SIZ\_MAX][SIZ\_MAX]){**

**int cont=0; // Contador para la matriz temporal que es lineal**

**float TempMat[SIZ\_MAX\*SIZ\_MAX],buffer=0;**

**for(int i=0;i<SIZ\_MAX\*SIZ\_MAX;i++) //Leer la tuberia y guardar las cosas en TempMat**

**if(read(tuberia[0], &buffer, sizeof(buffer)))**

**TempMat[cont++]=buffer;**

**for(int i=2;i>=0;i--) //Se empieza desde el ultimo elemento**

**for(int j=2;j>=0;j--)//para no tener que hacer cont=0**

**matriz[i][j]=TempMat[--cont];**

**}**

****

Ejemplo Pipes:

Código 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, /\*hereda al proceso hijo los manejadores de la tuberia del padre\*/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;

}

Código Hijo:

#include "windows.h"

#include "stdio.h"

int main(){

char mensaje[20];

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 desde padre: %s\n",mensaje);

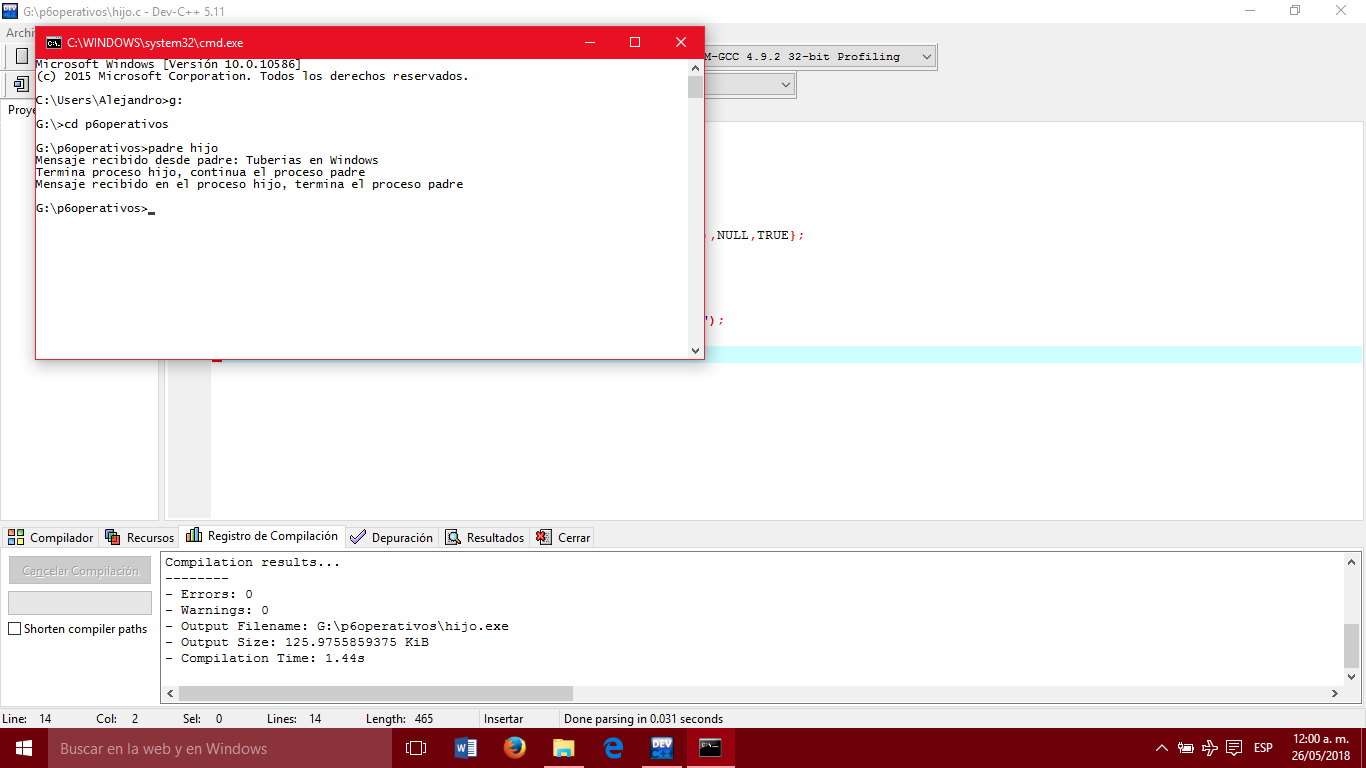
CloseHandle(hStdln);

printf("Termina proceso hijo, continua el proceso padre\n");

return 0;

}

Captura:



**Ejemplo Memoria Compartida**

Servidor:

#include "windows.h" /\*Servidor de la memoria compartida\*/

#include "stdio.h" /\*Ejecutar el servidor antes del cliente\*/

#define TAM\_MEM 27 /\*Tamaño de la memoria compartida en bytes\*/

int main(){

HANDLE hArchMapeo;

char \*idMemCompartida = "MemoriaCompartida";

char \*apDatos,\*apTrabajo,c;

if((hArchMapeo=CreateFileMapping(

INVALID\_HANDLE\_VALUE, /\*Usa memoria compartida\*/

NULL, //Seguridad por defecto

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 momria compartida

)==NULL){

printf("No se mapeo de la memoria compartida: (%i)\n",GetLastError());

exit(-1);

