

INSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE COMPUTO

INGENIERÍA EN SISTEMAS COMPUTACIONALES

Sistemas Operativos

**Practica 5.**

Comunicación interprocesos (IPC) en Linux y Windows

Grupo: 4CM1

Equipo 5

Integrantes:

* Guzmán Romano Rubén Darío
* Hernández Pérez Juan Manuel
* Rivas Carrera Diana Laura
* Vázquez Rodríguez Ulises

**Introducción**

La comunicación entre procesos, en inglés IPC (Inter-process Communication) es una función básica de los sistemas operativos. Los procesos pueden comunicarse entre sí a través de compartir espacios de memoria, ya sean variables compartidas o buffers, o a través de las herramientas provistas por las rutinas de IPC. La IPC provee un mecanismo que permite a los procesos comunicarse y sincronizarse entre sí, normalmente a través de un sistema de bajo nivel de paso de mensajes que ofrece la red subyacente.

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

Los procesos pueden estar ejecutándose en una o más computadoras conectadas a una red. Las técnicas de IPC están divididas dentro de métodos para: paso de mensajes, sincronización, memoria compartida y llamadas de procedimientos remotos (RPC). El método de IPC usado puede variar dependiendo del ancho de banda y latencia (el tiempo desde el pedido de información y el comienzo del envío de la misma) de la comunicación entre procesos, y del tipo de datos que están siendo comunicados.

Administración de memoria

La parte del sistema operativo que administra la memoria se llama administrador de memoria y su labor consiste en llevar un registro de las partes de memoria que se están utilizando y aquellas que no, con el fin de asignar espacio en memoria a los procesos cuando éstos la necesiten y liberándola cuando terminen, así como administrar el intercambio entre la memoria principal y el disco en los casos en los que la memoria principal no le pueda dar capacidad a todos los procesos que tienen necesidad de ella.

Los sistemas de administración de memoria se pueden clasificar en dos tipos: los que desplazan los procesos de la memoria principal al disco y viceversa durante la ejecución y los que no.

Para poder comprender como se administra la memoria es necesario conocer primero que es la memoria principal, por lo que a continuación se citaran algunas definiciones:

La memoria principal puede ser considerada como un arreglo lineal de localidades de almacenamiento de un byte de tamaño. Cada localidad de almacenamiento tiene asignada una dirección que la identifica.

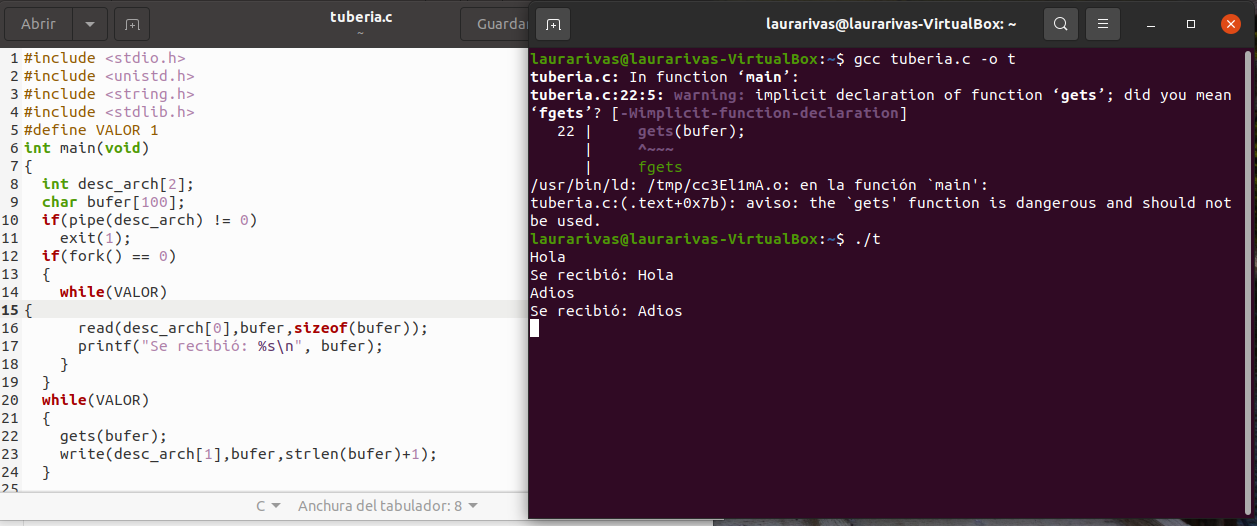
Memoria interna del sistema informático, con direcciones accesibles por los programas y que puede cargarse en los registros para su posterior ejecución o procesamiento.

Lugar en el cual se almacenan datos e instrucciones en una computadora antes y durante su ejecución.

1. Comandos

|  |  |
| --- | --- |
| pipe() | Crea una tubería  Crea una tubería, un canal de datos unidireccional que se puede utilizado para la comunicación entre procesos.  En caso de éxito, se devuelve cero. En caso de error, se devuelve -1 |
| shmget() | Asigna un segmento de memoria compartida de System V  Devuelve el identificador del segmento de memoria compartida de System V asociado con el valor de la clave del argumento.  Puede usarse para obtener el identificador de un segmento de memoria compartida creado previamente (cuando shmflg es cero y la clave no tiene el valor IPC\_PRIVATE), o para crear un nuevo conjunto. |
| shmat() | Adjunta el segmento de memoria compartida identificado por shmid al espacio de direcciones del proceso de llamada.  En caso de éxito, shmat () devuelve la dirección del segmento de memoria compartida adjunto; on error (void \*) -1 se devuelve |

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



Observaciones: En primer instante, al momento de compilarlo, nos marcaba una advertencia, donde decía que podíamos incluir stdlib.h para mejor funcionamiento.

De igual manera observamos el funcionamiento, y como vemos, podemos agregar un exit, para que este termine el proceso, pero de igual manera cumple con la función

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

#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**;