}

if((apDatos=(char\*)MapViewOfFile(hArchMapeo,/\*Manejador de mapeo\*/

FILE\_MAP\_ALL\_ACCESS, /\*Permiso de lectura/escritura en la memoria\*/

0,

0,

TAM\_MEM))==NULL){

printf("No se creo 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);

}

Cliente:

#include "windows.h" /\*Cliente de la memoria compartida\*/

#include "stdio.h"

#define TAM\_MEM 27 /\*Tamaño de la memoria compartida en bytes\*/

int main(){

HANDLE hArchMapeo;

char \*idMemCompartida = "MemoriaCompartida";

char \*apDatos,\*apTrabajo,c;

if((hArchMapeo=OpenFileMapping(

FILE\_MAP\_ALL\_ACCESS, //Acceso de lectura/escritura de la memoria compartida

FALSE, //No se hereda el nombre

idMemCompartida) //identificador de la momria compartida

)==NULL){

printf("No se abrio el archivo de mapeo de la memoria compartida: (%i)\n",GetLastError());

exit(-1);

}

if((apDatos=(char\*)MapViewOfFile(hArchMapeo,/\*Manejador de mapeo\*/

FILE\_MAP\_ALL\_ACCESS, /\*Permiso de lectura/escritura en la memoria\*/

0,

0,

TAM\_MEM))==NULL){

printf("No se accedió a la memoria compartida: (%i)\n",GetLastError());

CloseHandle(hArchMapeo);

exit(-1);

}

for(apTrabajo=apDatos;\*apTrabajo!='\0';apTrabajo++)

putchar(\*apTrabajo);

putchar('\n');

\*apDatos='\*';

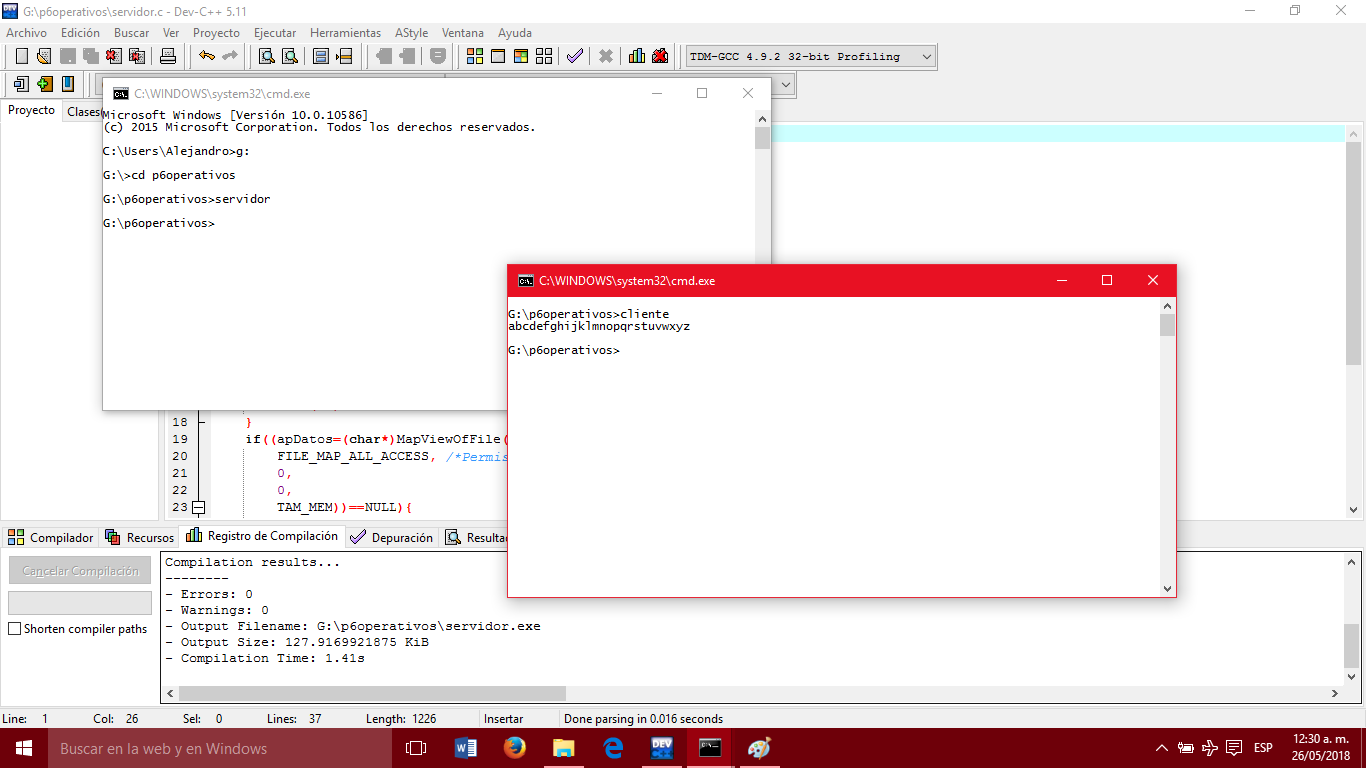
UnmapViewOfFile(apDatos);

CloseHandle(hArchMapeo);

exit(0);

}

Captura:



Programa Pipes:

**#include <windows.h>**

**#include <stdio.h>**

**#include <time.h>**

**#include "matrix.c"**

**HANDLE proceso(char \*name, HANDLE hRead, int nivel){**

**STARTUPINFO si;**

**PROCESS\_INFORMATION pi;**

**ZeroMemory(&pi, sizeof(pi));**

**ZeroMemory(&si, sizeof(si));**

**GetStartupInfo(&si);**

**si.hStdInput = hRead;**

**si.hStdError = GetStdHandle(STD\_ERROR\_HANDLE);**

**si.hStdOutput = GetStdHandle(STD\_OUTPUT\_HANDLE);**

**si.dwFlags = STARTF\_USESTDHANDLES;**

**si.cb = sizeof(si);**

**char args[100];**

**sprintf(args, "%s %d", name, nivel);**

**CreateProcess(NULL, args, NULL, NULL, TRUE, 0, NULL, NULL, &si, &pi);**

**return pi.hProcess;**

**}**

**//Create pipe**

**void creaTuberia(HANDLE \*hRead, HANDLE \*hWrite){**

**SECURITY\_ATTRIBUTES pipeSeg = {sizeof(SECURITY\_ATTRIBUTES), NULL, TRUE};**

**CreatePipe(hRead, hWrite, &pipeSeg, 0);**

**}**

**//Escribir en archivo**

**void escribir(matriz A, HANDLE hWrite){**

**int i,j;**

**for(i = 0; i < N; ++i)**

**for(j = 0; j < N; ++j)**

**WriteFile(hWrite, &A[i][j], sizeof(double), NULL, NULL);**

**}**

**//Leer archivo**

**void leer(matriz A, HANDLE hRead){**

**int i,j;**

**for(i = 0; i < N; ++i)**

**for(j = 0; j < N; ++j)**

**ReadFile(hRead, &A[i][j], sizeof(double), NULL, NULL);**

**}**

**int main(int argc, char \*argv[]){**

**printf("Pipes\n");**

**srand(time(NULL));**

**int nivel = 0; //0-padre, 1-hijo, 2-nieto**

**HANDLE hRead, hWrite;**

**if(argc > 1)**

**sscanf(argv[1], "%d", &nivel);**

**if(nivel == 0){ //padre**

**creaTuberia(&hRead, &hWrite);**

**matriz A = matrizAleatoria();**

**matriz B = matrizAleatoria();**

**escribir(A, hWrite);**

**escribir(B, hWrite);**

**WriteFile(hWrite, &hWrite, sizeof(HANDLE), NULL, NULL);**

**HANDLE hProc = proceso(argv[0], hRead, 1);**

**WaitForSingleObject(hProc, INFINITE);**

**matriz AB = nueva();**

**leer(AB, hRead);**

**printf("Proceso hijo: producto de matrices a+b.