}

* **Ejecución:**

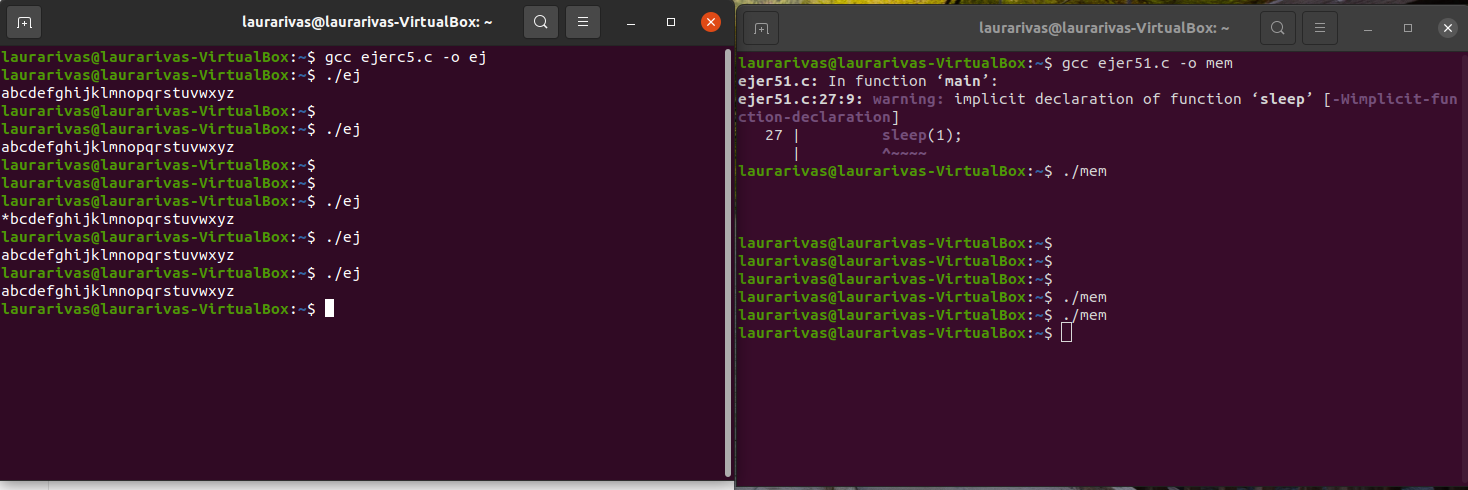
Texto

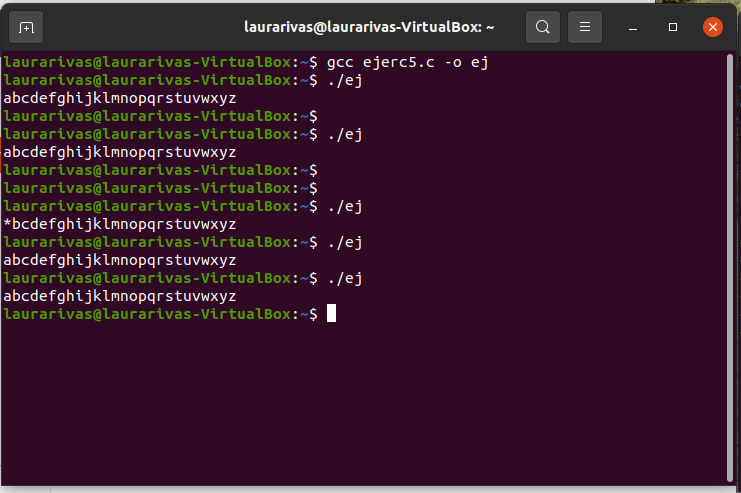
Descripción generada automáticamente

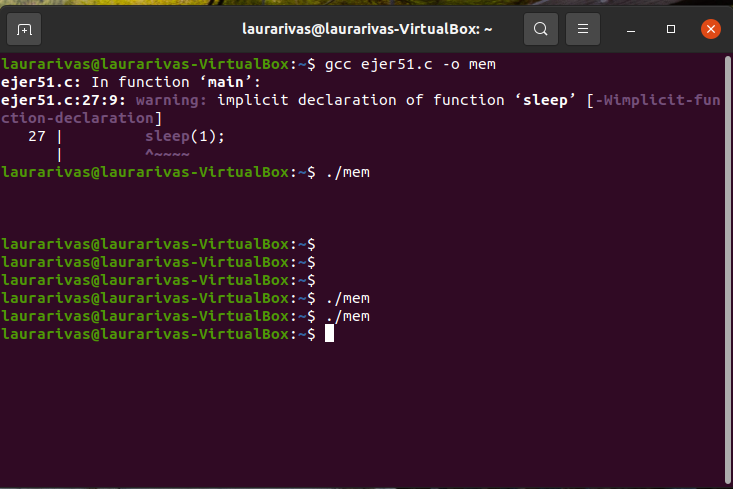
* **Observaciones:**

El proceso padre crea una tubería y escribe en ella un mensaje para el proceso hijo. Asimismo, crea al proceso hijo modificándole el manejador de su entrada estándar para que lea los mensajes de la tubería, luego se queda en espera del proceso hijo. Por otro lado, el hijo lee el mensaje escrito en la tubería, lo imprime en pantalla y termina su ejecución, entonces el proceso padre termina su ejecución también.

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

****

****

****

**Observaciones:** El primer programa espera a que el segundo ponga un asterisco en la primera posición de la cadena para terminar su ejecución

1. 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.

* **Para Linux**

#include<stdio.h>

#include<unistd.h>

#include<string.h>

#include <stdlib.h>

#define **VALOR** 1

float mat1inv[10][10]={};

float mat2inv[10][10]={};

void **Inversa**(float matrix[][40], int n,int opc){

    float temp;

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < 2 \* n; j++) {

            if (j == (i + n))

                matrix[i][j] = 1;

        }

    }

    for (int i = n - 1; i > 0; i--) {

        if (matrix[i - 1][0] < matrix[i][0])

            for (int j = 0; j < 2 \* n; j++) {

                temp = matrix[i][j];

                matrix[i][j] = matrix[i - 1][j];

                matrix[i - 1][j] = temp;

            }

    }

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < 2 \* n; j++) {

            if (j != i) {

                temp = matrix[j][i] / matrix[i][i];

                for (int k = 0; k < 2 \* n; k++) {

                    matrix[j][k] -= matrix[i][k] \* temp;

                }

            }

        }

    }

    for (int i = 0; i < n; i++) {

        temp = matrix[i][i];

        for (int j = 0; j < 2 \* n; j++) {

            matrix[i][j] = matrix[i][j] / temp;

        }

    }

    for (int i = 0; i < n; i++) {

        for (int j = n; j < 2 \* n; j++) {

            if(opc==0)

             mat1inv[i][j-n]=matrix[i][j];

            else

             mat2inv[i][j-n]=matrix[i][j];

        }

    }

    return;

}

int **main**(void){

  int tube1[2], tube2[2], tube3[2], tube4[2];

*//crea 4 tuberias*

  if(**pipe**(tube1)!=0)

**exit**(1);

  if(**pipe**(tube2)!=0)

**exit**(1);

  if(**pipe**(tube3)!=0)

**exit**(1);

  if(**pipe**(tube4)!=0)

**exit**(1);

  if(**fork**()==0){  *//proceso hijo*

    while(VALOR){

      float mult1[40][40]={};

      float mult2[40][40]={};

      float multRes[40][40]={};

float m1[40][40]={{1,0,0,0,0,1,1,0,0,0},

                  {1,2,0,0,2,0,0,0,0,0},

                  {1,1,1,2,0,1,0,1,6,0},

                  {1,2,3,2,0,1,0,0,0,0},

                  {1,2,3,0,5,0,2,0,1,0},

                  {1,2,0,1,5,6,0,0,0,0},

                  {2,2,0,4,5,6,7,0,2,0},

                  {1,2,0,4,5,6,7,8,0,0},

                  {1,2,1,4,5,6,7,8,9,0},

                  {1,2,2,4,5,6,7,8,9,1}};

float m2[40][40]={{1,9,8,7,6,5,4,3,2,1},

                  {0,9,8,7,6,5,4,3,2,1},

                  {0,4,8,7,6,5,4,3,2,1},

                  {2,0,2,7,6,5,4,3,2,1},

                  {0,1,1,0,6,5,4,3,2,1},

                  {0,2,0,0,0,5,4,3,2,1},

                  {0,3,2,0,4,0,4,3,2,1},

                  {0,4,0,0,1,0,0,3,2,1},

                  {0,0,0,4,0,3,0,0,2,1},

                  {1,1,0,2,0,0,1,2,0,1}};

*//lee la tuberia 1 en las variables mult1 y mult2*

*//para multiplicarlas*

**read**(tube1[0],mult1,sizeof(mult1));

**read**(tube1[0],mult2,sizeof(mult2));

*// multiplica matrices*

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

      for (int i = 0; i < 10; i++){

        for (int j = 0; j < 10; j++){

            multRes[i][j] = 0;

            for (int k = 0; k < 10; k++)

                multRes[i][j] +=mult1[i][k]\*mult2[k][j];

**printf**("%.0f  ", multRes[i][j]);

                 }

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

         }

*//escribe el resultado de la multiplicacion*

*//en la tuberia 3*

**write**(tube3[1], multRes,sizeof(multRes));

*//Escribe en la tuberia 2 matrices*

*//que leera su hijo/nieto del principal*

**write**(tube2[1], m1,sizeof(m1));

**write**(tube2[1], m2,sizeof(m2));

      if(**fork**()==0){    *//proceso nieto*

        while(VALOR){

          float suma1[40][40]={};

          float suma2[40][40]={};

          float sumaRes[40][40]={};

*//Lee las matrices de la tuberia 2*

*//guardandolas en variables para hacer la suma*

**read**(tube2[0],suma1,sizeof(suma1));

**read**(tube2[0],suma2,sizeof(suma2));

*//suma matrices*

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

          for(int i=0; i<10; i++) {

            for(int j=0; j<10; j++) {

              sumaRes[i][j]=(int)suma1[i][j]+suma2[i][j];