\n");**

**printMatriz(AB);**

**printf("Proceso Abuelo: Suma de matrices c+d\n");**

**matriz C\_D = nueva();**

**leer(C\_D, hRead);**

**printMatriz(C\_D);**

**matriz AB\_inv = inversa(AB);**

**matriz C\_D\_inv = inversa(C\_D);**

**writeFile(AB\_inv, "inversaa\*bPIPE.txt");**

**writeFile(C\_D\_inv, "inversac+dPIPE.txt");**

**printf("Inversa de AB:\n");**

**printMatriz(AB\_inv);**

**printf("Inversa de C+D:\n");**

**printMatriz(C\_D\_inv);**

**}else if(nivel == 1){ //hijo**

**hRead = GetStdHandle(STD\_INPUT\_HANDLE);**

**matriz A = nueva();**

**matriz B = nueva();**

**printf("Proceso Hijo: Matriz A\n");**

**leer(A, hRead);**

**printMatriz(A);**

**printf("Proceso Hijo: Matriz B\n");**

**leer(B, hRead);**

**printMatriz(B);**

**matriz AB = multiplicacion(A, B);**

**ReadFile(hRead, &hWrite, sizeof(HANDLE), NULL, NULL);**

**escribir(AB, hWrite);**

**HANDLE hRead2, hWrite2;**

**creaTuberia(&hRead2, &hWrite2);**

**matriz C = matrizAleatoria();**

**matriz D = matrizAleatoria();**

**C = matrizAleatoria();**

**D = matrizAleatoria();**

**escribir(C, hWrite2);**

**escribir(D, hWrite2);**

**WriteFile(hWrite2, &hWrite, sizeof(HANDLE), NULL, NULL);**

**HANDLE hProc = proceso(argv[0], hRead2, 2);**

**WaitForSingleObject(hProc, INFINITE);**

**}else if(nivel == 2){ //nieto**

**hRead = GetStdHandle(STD\_INPUT\_HANDLE);**

**matriz C = nueva();**

**matriz D = nueva();**

**printf("Proceso Nieto: Matriz C\n");**

**leer(C, hRead);**

**printMatriz(C);**

**printf("Proceso Nieto: Matriz D\n");**

**leer(D, hRead);**

**printMatriz(D);**

**ReadFile(hRead, &hWrite, sizeof(HANDLE), NULL, NULL);**

**matriz C\_D = suma(C, D);**

**escribir(C\_D, hWrite);**

**}**

**return 0;**

**}**

**Memoria Compartida**

**#include <windows.h>**

**#include <stdio.h>**

**#include <time.h>**

**#include "matrix.c"**

**//Crear proceso**

**void proceso(char \*name, int nivel){**

**STARTUPINFO si;**

**PROCESS\_INFORMATION pi;**

**ZeroMemory(&si, sizeof(si));**

**si.cb = sizeof(si);**

**ZeroMemory(&pi, sizeof(pi));**

**char args[100];**

**sprintf(args, "%s %d", name, nivel);**

**CreateProcess(NULL, args, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi);**

**}**

**//EScribir archivo**

**void escribir(matriz A, double \*\*p){**

**int i,j;**

**for( i = 0; i < N; ++i){**

**for( j = 0; j < N; ++j){**

**\*\*p = A[i][j];**

**(\*p)++;**

**}**

**}**

**}**

**//Leer archivo**

**void leer(matriz A, double \*\*p){**

**int i,j;**

**for(i = 0; i < N; ++i){**

**for( j = 0; j < N; ++j){**

**A[i][j] = \*\*p;**

**(\*p)++;**

**}**

**}**

**}**

**int main(int argc, char \*argv[]){**

**printf("Memoria Compartida\n");**

**srand(time(NULL));**

**int sizeMat = N \* sizeof(double);**

**int nivel = 0; //0-padre, 1-hijo, 2-nieto**

**if(argc > 1)**

**sscanf(argv[1], "%d", &nivel);**

**if(nivel == 0){**

**char \*idMem = "Producto AB";**

**HANDLE hMem = CreateFileMapping(INVALID\_HANDLE\_VALUE, NULL, PAGE\_READWRITE, 0, 3 \* sizeMat + 10, idMem);**

**double \*p = MapViewOfFile(hMem, FILE\_MAP\_ALL\_ACCESS, 0, 0, 3 \* sizeMat + 10);**

**matriz A = matrizAleatoria();**

**matriz B = matrizAleatoria();**

**escribir(A, &p);**

**escribir(B, &p);**

**proceso(argv[0], 1);**

**while(\*p != inf)**

**Sleep(1);**

**\*p++;**

**matriz AB = nueva();**

**leer(AB, &p);**

**while(\*p != inf)**

**Sleep(1);**

**UnmapViewOfFile(p);**

**CloseHandle(hMem);**

**printf("Proceso hijo: producto de matrices a+b.\n");**

**printMatriz(AB);**

**char \*idMem2 = "Suma C+D";**

**HANDLE hMem2 = OpenFileMapping(FILE\_MAP\_ALL\_ACCESS, FALSE, idMem2);**

**double \*p2 = MapViewOfFile(hMem2, FILE\_MAP\_ALL\_ACCESS, 0, 0, 3 \* sizeMat + 10);**

**printf("Proceso Abuelo: Suma de matrices c+d\n");**

**matriz C\_D = nueva();**

**leer(C\_D, &p2);**

**leer(C\_D, &p2);**

**\*p2++;**

**leer(C\_D, &p2);**

**\*p2++ = inf;**

**printMatriz(C\_D);**

**UnmapViewOfFile(p2);**

**CloseHandle(hMem2);**

**matriz AB\_inv = inversa(AB);**

**matriz C\_D\_inv = inversa(C\_D);**

**writeFile(AB\_inv, "inversaa\*bMEM.txt");**

**writeFile(C\_D\_inv, "inversac+dMEM.txt");**

**printf("Inversa de AB:\n");**

**printMatriz(AB\_inv);**

**printf("Inversa de C+D:\n");**

**printMatriz(C\_D\_inv);**

**}else if(nivel == 1){**

**char \*idMem = "Producto AB";**

**HANDLE hMem = OpenFileMapping(FILE\_MAP\_ALL\_ACCESS, FALSE, idMem);**

**double \*p = MapViewOfFile(hMem, FILE\_MAP\_ALL\_ACCESS, 0, 0, 3 \* sizeMat + 10);**

**matriz A = nueva();**

**matriz B = nueva();**

**printf("Proceso Hijo: Matriz A\n");**

**leer(A, &p);**

**printMatriz(A);**

**printf("Proceso Hijo: Matriz B\n");**

**leer(B, &p);**

**printMatriz(B);**

**matriz AB = multiplicacion(A, B);**

**\*p++ = inf;**

**escribir(AB, &p);**

**char \*idMem2 = "Suma C+D";**

**HANDLE hMem2 = CreateFileMapping(INVALID\_HANDLE\_VALUE, NULL, PAGE\_READWRITE, 0, 3 \* sizeMat + 10, idMem2);**

**double \*p2 = MapViewOfFile(hMem2, FILE\_MAP\_ALL\_ACCESS, 0, 0, 3 \* sizeMat + 10);**

**matriz C = matrizAleatoria();**

**matriz D = matrizAleatoria();**

**C = matrizAleatoria();**

**D = matrizAleatoria();**

**escribir(C, &p2);**

**escribir(D, &p2);**

**proceso(argv[0], 2);**

**while(\*p2 != inf)**

**Sleep(1);**

**\*p++ = inf;**

**UnmapViewOfFile(p);**

**CloseHandle(hMem);**

**UnmapViewOfFile(p2);**

**CloseHandle(hMem2);**

**}else if(nivel == 2){**

**char \*idMem = "Suma C+D";**

**HANDLE hMem = OpenFileMapping(FILE\_MAP\_ALL\_ACCESS, FALSE, idMem);**

**double \*p = MapViewOfFile(hMem, FILE\_MAP\_ALL\_ACCESS, 0, 0, 3 \* sizeMat + 10);**

**matriz C = nueva();**

**matriz D = nueva();**

**printf("Proceso Nieto: Matriz C\n");**

**leer(C, &p);**

**printMatriz(C);**

**printf("Proceso Nieto: Matriz D\n");**

**leer(D, &p);**

**printMatriz(D);**

**matriz C\_D = suma(C, D);**

**\*p++ = inf;**

**escribir(C\_D, &p);**

**while(\*p != inf)**

**Sleep(1);**

**UnmapViewOfFile(p);**

**CloseHandle(hMem);**

**}**

**return 0;**

**}**

**Archivo Matrix.c**

**#include <stdbool.h>**

**#include <math.h>**

**typedef double\* vector;**

**typedef vector\* matriz;**

**#define N 10**

**//Crear matriz**

**matriz nueva(){**

**int i;**

**matriz A = calloc(N, sizeof(vector));**

**for(i = 0; i < N; ++i)**

**A[i] = calloc(N, sizeof(double));**

**return A;**

**}**

**//Det = 0**

**bool esCero(double x){**

**return fabs(x) < 1e-8;**

**}**

**//Suma de matrices**

**matriz suma(matriz A, matriz B){**

**matriz C = nueva();**

**int i,j;**

**for(i = 0; i < N; ++i)**

**for(j = 0; j < N; ++j)**

**C[i][j] = A[i][j] + B[i][j];**

**return C;**

**}**

**//Resta de matrices**

**matriz resta(matriz A, matriz B){**

**matriz C = nueva();**

**int i,j;**

**for(i = 0; i < N; ++i)**

**for(j = 0; j < N; ++j)**

**C[i][j] = A[i][j] - B[i][j];**

**return C;**

**}**

**//Producto de matrices**

**matriz multiplicacion(matriz A, matriz B){**

**int i,j,k;**

**matriz C = nueva();**

**for(i = 0; i < N; ++i)**