**printf**("%.0f ", sumaRes[i][j]);

                 }

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

          }

*//Escribe el resultado en la tuberia 4*

**write**(tube4[1], sumaRes,sizeof(sumaRes));

**exit**(0);

        }

      }

**exit**(0);

    }

  }

*//proceso abuelo(main)*

   while(VALOR){

*//matrices en el proceso principal*

float mat1[40][40]={{1,0,0,0,0,1,1,0,0,0},

                  {1,2,0,0,2,0,0,0,0,0},

                  {1,1,1,2,0,1,0,1,6,0},

                  {1,2,3,2,0,1,0,0,0,1},

                  {1,2,3,0,5,0,2,0,1,2},

                  {1,2,0,1,5,6,0,0,0,1},

                  {1,2,0,4,5,6,7,0,2,0},

                  {1,2,0,4,5,6,7,8,0,2},

                  {1,2,1,4,5,1,7,8,9,0},

                  {1,2,2,4,5,6,7,8,9,10}};

float mat2[40][40]={{10,9,8,7,6,5,4,3,2,1},

                  {0,9,8,7,6,5,4,3,2,1},

                  {0,4,8,7,6,5,4,3,2,1},

                  {2,0,2,7,6,5,4,3,2,1},

                  {0,1,1,0,6,5,4,3,2,1},

                  {0,2,0,0,0,5,4,3,2,1},

                  {0,3,2,0,4,0,4,3,2,1},

                  {0,4,0,0,1,0,0,3,2,1},

                  {0,0,0,4,1,1,0,0,2,1},

                  {1,1,0,2,0,1,1,2,0,1}};

float multi[40][40]={};

float suma[40][40]={};

  FILE\* fichero;

      fichero = **fopen**("inversaMult.txt", "w");

  FILE\* fichero2;

      fichero2 = **fopen**("inversaSuma.txt", "w");

*//Escribe matrices en la tuberia 1*

*//que leera el proceso hijo*

**write**(tube1[1], mat1,sizeof(mat1));

**write**(tube1[1], mat2,sizeof(mat2));

*//Lee las tuberias 3 y 4*

*//con los resultados de las operaciones*

**read**(tube3[0],multi,sizeof(multi));

**read**(tube4[0],suma,sizeof(suma));

*//calcula las inversas y las escribe en archivos*

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

**Inversa**(multi,10,0);

    for(int i=0; i<10; i++) {

      for(int j=0; j<10; j++) {

**fprintf**(fichero, "%.2f ",mat1inv[i][j]);

**printf**("%.2f  ", mat1inv[i][j]);

      }

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

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

    }

**Inversa**(suma,10,1);

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

    for(int i=0; i<10; i++) {

      for(int j=0; j<10; j++) {

**fprintf**(fichero2, "%.2f ",mat2inv[i][j]);

**printf**("%.2f  ", mat2inv[i][j]);

      }

**fprintf**(fichero2,"\n");

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

    }

**fclose**(fichero);

**fclose**(fichero2);

**exit**(0);

  }

}

**Ejecución**

Calendario

Descripción generada automáticamente

Imagen que contiene Gráfico

Descripción generada automáticamente

Calendario

Descripción generada automáticamente con confianza baja

Imagen que contiene Calendario

Descripción generada automáticamente Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Interfaz de usuario gráfica

Descripción generada automáticamente

* **Para Windows**

**Programa padre (PMatrizTub.c)**

#include <windows.h>

#include <stdio.h>

#include <string.h>

#include<math.h>

float determinante**(**float **[][**10**],** float**);**

void cofactor**(**float **[][**10**],** float**);**

void traspuesta**(**float **[][**10**],** float **[][**10**],** float**);**

**typedef** struct argumentos**{**

int matrizA**[**10**][**10**];**

int matrizB**[**10**][**10**];**

HANDLE hEscrituraPipe**;**

HANDLE hLecturaPipe**;**

**}\***datos**;**

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

int i**,**j**;** //Contadores

float d**;** //Determinante

float matrizS**[**10**][**10**],** matrizP**[**10**][**10**];**

datos datosHijo **=** **(**datos**)**malloc**(sizeof(**struct argumentos**));**

DWORD escritosHijo**,**escritosNieto**,** leidosH**,** leidosN**;**

HANDLE hLecturaPipeIda**,** hEscrituraPipeIda**,**hLecturaPipeVuelta**,** hEscrituraPipeVuelta**,**hLecturaPipeVueltaN**,** hEscrituraPipeVueltaN**;**

PROCESS\_INFORMATION piHijo**;**

STARTUPINFO siHijo**;**

SECURITY\_ATTRIBUTES pipeSeg **=**

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

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

**{**4**,**0**,**1**,**9**,**0**,**6**,**6**,**2**,**3**,**4**},**

**{**1**,**4**,**6**,**7**,**2**,**6**,**3**,**6**,**8**,**3**},**

**{**0**,**1**,**6**,**8**,**0**,**1**,**8**,**5**,**2**,**1**},** **{**9**,**6**,**0**,**4**,**9**,**1**,**4**,**9**,**3**,**2**},** **{**0**,**3**,**4**,**6**,**8**,**1**,**3**,**4**,**5**,**2**},**

**{**1**,**2**,**3**,**9**,**8**,**7**,**7**,**7**,**3**,**1**},**

**{**4**,**3**,**5**,**2**,**1**,**5**,**8**,**9**,**3**,**6**}};**

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

**for(**i**=**0**;**i**<**10**;**i**++){**//Guarda matrices en la estructura

**for(**j**=**0**;**j**<**10**;**j**++){**

datosHijo**->**matrizA**[**i**][**j**]** **=** matrizA**[**i**][**j**];**

datosHijo**->**matrizB**[**i**][**j**]** **=** matrizB**[**i**][**j**];**

**}**

**}**

// Obtención de información para la inicialización del proceso hijo

GetStartupInfo **(&**siHijo**);**

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

//Envia al Hijo

CreatePipe**(&**hLecturaPipeIda**,&**hEscrituraPipeIda**,&**pipeSeg**,** 0**);**

//Recibe del Hijo

CreatePipe**(&**hLecturaPipeVuelta**,&**hEscrituraPipeVuelta**,&**pipeSeg**,**0**);**

//Recibe del Nieto

CreatePipe**(&**hLecturaPipeVueltaN**,&**hEscrituraPipeVueltaN**,&**pipeSeg**,**0**);**

//Guarda los manejadores del nieto

datosHijo**->**hEscrituraPipe **=** hEscrituraPipeVueltaN**;**

datosHijo**->**hLecturaPipe **=** hLecturaPipeVueltaN**;**

//Envía datos al Hijo

WriteFile**(**hEscrituraPipeIda**,** datosHijo**,** **sizeof(**struct argumentos**),** **&**escritosHijo**,** **NULL);**

//Cambia las entradas y salidas del Hijo

siHijo**.**hStdInput **=** hLecturaPipeIda**;**

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

siHijo**.**hStdOutput **=** hEscrituraPipeVuelta**;**

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

//Creación del Proceso Hijo

CreateProcess **(NULL,**argv**[**1**],NULL,NULL,**TRUE/\*Hereda\*/**,**0**,NULL,NULL,&**siHijo**,&**piHijo**);**

WaitForSingleObject **(**piHijo**.**hProcess**,** INFINITE**);**

//Lee la matriz enviada por el Hijo

ReadFile**(**hLecturaPipeVuelta**,** matrizP**,** **sizeof(**matrizP**),** **&**leidosH**,** **NULL);**

printf**(**"\nLa matriz devuelta por del HIJO (Producto) es: \n\n"**);**

**for(**i**=**0**;**i**<**10**;**i**++){**

**for(**j**=**0**;**j**<**10**;**j**++)**

printf**(**"%4.0f "**,** matrizP**[**i**][**j**]);**

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

**}**

//Lee la matriz enviada por el Nieto

ReadFile**(**hLecturaPipeVueltaN**,** matrizS**,** **sizeof(**matrizS**),** **&**leidosN**,** **NULL);**

printf**(**"\n\nLa matriz devuelta por del NIETO (Suma) es: \n\n"**);**

**for(**i**=**0**;**i**<**10**;**i**++){**

**for(**j**=**0**;**j**<**10**;**j**++)**

printf**(**"%4.0f "**,** matrizS**[**i**][**j**]);**

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

**}**

printf**(**"\n\nLa matriz inversa de la matriz del HIJO (Producto) es: \n\n"**);**

d **=** determinante**(**matrizP**,** 10**);**

**if** **(**d **==** 0**)**

printf**(**"\n Determinate = 0. No es posible calcular su matriz inversa \n\n"**);**

**else**

cofactor**(**matrizP**,** 10**);**

printf**(**"\n\nLa matriz inversa de la matriz del NIETO (Suma) es: \n\n"**);**

d **=** determinante**(**matrizS**,** 10**);**

**if** **(**d **==** 0**)**

printf**(**"\n Determinate = 0. No es posible calcular su matriz inversa \n"**);**

**else**

cofactor**(**matrizS**,** 10**);**

CloseHandle**(**hLecturaPipeIda**);**

CloseHandle**(**hEscrituraPipeIda**);**

CloseHandle**(**hLecturaPipeVuelta**);**

CloseHandle**(**hEscrituraPipeVuelta**);**

CloseHandle**(**hLecturaPipeVueltaN**);**

CloseHandle**(**hEscrituraPipeVueltaN**);**

CloseHandle**(**piHijo**.**hThread**);**

CloseHandle**(**piHijo**.**hProcess**);**

**return** 0**;**

**} //Fin de la función principal**

float determinante**(**float a**[**10**][**10**],** float k**){**

float s **=** 1**,** det **=** 0**,** b**[**10**][**10**];**

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

**if** **(**k **==** 1**){**

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

**}else{**

det **=** 0**;**

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

m **=** 0**;**

n **=** 0**;**

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

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

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

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

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

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

n**++;**

**else{**

n **=** 0**;**

m**++;**

**}**

**}**

**}**

**}**

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

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

**}**

**}**

**return** **(**det**);**

**}**

void cofactor**(**float num**[**10**][**10**],** float f**){**

float b**[**10**][**10**],** fac**[**10**][**10**];**

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**)** **\*** determinante**(**b**,** f **-** 1**);**

**}**

**}**

traspuesta**(**num**,** fac**,** f**);**

**}**

//encontrando la Mat. traspuesta

void traspuesta**(**float num**[**10**][**10**],** float fac**[**10**][**10**],** float r**){**

int i**,** j**;**

float b**[**10**][**10**],** inversa**[**10**][**10**],** d**;**

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

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

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

**}**

**}**

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

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

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

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

**}**

**}**

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

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

printf**(**"%10.4f"**,** inversa**[**i**][**j**]);**

**}**

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

**}**

static int c**=**1**;**

char nombre**[**20**];**

sprintf**(**nombre**,** "Inversa\_%d.txt"**,** c**++);**

guardaMatriz**(**inversa**,** nombre**);**

**}**

void guardaMatriz**(**float matriz**[**10**][**10**],** char**\*** nombreA**){**

int i**,**j**,** n**=**10**;**

FILE**\*** fichero **=** fopen**(**nombreA**,** "wt"**);**

**if(**fichero **==** **NULL){**

exit**(**0**);**

**}**

char buffer**[**15**];**

char linea**[**120**];**

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

linea**[**0**]** **=** '\0'**;**

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

sprintf**(**buffer**,** "%10.4f "**,** matriz**[**i**][**j**]);**

strcat**(**linea**,** buffer**);**

**}**

int len **=** strlen**(**linea**);**

linea**[**len **-** 1**]** **=** '\n'**;**

fputs**(**linea**,** fichero**);**

**}**

fclose**(**fichero**);**

**return;**

**}**

**Programa Hijo (HMatrizTub.c)**

#include <windows.h>

#include <stdio.h>

#include <string.h>

**typedef** struct argumentos**{**

int matrizA**[**10**][**10**];**

int matrizB**[**10**][**10**];**

HANDLE hEscrituraPipe**;**

HANDLE hLecturaPipe**;**

**}\***datos**;**

int main **()** **{**

int i**,**j**,**k**,** n**=**10**;**

float suma**;**

float matrizP**[**10**][**10**];**

datos datosHijo **=** **(**datos**)**malloc**(sizeof(**struct argumentos**));**

datos datosNieto **=** **(**datos**)**malloc**(sizeof(**struct argumentos**));**

DWORD leidos**,** escritos**;**

HANDLE hStdIn **=** GetStdHandle**(**STD\_INPUT\_HANDLE**);**

HANDLE hStdOut **=** GetStdHandle**(**STD\_OUTPUT\_HANDLE**);**

HANDLE hLecturaPipeIdaN**,** hEscrituraPipeIdaN**;**

SECURITY\_ATTRIBUTES pipeSeg **=**

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

int matrizA**[**10**][**10**]** **={ {**4**,**6**,**1**,**0**,**9**,**6**,**4**,**4**,**3**,**2**},** **{**5**,**8**,**2**,**5**,**8**,**8**,**9**,**6**,**6**,**3**},**

**{**6**,**9**,**1**,**7**,**9**,**0**,**0**,**7**,**8**,**2**},**

**{**4**,**0**,**1**,**8**,**7**,**7**,**7**,**9**,**9**,**3**},**

**{**8**,**3**,**4**,**0**,**5**,**2**,**8**,**8**,**9**,**4**},**

**{**3**,**3**,**3**,**3**,**1**,**7**,**3**,**3**,**8**,**8**},**

**{**2**,**6**,**2**,**3**,**3**,**9**,**4**,**4**,**6**,**9**},**

**{**1**,**0**,**4**,**1**,**8**,**4**,**5**,**5**,**4**,**7**},**

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

**{**1**,**3**,**5**,**4**,**0**,**9**,**3**,**1**,**1**,**5**}};**

int matrizB**[**10**][**10**]** **={ {**5**,**2**,**1**,**5**,**9**,**6**,**8**,**2**,**2**,**5**},**

**{**2**,**3**,**4**,**4**,**4**,**7**,**9**,**2**,**3**,**2**},**

**{**0**,**5**,**6**,**1**,**4**,**0**,**0**,**7**,**4**,**5**},**

**{**9**,**7**,**7**,**3**,**6**,**9**,**5**,**8**,**4**,**7**},**

**{**9**,**5**,**8**,**8**,**4**,**3**,**5**,**8**,**5**,**8**},**

**{**2**,**7**,**9**,**6**,**8**,**4**,**3**,**6**,**5**,**5**},**

**{**2**,**9**,**9**,**5**,**3**,**6**,**4**,**3**,**6**,**0**},**

**{**6**,**0**,**5**,**9**,**5**,**2**,**7**,**2**,**6**,**0**},**

**{**4**,**9**,**3**,**0**,**7**,**3**,**6**,**8**,**7**,**3**},**

**{**3**,**8**,**4**,**5**,**0**,**4**,**0**,**9**,**7**,**2**}};**

**for(**i**=**0**;**i**<**10**;**i**++){**//Guarda las matrices en la estructura

**for(**j**=**0**;**j**<**10**;**j**++){**

datosNieto**->**matrizA**[**i**][**j**]** **=** matrizA**[**i**][**j**];**

datosNieto**->**matrizB**[**i**][**j**]** **=** matrizB**[**i**][**j**];**

**}**

**}**

//Creacion y escritura en tuberia de IDA al nieto

CreatePipe **(&**hLecturaPipeIdaN**,** **&**hEscrituraPipeIdaN**,** **&**pipeSeg**,** 0**);**

WriteFile**(**hEscrituraPipeIdaN**,** datosNieto**,** **sizeof(**struct argumentos**),** **&**escritos**,** **NULL);**

//Lectura desde la tubería sin nombre del padre

ReadFile**(**hStdIn**,** datosHijo**,** **sizeof(**struct argumentos**),** **&**leidos**,** **NULL);**

//Creación del proceso Nieto

PROCESS\_INFORMATION piNieto**;**

STARTUPINFO siNieto**;**

GetStartupInfo **(&**siNieto**);**

//Modifica las Entradas y Salidas del Nieto

siNieto**.**hStdInput **=** hLecturaPipeIdaN**;**

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

//Salida hacia el padre

siNieto**.**hStdOutput **=** datosHijo**->**hEscrituraPipe**;**

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

//Crea al proceso nieto

CreateProcess **(NULL,** "NMatrizTub.exe"**,NULL,NULL,**TRUE**,**0**,NULL,NULL,** **&**siNieto**,** **&**piNieto**);**

WaitForSingleObject **(**piNieto**.**hProcess**,** INFINITE**);**

//Multiplicacion de matrices

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

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

suma **=** 0**;**

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

suma **+=** datosHijo**->**matrizA**[**j**][**k**]** **\*** datosHijo**->**matrizB**[**k**][**i**];**

**}**

matrizP**[**j**][**i**]** **=** suma**;**

**}**

**}**

//Envia la matriz producto al padre

WriteFile**(**hStdOut**,** matrizP**,** **sizeof(**matrizP**),** **&**escritos**,** **NULL);**