**for(j = 0; j < N; ++j)**

**for(k = 0; k < N; ++k)**

**C[i][j] += A[i][k] \* B[k][j];**

**return C;**

**}**

**//Matriz transpuesta**

**matriz transpuesta(matriz A){**

**matriz C = nueva();**

**int i,j;**

**for(i = 0; i < N; ++i)**

**for( j = 0; j < N; ++j)**

**C[i][j] = A[j][i];**

**return C;**

**}**

**//generara matriz inversa**

**matriz inversa(matriz A){**

**matriz inv = nueva();**

**int i,j,k,l;**

**for(i = 0; i < N; ++i)**

**inv[i][i] = 1;**

**while(i < N && j < N){**

**if(esCero(A[i][j])){**

**for( k = i + 1; k < N; ++k){**

**if(!esCero(A[k][j])){**

**vector tmp = A[i];**

**A[i] = A[k];**

**A[k] = tmp;**

**tmp = inv[i];**

**inv[i] = inv[k];**

**inv[k] = tmp;**

**break;**

**}**

**}**

**}**

**if(!esCero(A[i][j])){**

**for( l = 0; l < N; ++l)**

**inv[i][l] /= A[i][j];**

**for( l = N - 1; l >= j; --l)**

**A[i][l] /= A[i][j];**

**for( k = 0; k < N; ++k){**

**if(i == k) continue;**

**for( l = 0; l < N; ++l)**

**inv[k][l] -= inv[i][l] \* A[k][j];**

**for(l = N; l >= j; --l)**

**A[k][l] -= A[i][l] \* A[k][j];**

**}**

**++i;**

**}**

**++j;**

**}**

**return inv;**

**}**

**//crea matriz de manera aleatoria**

**matriz matrizAleatoria(){**

**matriz A = nueva();**

**int i,j;**

**for(i = 0; i < N; ++i)**

**for(j = 0; j < N; ++j)**

**A[i][j] = rand() % 10;**

**return A;**

**}**

**//escribe sobre el archivo**

**void writeFile(matriz A, char\* nombre){**

**FILE \*fp = fopen(nombre, "w");**

**int i,j;**

**for( i = 0; i < N; ++i){**

**for(j = 0; j < N; ++j)**

**fprintf(fp, "%0.2f ", A[i][j]);**

**fprintf(fp, "\n");**

**}**

**fclose(fp);**

**}**

**//leer archivo**

**matriz readFile(char\* nombre){**

**matriz A = nueva();**

**int i,j;**

**FILE \* fp = fopen(nombre, "r");**

**for(i = 0; i < N; ++i)**

**for(j = 0; j < N; ++j)**

**fscanf(fp, "%.2f", &A[i][j]);**

**fclose(fp);**

**return A;**

**}**

**//imprimir archivo**

**void printMatriz(matriz A){**

**int i,j;**

**for(i = 0; i < N; ++i){**

**for( j = 0; j < N; ++j)**

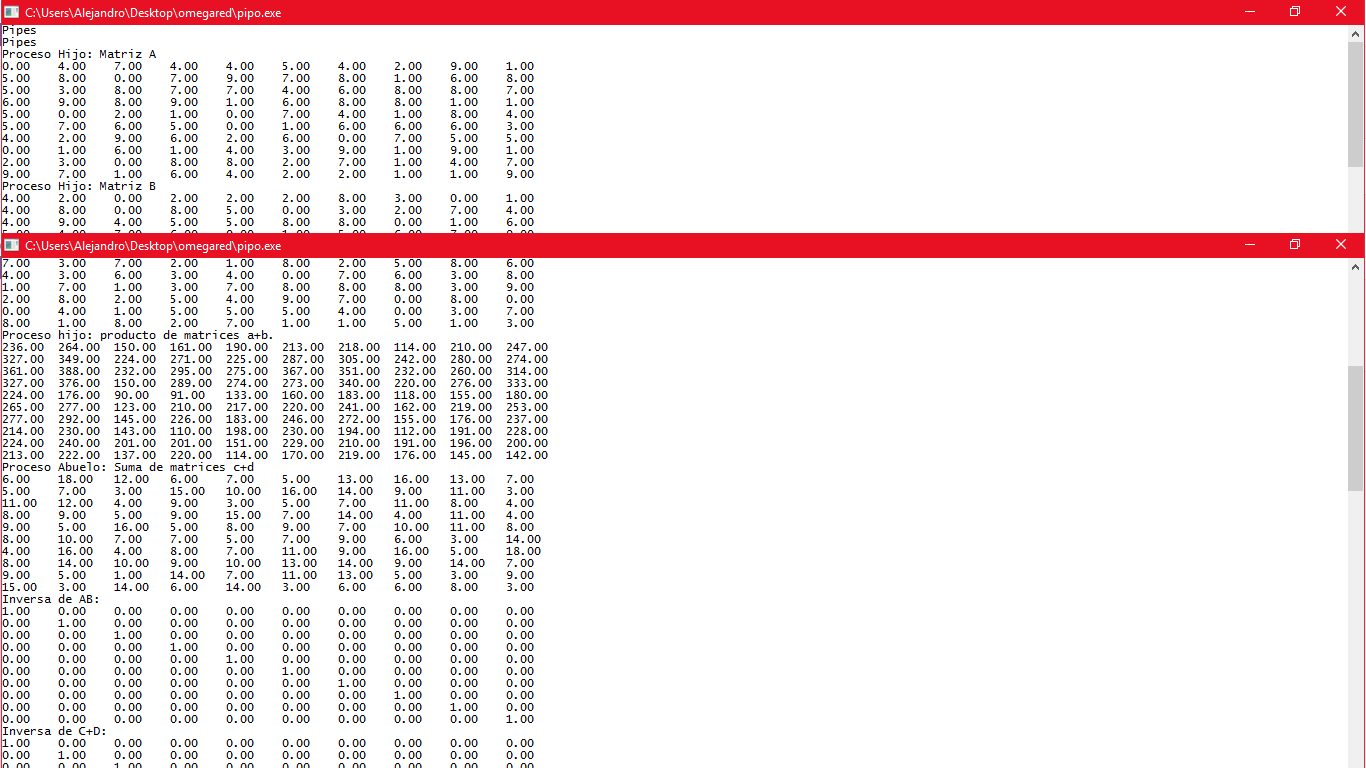
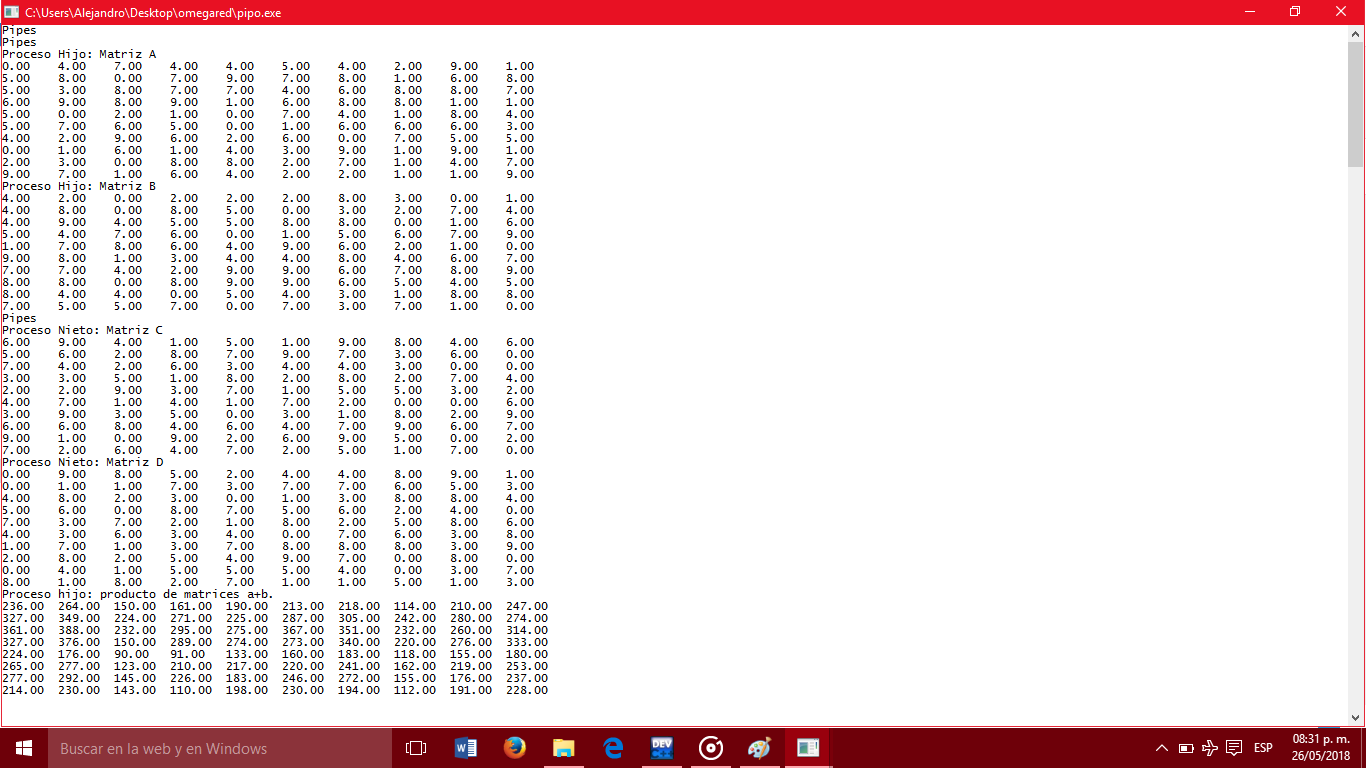
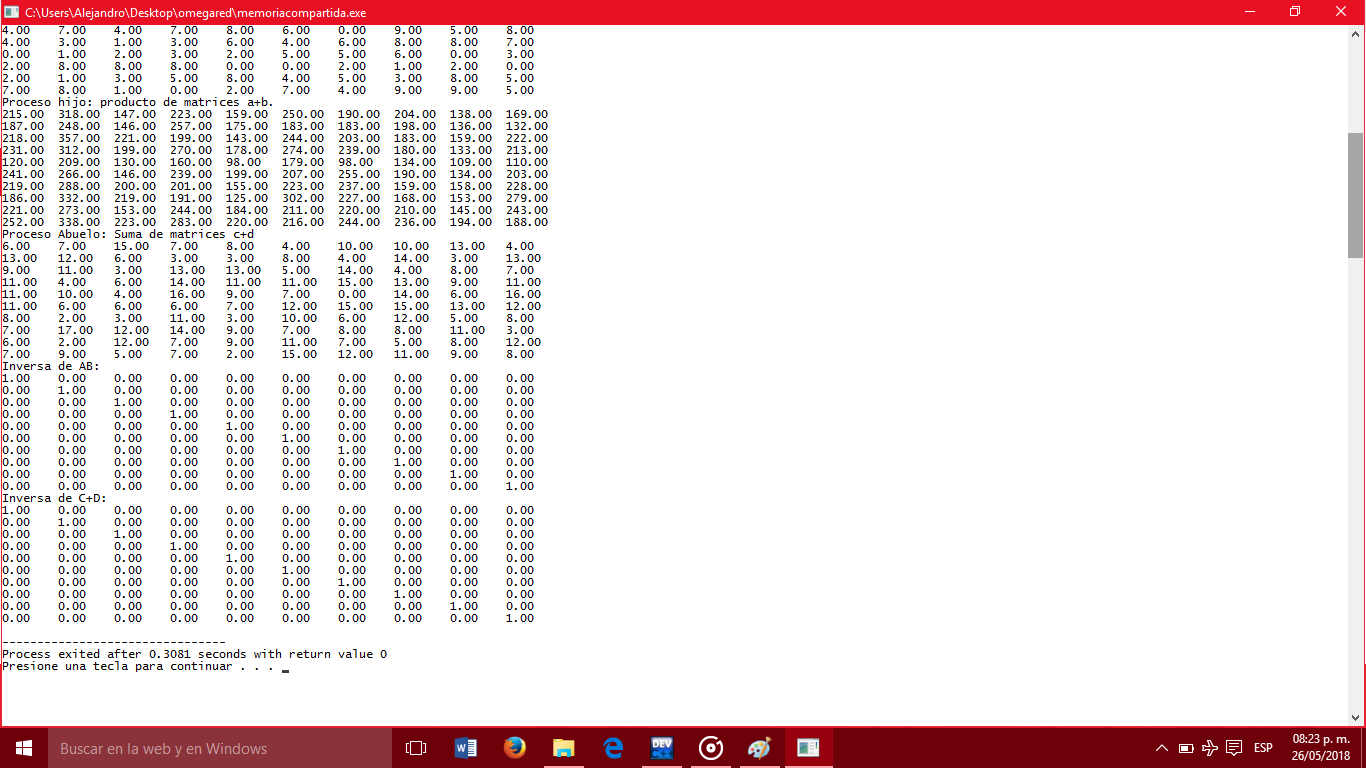
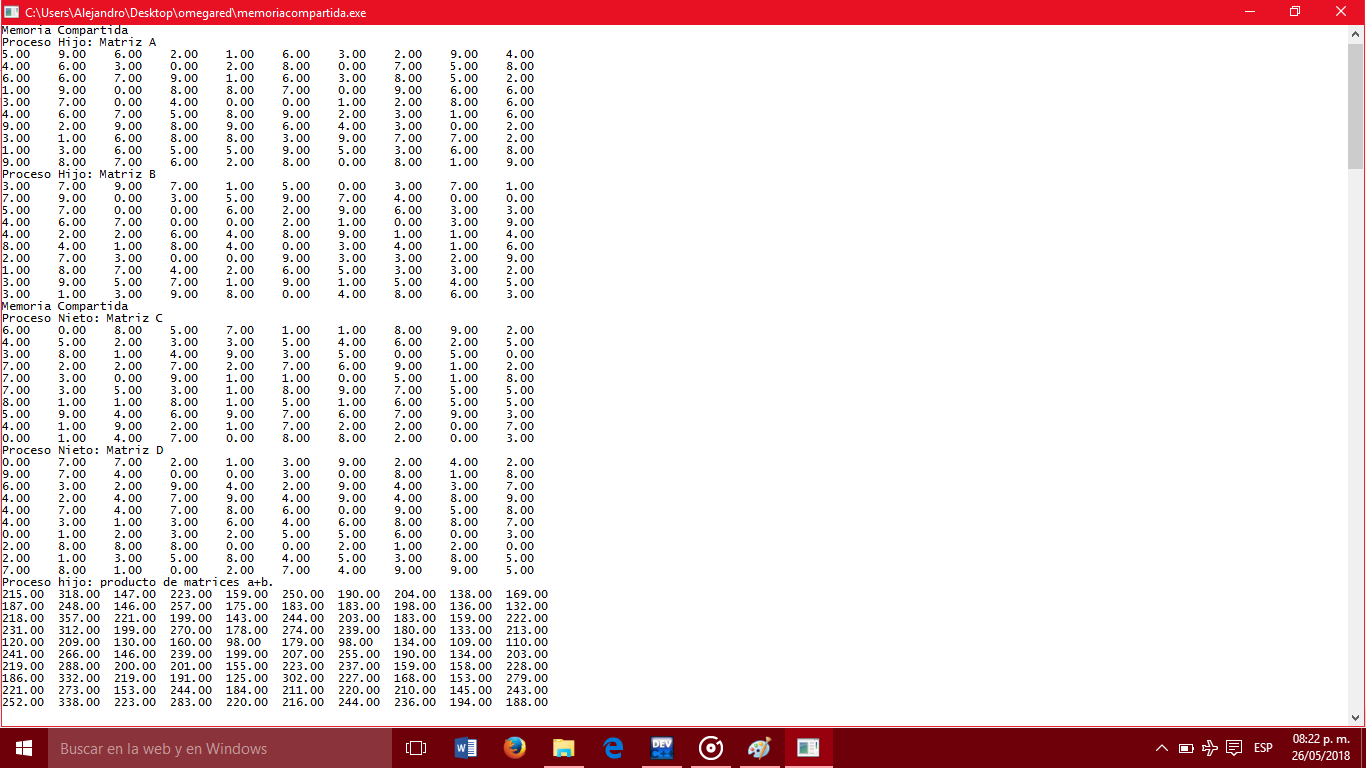
**printf("%.2f\t", A[i][j]);**