CloseHandle(hLecturaPipeIdaN);

CloseHandle(hEscrituraPipeIdaN);

CloseHandle**(**hStdIn**);**

CloseHandle**(**hStdOut**);**

**return** 0**;**

**}**

**Proceso Nieto (NMatrizTub.c)**

#include <windows.h>

#include <stdio.h>

**typedef** struct argumentos**{**

int matrizA**[**10**][**10**];**

int matrizB**[**10**][**10**];**

HANDLE hEscrituraPipe**;**

HANDLE hLecturaPipe**;**

**}\***datos**;**

int main **()** **{**

float matrizSuma**[**10**][**10**];**

char mensaje**[**20**];**

datos datosNieto **=** **(**datos**)**malloc**(sizeof(**struct argumentos**));**

DWORD leidos**,** escritos**;**

HANDLE hStdIn **=** GetStdHandle**(**STD\_INPUT\_HANDLE**);**

HANDLE hStdOut **=** GetStdHandle**(**STD\_OUTPUT\_HANDLE**);**

SECURITY\_ATTRIBUTES pipeSeg **=**

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

// Lectura desde la tubería sin nombre del Hijo

ReadFile**(**hStdIn**,**datosNieto**,sizeof(**struct argumentos**),&**leidos**,** **NULL);**

//Suma de Matrices

int i**,**j**,**n**=**10**;**

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

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

matrizSuma**[**i**][**j**]** **=** datosNieto**->**matrizA**[**i**][**j**] +** datosNieto**->**matrizB**[**i**][**j**];**

**}**

**}**

//Envía la Matriz Suma por la tubería hacia el Padre

WriteFile**(**hStdOut**,** matrizSuma**,sizeof(**matrizSuma**),&**escritos**,** **NULL);**

CloseHandle**(**hStdIn**);**

CloseHandle**(**hStdOut**);**

**return** 0**;**

**}**

* **Ejecución**

Imagen que contiene Calendario

Descripción generada automáticamente

**Calendario

Descripción generada automáticamente con confianza media**

**Tabla

Descripción generada automáticamente**

**Tabla

Descripción generada automáticamente**

**Comprobación**

**Matriz inversa de la Matriz Producto**

**Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente**

**Matriz Inversa de la Matriz Suma**

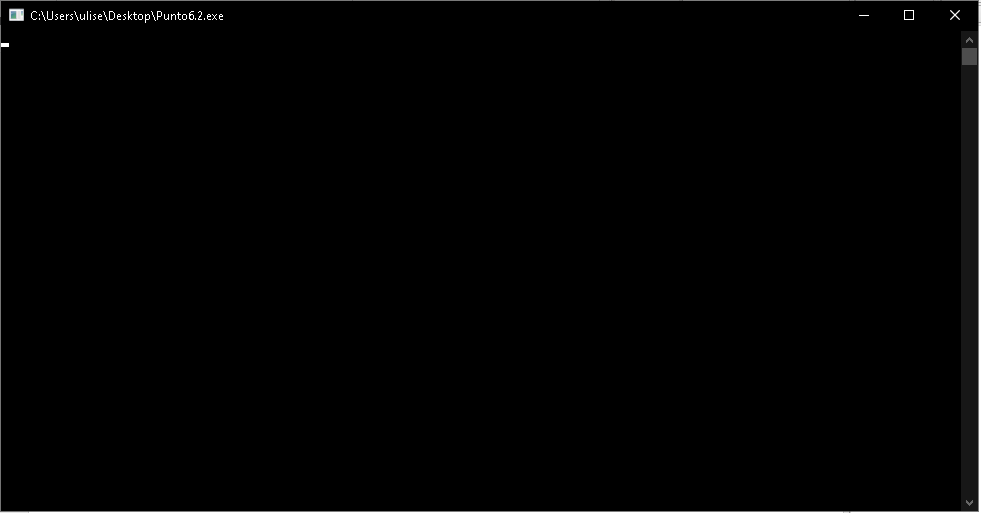
**Interfaz de usuario gráfica

Descripción generada automáticamente**

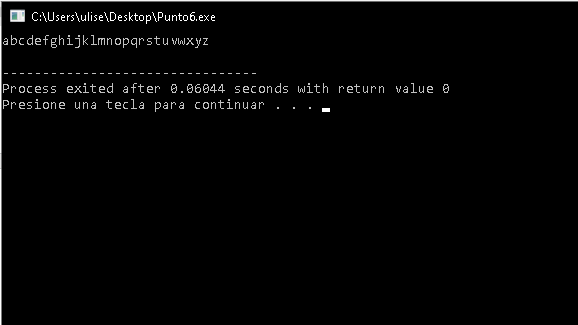
6. Capture, compile y ejecute los siguientes programas para Windows. Observe su funcionamiento.

**Capturas de la ejecución**

Ejecución del servidor de la memoria compartida



Ejecución del cliente de memoria compartida



**Observaciones**

Primero se ejecuto el servidor como se mencionó en la clase y después al cliente, cuando se ejecuta el servidor no muestra nada, lo cual significa que se ejecutó correctamente, después se ejecuta el cliente y nos muestra el abecedario que es lo que debe ejecutarse. Note que si se ejecuta primero el cliente nos muestra el mensaje de que “no se abrió el archivo de mapeo de la memoria compartida”

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.

* **Para Linux**

**servidor**

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include<sys/wait.h>

#define **TAM\_MEM** 27

int mat1[10][10]={{1,0,0,0,0,1,1,0,0,0},

                  {1,2,0,0,2,0,0,0,0,0},

                  {1,1,1,2,0,1,0,1,6,0},

                  {1,2,3,2,0,1,0,0,0,1},

                  {1,2,3,0,5,0,2,0,1,2},

                  {1,2,0,1,5,6,0,0,0,1},

                  {1,2,0,4,5,6,7,0,2,0},

                  {1,2,0,4,5,6,7,8,0,2},

                  {1,2,1,4,5,1,7,8,9,0},

                  {1,2,2,4,5,6,7,8,9,10}};

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

                  {0,9,8,7,6,5,4,3,2,1},

                  {0,4,8,7,6,5,4,3,2,1},

                  {2,0,2,7,6,5,4,3,2,1},

                  {0,1,1,0,6,5,4,3,2,1},

                  {0,2,0,0,0,5,4,3,2,1},

                  {0,3,2,0,4,0,4,3,2,1},

                  {0,4,0,0,1,0,0,3,2,1},

                  {0,0,0,4,1,1,0,0,2,1},

                  {1,1,0,2,0,1,1,2,0,1}};

float mat1inv[10][10]={};

float mat2inv[10][10]={};

float multi[40][40]={};

float suma[40][40]={};

void **inversa**(float matrix[][40], int order,int opc){

    float temp;

    for (int i = 0; i < order; i++) {

        for (int j = 0; j < 2 \* order; j++) {

            if (j == (i + order))

                matrix[i][j] = 1;

        }

    }

    for (int i = order - 1; i > 0; i--) {

        if (matrix[i - 1][0] < matrix[i][0])

            for (int j = 0; j < 2 \* order; j++) {

                temp = matrix[i][j];

                matrix[i][j] = matrix[i - 1][j];

                matrix[i - 1][j] = temp;

            }

    }

    for (int i = 0; i < order; i++) {

        for (int j = 0; j < 2 \* order; j++) {

            if (j != i) {

                temp = matrix[j][i] / matrix[i][i];

                for (int k = 0; k < 2 \* order; k++) {

                    matrix[j][k] -= matrix[i][k] \* temp;

                }

            }

        }

    }

    for (int i = 0; i < order; i++) {

        temp = matrix[i][i];

        for (int j = 0; j < 2 \* order; j++) {

            matrix[i][j] = matrix[i][j] / temp;

        }

    }

    for (int i = 0; i < order; i++) {

        for (int j = order; j < 2 \* order; j++) {

            if(opc==0)

             mat1inv[i][j-order]=matrix[i][j];

            else

             mat2inv[i][j-order]=matrix[i][j];

        }

    }

    return;

}

int **main**(){

  char c;

  int shmid;

  int shmid2;

  int shmid3;

  key\_t llave;

  int \*shm, \*s;

  int \*shm2, \*s2;

  int \*shm3, \*s3;

  int i,j,k;

  float \*\*m1, \*\*m2;

  llave = 5678;

  if((shmid = **shmget**(llave, TAM\_MEM, IPC\_CREAT | 0666)) < 0){

**perror**("1 Error al obtener memoria compartida 1: shmget");

**exit**(-1);

  }

  if((shm = **shmat**(shmid,NULL,0)) == (int \*)-1){

**perror**("Error al enlazar la memoria compartida 1: shmat");

**exit**(-1);

  }

  llave = 5679;

  if((shmid2 = **shmget**(llave, TAM\_MEM,IPC\_CREAT | 0666)) <0){

**perror**("Error al obtener memoria compartida 2: shmget"); **exit**(-1);

  }

  if((shm2 = **shmat**(shmid2,NULL,0)) == (int\*)-1){

**perror**("Error al enlazar la memoria compartida 2: shmat");

**exit**(-1);

  }

  llave = 5681;

  if((shmid3 = **shmget**(llave, TAM\_MEM, IPC\_CREAT | 0666)) <0){

**perror**("Error al obtener memoria compartida 3: shmget"); **exit**(-1);

  }

  if((shm3 = **shmat**(shmid3,NULL,0)) == (int\*)-1){

**perror**("Error al enlazar la memoria compartida 3: shmat");

**exit**(-1);

  }

*//envia matriz*

  s = shm;

  for(i = 0,k = 0; i < 10; i++){

    for(j = 0; j < 10; j++){

      s[k++] = mat1[i][j];

    }

  }

  for(i = 0; i < 10; i++){

    for(j = 0; j < 10; j++){

      s[k++] = mat2[i][j];

    }

  }

  while(\*shm != -10)

**sleep**(1);

*//recibe multiplicacion*

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

  s2 = shm2;

  for(i = 0, k = 0; i < 10; i++){

    for(j = 0; j < 10; j++){

      multi[i][j] = s2[k++];

**printf**("%.0f ",multi[i][j]);

    }

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

  }

  \*shm2=-10;

*//recibe suma*

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

  s3 = shm3;

  for(i = 0, k = 0; i < 10; i++){

    for(j = 0; j < 10; j++){

      suma[i][j] = s3[k++];

**printf**("%.0f ",suma[i][j]);

    }

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

  }

  \*shm3=-10;

FILE\* fichero;

      fichero = **fopen**("InversaMult.txt", "w");

FILE\* fichero2;

      fichero2 = **fopen**("InversaSuma.txt", "w");

**inversa**(multi,10,0);

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

  for(int i=0; i<10; i++) {

    for(int j=0; j<10; j++) {

**printf**("%.2f ", (float)(mat1inv[i][j]));

**fprintf**(fichero, "%.2f  ",mat1inv[i][j]);

    }

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

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

  }

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

**inversa**(suma,10,1);

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

  for(int i=0; i<10; i++) {

    for(int j=0; j<10; j++) {

**printf**("%.3f ", (float)(mat2inv[i][j]));

**fprintf**(fichero2, "%.2f ",mat2inv[i][j]);

    }

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

**fprintf**(fichero2,"\n");

  }

**fclose**(fichero);

**fclose**(fichero2);

**exit**(0);

}

**Clientes**

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include<sys/wait.h>

#define **TAM\_MEM** 27

int **main**(){

  char c;

  int shmid;

  int shmid2;

  int shmid3;

  key\_t llave;

  int \*shm, \*s;

  int \*shm2, \*s2;

  int \*shm3, \*s3;

  int \*\*mat, \*\*mat2, \*\*sum,\*\*mat3, \*\*mat4;

  int i, j, k;

  llave = 5678;

  if((shmid = **shmget**(llave,TAM\_MEM, 0666)) <0){

**perror**("Error al obtener memoria compartida: shmget"); **exit**(-1);

  }

  if((shm = **shmat**(shmid,NULL,0)) == (int\*)-1){

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

**exit**(-1);

  }

  llave = 5679;

  if((shmid2 = **shmget**(llave,TAM\_MEM,IPC\_CREAT | 0666)) <0){

**perror**("Error al obtener memoria compartida: shmget"); **exit**(-1);

  }

  if((shm2 = **shmat**(shmid2,NULL,0)) == (int\*)-1){

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

**exit**(-1);

  }

  llave = 5680;

  if((shmid3 = **shmget**(llave, TAM\_MEM, IPC\_CREAT | 0666)) <0){

**perror**("Error al obtener memoria compartida: shmget"); **exit**(-1);

  }

  if((shm3 = **shmat**(shmid3,NULL,0)) == (int\*)-1){

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

**exit**(-1);

  }

  int mult1[10][10]={};

  int mult2[10][10]={};

  float multRes[10][10]={};

*//recibe matrices*

  s = shm;

  for(i = 0, k =0; i < 10; i++){

    for(j = 0; j < 10; j++){

      mult1[i][j] = s[k++];

    }

  }

  for(i = 0; i < 10; i++){

    for(j = 0; j < 10; j++){

      mult2[i][j] = s[k++];

    }

  }

  \*shm=-10;

  s3 = shm3;

  for(i = 0,k = 0; i < 10; i++){

    for(j = 0; j < 10; j++){

      s3[k++] = mult1[i][j];

    }

  }

  for(i = 0; i < 10; i++){

    for(j = 0; j < 10; j++){

      s3[k++] = mult2[i][j];

    }

  }

  while(\*shm3 != -10)

**sleep**(1);

*//envia matriz resultado*

  s2 = shm2;

  int z=0;

  for (i = 0, z = 0; i < 10; i++){

    for (j = 0; j < 10; j++){

        s2[z]=0;

        for (k = 0; k < 10; k++)

            s2[z]+=mult1[i][k]\*mult2[k][j];

        z++;

    }

  }

  while(\*shm2 != -10)

**sleep**(1);

**exit**(0);

}

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include<sys/wait.h>

#define **TAM\_MEM** 27

int **main**(){

  char c;

  int shmid;

  int shmid2;

  int shmid3;

  key\_t llave;

  int \*shm, \*s;

  int \*shm2, \*s2;

  int \*shm3, \*s3;

  int i, j, k;

  int suma1[10][10]={};

  int suma2[10][10]={};

  float sumaRes[10][10]={};

  llave = 5680;

  if((shmid = **shmget**(llave,TAM\_MEM, IPC\_CREAT |0666)) <0){

**perror**("Error al obtener memoria compartida: shmget");

**exit**(-1);

  }

  if((shm = **shmat**(shmid,NULL,0)) == (int\*)-1){

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

**exit**(-1);

  }

  llave = 5681;

  if((shmid3= **shmget**(llave, TAM\_MEM, IPC\_CREAT | 0666)) <0){

**perror**("Error al obtener memoria compartida: shmget"); **exit**(-1);

  }

  if((shm3 = **shmat**(shmid3,NULL,0)) == (int\*)-1){

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

**exit**(-1);

  }

*//recibe matrices*

  s = shm;

  for(i = 0, k =0; i < 10; i++){

    for(j = 0; j < 10; j++){

      suma1[i][j] = s[k++];

    }

  }

  for(i = 0; i < 10; i++){

    for(j = 0; j < 10; j++){

      suma2[i][j] = s[k++];

    }

  }

  \*shm=-10;

*//envia matriz resultado*

  s3 = shm3;

  int z=0;

  for (i = 0, z = 0; i < 10; i++){

    for (j = 0; j < 10; j++){

        s3[z]=suma1[i][j]+suma2[i][j];

        z++;

    }

  }

  while(\*shm3 != -10)