**printf("\n");**

**}**

**}**

**Capturas**

****

**Análisis Crítico**

Se pudieron observar similitudes con los programas mediante ambos métodos, ya que el objetivo era que realizaran lo mismo, sin embargo, se observaron menores problemas al realizar los programas para Windows utilizando memoria compartida, ya que de alguna manera, los recursos de la memoria se encuentran en el mismo lugar, a diferencia de las tuberías (que tienen un funcionamiento similar al de los hilos), en los cuales se tienen que mandar mandar los parámetros de trabajo para cada proceso. Al igual que en prácticas realizadas anteriormente, el portar (o realizar) las mismas operaciones que se hacen en Linux a Windows implica un poco más de esfuerzo.

**Conclusiones:**

* Reyes Valenzuela: Tuve un poco de problemas al realizar la aplicación de memoria compartida, debido al manejo de los identificadores de partición y de los espacios asignados, sin embargo si noté que fue más rápida la ejecución del programa mediante este mecanismo, que a comparación de las tuberías.
* Guerra Vargas: Con esta práctica pudimos familiarizarnos más con las diferentes maneras de comunicar función entre procesos padres e hijos, ver cómo funcionan las tuberías o pipes, darnos cuenta como deben de responderse al momento de querer compartir la información, todos los procesos la comparten, no solo los que el usuario quiera que las tenga, por ejemplo, si un nieto tiene información debe de pasarla al padre y después al abuelo, en la memoria compartida también es una forma más simplificada para pasar los datos, y en mi opinión personal, mas rapida que usar pipes.
* Dávila Méndez: En general, la implementación de memoria compartida simplifica bastante el paso de datos entre procesos, ya sea en procesos hijos de otro o en procesos independientes. Al parecer, el método de memoria compartida (utilizando pipes) es más común cuando se tienen procesos hijos dentro de otro proceso, a diferencia de cuando se utiliza shmget y funciones similares, método que permite la transferencia de información entre procesos completamente independientes. Se observó que al utilizar las funciones como shmget, es posible que los datos permanezcan en memoria durante cierto tiempo, pudiendo causar problemas, a no ser que se utilice shmctl para eliminar dichos datos. De manera similar, las tuberías no pueden ser cerradas antes de que otro proceso desee usarlas.