**sleep**(1);

**exit**(0);

}

Texto

Descripción generada automáticamente

Imagen que contiene electrónica, teclado, tabla, computadora

Descripción generada automáticamente Patrón de fondo

Descripción generada automáticamente

Imagen que contiene Calendario

Descripción generada automáticamente

Imagen que contiene Gráfico

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

* **Para Windows**

**Proceso padre (PMatMemC.c)**

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

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

float determinante**(**float **[][**10**],** float**);**

void cofactor**(**float **[][**10**],** float**);**

void traspuesta**(**float **[][**10**],** float **[][**10**],** float**);**

void guardaMatriz**(**float **[][**10**],** char**\*** **);**

**typedef** struct argumentos**{**

int matrizA**[**10**][**10**];**

int matrizB**[**10**][**10**];**

float matrizR**[**10**][**10**];**

**}**estructura**,** **\***datos**;**

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

float matriz**[**10**][**10**];**

HANDLE hArchMapeoHijo**,** hArchMapeoNieto**;**

int i**,**j**;** //Contadores

float d**;** //Determinante

char **\***IdMemCompP\_Hijo **=** "MemCompP\_Hijo"**;**

char **\***IdMemCompP\_Nieto **=** "MemCompP\_Nieto"**;**

STARTUPINFO siHijo**;**

PROCESS\_INFORMATION piHijo**;**

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

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

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

datos datosHijo **=** **(**datos**)**malloc**(sizeof(**estructura**));**

datos datosNieto **=** **(**datos**)**malloc**(sizeof(**estructura**));**

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

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

//Creación de Memoria COMPARTIDA CON EL HIJO

**if((**hArchMapeoHijo**=**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

**sizeof(**estructura**),** // tamaño maxixmo parte baja de un DWORD

IdMemCompP\_Hijo**)** // identificador de la memoria compartida

**)** **==** **NULL)** **{**

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

exit**(-**1**);**

**}**

**if((**datosHijo**=(**datos**)**MapViewOfFile**(**hArchMapeoHijo**,** // Manejador del mapeo

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

0**,**

0**,**

**sizeof(**estructura**)))** **==** **NULL)** **{**

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

CloseHandle**(**hArchMapeoHijo**);**

exit**(-**1**);**

**}** //Fin de la creación de MemCompHijo

//Creación de Memoria COMPARTIDA CON EL NIETO

**if((**hArchMapeoNieto**=**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

**sizeof(**estructura**),** // tamaño maxixmo parte baja de un DWORD

IdMemCompP\_Nieto**)** // identificador de la memoria compartida

**)** **==** **NULL)** **{**

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

exit**(-**1**);**

**}**

**if((**datosNieto**=(**datos**)**MapViewOfFile**(**hArchMapeoNieto**,** // Manejador del mapeo

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

0**,**

0**,**

**sizeof(**estructura**)))** **==** **NULL)** **{**

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

CloseHandle**(**hArchMapeoNieto**);**

exit**(-**1**);**

**}** //Fin de la creación de MemCompNieto

**for(**i**=**0**;**i**<**10**;**i**++){**//Escribe las matrices en la memoria compartida

**for(**j**=**0**;**j**<**10**;**j**++){**

datosHijo**->**matrizA**[**i**][**j**]** **=** matrizA**[**i**][**j**];**

datosHijo**->**matrizB**[**i**][**j**]** **=** matrizB**[**i**][**j**];**

**}**

**}**

//Creación del PROCESO HIJO

**if(!**CreateProcess **(NULL,** argv[1]**,NULL,NULL,**FALSE**,**0**,NULL,NULL,** **&**siHijo**,** **&**piHijo**)){**

printf**(**"Error al crear el proceso Hijo"**);**

exit**(-**1**);**

**}**

WaitForSingleObject **(**piHijo**.**hProcess**,** INFINITE**);**

//Lee la matriz enviada por el Hijo

printf**(**"\nLa matriz devuelta por del HIJO (Producto) es: \n\n"**);**

**for(**i**=**0**;**i**<**10**;**i**++){**

**for(**j**=**0**;**j**<**10**;**j**++)**

printf**(**"%4.0f "**,** datosHijo**->**matrizR**[**i**][**j**]);**

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

**}**

//Lee la matriz enviada por el Nieto

printf**(**"\n\nLa matriz devuelta por del NIETO (Suma) es: \n\n"**);**

**for(**i**=**0**;**i**<**10**;**i**++){**

**for(**j**=**0**;**j**<**10**;**j**++)**

printf**(**"%4.0f "**,** datosNieto**->**matrizR**[**i**][**j**]);**

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

**}**

//Matrices Inversas

printf**(**"\n\nLa matriz inversa de la matriz del HIJO (Producto) es: \n\n"**);**

d **=** determinante**(**datosHijo**->**matrizR**,** 10**);**

**if** **(**d **==** 0**)**

printf**(**"\n Determinate = 0. No es posible calcular su matriz inversa \n\n"**);**

**else**

cofactor**(**datosHijo**->**matrizR**,** 10**);**

printf**(**"\n\nLa matriz inversa de la matriz del NIETO (Suma) es: \n\n"**);**

d **=** determinante**(**datosNieto**->**matrizR**,** 10**);**

**if** **(**d **==** 0**)**

printf**(**"\n Determinate = 0. No es posible calcular su matriz inversa \n"**);**

**else**

cofactor**(**datosNieto**->**matrizR**,** 10**);**

UnmapViewOfFile**(**datosHijo**);**

UnmapViewOfFile**(**datosNieto**);**

CloseHandle**(**hArchMapeoHijo**);**

CloseHandle**(**hArchMapeoNieto**);**

CloseHandle**(**piHijo**.**hThread**);**

CloseHandle**(**piHijo**.**hProcess**);**

exit**(**0**);**

**}**

float determinante**(**float a**[**10**][**10**],** float k**){**

float s **=** 1**,** det **=** 0**,** b**[**10**][**10**];**

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

**if** **(**k **==** 1**){**

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

**}else{**

det **=** 0**;**

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

m **=** 0**;**

n **=** 0**;**

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

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

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

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

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

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

n**++;**

**else{**

n **=** 0**;**

m**++;**

**}**

**}**

**}**

**}**

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

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

**}**

**}**

**return** **(**det**);**

**}**

void cofactor**(**float num**[**10**][**10**],** float f**){**

float b**[**10**][**10**],** fac**[**10**][**10**];**

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**)** **\*** determinante**(**b**,** f **-** 1**);**

**}**

**}**

traspuesta**(**num**,** fac**,** f**);**

**}**

//encontramdo la Mat. traspuesta

void traspuesta**(**float num**[**10**][**10**],** float fac**[**10**][**10**],** float r**){**

int i**,** j**;**

float b**[**10**][**10**],** inversa**[**10**][**10**],** d**;**

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

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

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

**}**

**}**

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

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

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

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

**}**

**}**

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

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

printf**(**"%10.4f"**,** inversa**[**i**][**j**]);**

**}**

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

**}**

static int c**=**1**;**

char nombre**[**20**];**

sprintf**(**nombre**,** "Inversa\_MemComp\_%d.txt"**,** c**++);**

guardaMatriz**(**inversa**,** nombre**);**

**}**

void guardaMatriz**(**float matriz**[**10**][**10**],** char**\*** nombreA**){**

int i**,**j**,** n**=**10**;**

FILE**\*** fichero **=** fopen**(**nombreA**,** "wt"**);**

**if(**fichero **==** **NULL){**

exit**(**0**);**

**}**

char buffer**[**15**];**

char linea**[**120**];**

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

linea**[**0**]** **=** '\0'**;**

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

sprintf**(**buffer**,** "%10.4f "**,** matriz**[**i**][**j**]);**

strcat**(**linea**,** buffer**);**

**}**

int len **=** strlen**(**linea**);**

linea**[**len **-** 1**]** **=** '\n'**;**

fputs**(**linea**,** fichero**);**

**}**

fclose**(**fichero**);**

**return;**

**}**

**Proceso Hijo (HMatMemC.c)**

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

#include <stdio.h>

**typedef** struct argumentos**{**

int matrizA**[**10**][**10**];**

int matrizB**[**10**][**10**];**

float matrizR**[**10**][**10**];**

**}**estructura**,** **\***datos**;**

int main**(**void**){**

int i**,**j**,**k**,** n**=**10**;**

float suma**;**

HANDLE hArchMapeoHijo**,** hArchMapeoNieto**;**

char **\***idMemCompP\_Hijo **=** "MemCompP\_Hijo"**;**

char **\***idMemCompH\_Nieto **=** "MemCompH\_Nieto"**;**

STARTUPINFO siNieto**;**

PROCESS\_INFORMATION piNieto**;**

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

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

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

datos datosHijo **=** **(**datos**)**malloc**(sizeof(**estructura**));**

datos datosNieto **=** **(**datos**)**malloc**(sizeof(**estructura**));**

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

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

//Creación de MEMORIA COMPARTIDA HIJO-NIETO

**if((**hArchMapeoNieto**=**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

**sizeof(**estructura**),** // tamaño maxixmo parte baja de un DWORD

idMemCompH\_Nieto**)** // identificador de la memoria compartida

**)** **==** **NULL)** **{**

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

exit**(-**1**);**

**}**

**if((**datosNieto**=(**datos**)**MapViewOfFile**(**hArchMapeoNieto**,** // Manejador del mapeo

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

0**,**

0**,**

**sizeof(**estructura**)))** **==** **NULL)** **{**

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

CloseHandle**(**hArchMapeoNieto**);**

exit**(-**1**);**

**}** //Fin de la creación de MemCompNieto

**for(**i**=**0**;**i**<**10**;**i**++){**//Escribe las matrices en la memoria compartida

**for(**j**=**0**;**j**<**10**;**j**++){**

datosNieto**->**matrizA**[**i**][**j**]** **=** matrizA**[**i**][**j**];**

datosNieto**->**matrizB**[**i**][**j**]** **=** matrizB**[**i**][**j**];**

**}**

**}**

//Creacion del PROCESO NIETO

**if(!**CreateProcess **(NULL,** "NMatMemC.exe"**,NULL,NULL,**FALSE**,**0**,NULL,NULL,** **&**siNieto**,** **&**piNieto**)){**

printf**(**"Error al crear el proceso Nieto"**);**

exit**(-**1**);**

**}**

//Abre memoria COMPARTIDA CON EL PADRE

**if((**hArchMapeoHijo**=**OpenFileMapping**(**

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

FALSE**,** // no se hereda el nombre

idMemCompP\_Hijo**)** // identificador de la memoria compartida

**)** **==** **NULL)** **{**

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

exit**(-**1**);**

**}**

**if((**datosHijo**=(**datos**)**MapViewOfFile**(**hArchMapeoHijo**,** // Manejador del mapeo

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

0**,**

0**,**

**sizeof(**estructura**)))** **==** **NULL)** **{**

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

CloseHandle**(**hArchMapeoHijo**);**

exit**(-**1**);**

**}**//Fin del proceso de apertura de compartida con el padre

//Multiplica las matrices y escribe el resultado en la memoria compartida

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

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

suma **=** 0**;**

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

suma **+=** datosHijo**->**matrizA**[**j**][**k**]** **\*** datosHijo**->**matrizB**[**k**][**i**];**

**}**

datosHijo**->**matrizR**[**j**][**i**]** **=** suma**;**

**}**

**}**

WaitForSingleObject **(**piNieto**.**hProcess**,** INFINITE**);**

UnmapViewOfFile**(**datosHijo**);**

UnmapViewOfFile**(**datosNieto**);**

CloseHandle**(**hArchMapeoHijo**);**

CloseHandle**(**hArchMapeoNieto**);**

CloseHandle**(**piNieto**.**hThread**);**

CloseHandle**(**piNieto**.**hProcess**);**

exit**(**0**);**

**}**

**Proceso Nieto (NMatMemC.c)**

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

#include <stdio.h>

**typedef** struct argumentos**{**

int matrizA**[**10**][**10**];**

int matrizB**[**10**][**10**];**

float matrizR**[**10**][**10**];**

**}**estructura**,** **\***datos**;**

int main**(**void**){**

int i**,**j**,**k**,** n**=**10**;**

HANDLE hArchMapeoHijo**,**hArchMapeoNieto**;**

char **\***idMemCompP\_Nieto **=** "MemCompP\_Nieto"**;**

char **\***idMemCompH\_Nieto **=** "MemCompH\_Nieto"**;**

datos datosHijo **=** **(**datos**)**malloc**(sizeof(**estructura**));**

datos datosNieto **=** **(**datos**)**malloc**(sizeof(**estructura**));**

datos datosAuxHijo **=** **(**datos**)**malloc**(sizeof(**estructura**));**

//Abre la memoria COMPARTIDA HIJO-NIETO

**if((** hArchMapeoHijo **=**OpenFileMapping**(**

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

FALSE**,** // no se hereda el nombre

idMemCompH\_Nieto**)** // identificador de la memoria compartida

**)** **==** **NULL)** **{**

printf**(**"No se 1abrio archivo de mapeo de la memoria compartida: (%i)\n"**,** GetLastError**());**

exit**(-**1**);**

**}**

**if((**datosHijo**=(**datos**)**MapViewOfFile**(**hArchMapeoHijo**,** // Manejador del mapeo

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

0**,**

0**,**

**sizeof(**estructura**)))** **==** **NULL)** **{**

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

CloseHandle**(**hArchMapeoHijo**);**

exit**(-**1**);**

**}**//Fin de abrir la memoria conpartida con el Hijo

//Abre la memoria COMPARTIDA PADRE-NIETO

**if((**hArchMapeoNieto**=**OpenFileMapping**(**

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

FALSE**,** // no se hereda el nombre

idMemCompP\_Nieto**)** // identificador de la memoria compartida

**)** **==** **NULL)** **{**

printf**(**"No se 2abrio archivo de mapeo de la memoria compartida: (%i)\n"**,** GetLastError**());**

exit**(-**1**);**

**}**

**if((**datosNieto**=(**datos**)**MapViewOfFile**(**hArchMapeoNieto**,** // Manejador del mapeo

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

0**,**

0**,**

**sizeof(**estructura**)))** **==** **NULL)** **{**

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

CloseHandle**(**hArchMapeoNieto**);**

exit**(-**1**);**

**}**//Fin de abrir la memoria compartida con el padre

//Suma de Matrices y la escribe en la memoria compartida con el padre

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

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

datosNieto**->**matrizR**[**i**][**j**]** **=** datosHijo**->**matrizA**[**i**][**j**]+**datosHijo**->**matrizB**[**i**][**j**];**

**}**

**}**

UnmapViewOfFile**(**datosNieto**);**

UnmapViewOfFile**(**datosHijo**);**

CloseHandle**(**hArchMapeoNieto**);**

CloseHandle**(**hArchMapeoHijo**);**

exit**(**0**);**

**}**

* **Ejecución**

**Calendario

Descripción generada automáticamente**

**Calendario

Descripción generada automáticamente con confianza media**

Se realizaron los cálculos con las mismas cuatro matrices para facilitar la evaluación de su correcto funcionamiento, y como se puede observar, se obtuvieron los mismos resultados:

**Tabla

Descripción generada automáticamenteTabla

Descripción generada automáticamente**

**Conclusión**

\*Algo sobre la comunicación interprocesos en general\*

En esta práctica en particular, estudiamos y realizamos la implementación de dos mecanismos de comunicación entre procesos: las tuberías (pipes) y la memoria compartida.

En primer lugar, usamos las tuberías para comunicar procesos en diferentes niveles en un árbol de procesos. En este punto, atendiendo a las practicas recomendadas, fue necesario crear una tubería para cada sentido de la comunicación entre procesos, es decir, una tubería de ida del proceso en el primer nivel hacia el segundo; otra del proceso en el segundo nivel hacia el proceso en el primero, otra más del proceso en el segundo nivel al tercero; y finalmente una del proceso en el tercer nivel hacia el primero. La creación y manejo de todas estas tuberías en ocasiones podía tornarse confuso, así que había que tener cuidado con los nombres de los diferentes manejadores. Para realizar las operaciones con matrices solicitadas en la práctica, fue necesario crear una estructura que almacenara estas matrices, y así poderlas enviar a través de la tubería. Además de esto, otra pequeña dificultad que se presentó fue pasarle el manejador de la tubería creada por el proceso del primer nivel al proceso del tercer nivel, esto lo solucionamos agregando un par de campos de tipo “handle”a la estructura de datos que enviábamos por la tubería, de manera que proceso padre enviara estos datos al hijo y posteriormente el hijo le hiciera llegar los datos al proceso nieto.

En la parte de memoria compartida observamos el funcionamiento con los algoritmos que se mencionan en la práctica. Observando el código se nos menciona que primero se bebe ejecutar el servidor de la memoria compartida y después el cliente, para Windows y Linux se debe hacer eso. Para Linux se debe especificar el tamaño de la memoria compartida y las llamadas al sistema son “shmget” y shmat, se nos menciona que uno es para obtener la memoria compartida y el otro es para enlazar. Para Windows es algo similar, se debe especificar el tamaño de la memoria compartida pero las llamadas al sistema son, CreateFileMapping y MapViewOfFile.

Para ambos (Windows y Linux), es necesario ejecutar primero el servidor antes que el cliente, ya que si no se hace de esta forma no se nos creara la memoria compartida.