****

**Sistemas Operativos**

**Práctica 7. Mecanismos de sincronización de procesos en Linux y Windows (semáforos)**

**Grupo: 2CM9**

**Equipo:**

* **Guerra Vargas Irving Cristóbal**
* **Reyes Valenzuela Alejandro**
* **Dávila Méndez Juan Manuel**

**Competencia.**

* **El alumno comprende el funcionamiento de los mecanismos de sincronización entre procesos cooperativos utilizando los semaforos como árbitros de acceso para el desarrollo de aplicaciones cooperativos tanto en el sistema operativo Linux como Windows.**

****

**Desarrollo:**

****

Código Padre

#include <windows.h> //Programa podre

#include <stdio.h>

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

STARTUPINFO si; //Estructura de información inicial para Windows

PROCESS\_INFORMATION pi; //Estructura de información para el adm. de procesos

HANDLE hSemaforo;

int i=1;

ZeroMemory(&si,sizeof(si));

si.cb=sizeof(si);

ZeroMemory(&pi,sizeof(pi));

if(argc!=2){

printf("Usar %s Nombre\_programa\_hijo\n");

return;

}

//Creación de semaforo

if((hSemaforo=CreateSemaphore(NULL,1,1,"Semaforo"))==NULL){

printf("Fallo al invocar CreateSemaphore: %d\n",GetLastError());

return -1;

}

//Creación proceso hijo

if(!CreateProcess(NULL,argv[1],NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){

printf("Fallo al invocar CreateProcess: %d\n",GetLastError());

return -1;

}

while(i<4){

//Prueba del semaforo

WaitForSingleObject(hSemaforo,INFINITE);

//Sección crítica

printf("Soy el padre entrando %i de 3 veces al semaforo\n",i);

Sleep(5000);

//Liberación de semáforo

if(!ReleaseSemaphore(hSemaforo,1,NULL)){

printf("Fallo al invocar ReleaseSemaphore: %d\n",GetLastError());

return -1;

}

printf("Soy el padre saliendo %i de 3 veces al semaforo\n",i);

Sleep(5000);

i++;

}

//Terminación controlada del proceso e hilo asociado de ejecución

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

}

Código Hijo

#include <windows.h> //Programa hijo

#include <stdio.h>

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

HANDLE hSemaforo;

int i=1;

//Apertura de semaforo

if((hSemaforo=OpenSemaphore(SEMAPHORE\_ALL\_ACCESS,FALSE,"Semaforo"))==NULL){

printf("Fallo al invocar OpenSemaphore: %d\n",GetLastError());

return -1;

}

while(i<4){

//Prueba del semaforo

WaitForSingleObject(hSemaforo,INFINITE);

//Sección crítica

printf("Soy el hijo entrando %i de 3 veces al semaforo\n",i);

Sleep(5000);

//Liberación de semáforo

if(!ReleaseSemaphore(hSemaforo,1,NULL)){

printf("Fallo al invocar ReleaseSemaphore: %d\n",GetLastError());

return -1;

}

printf("Soy el hijo liberando %i de 3 veces al semaforo\n",i);

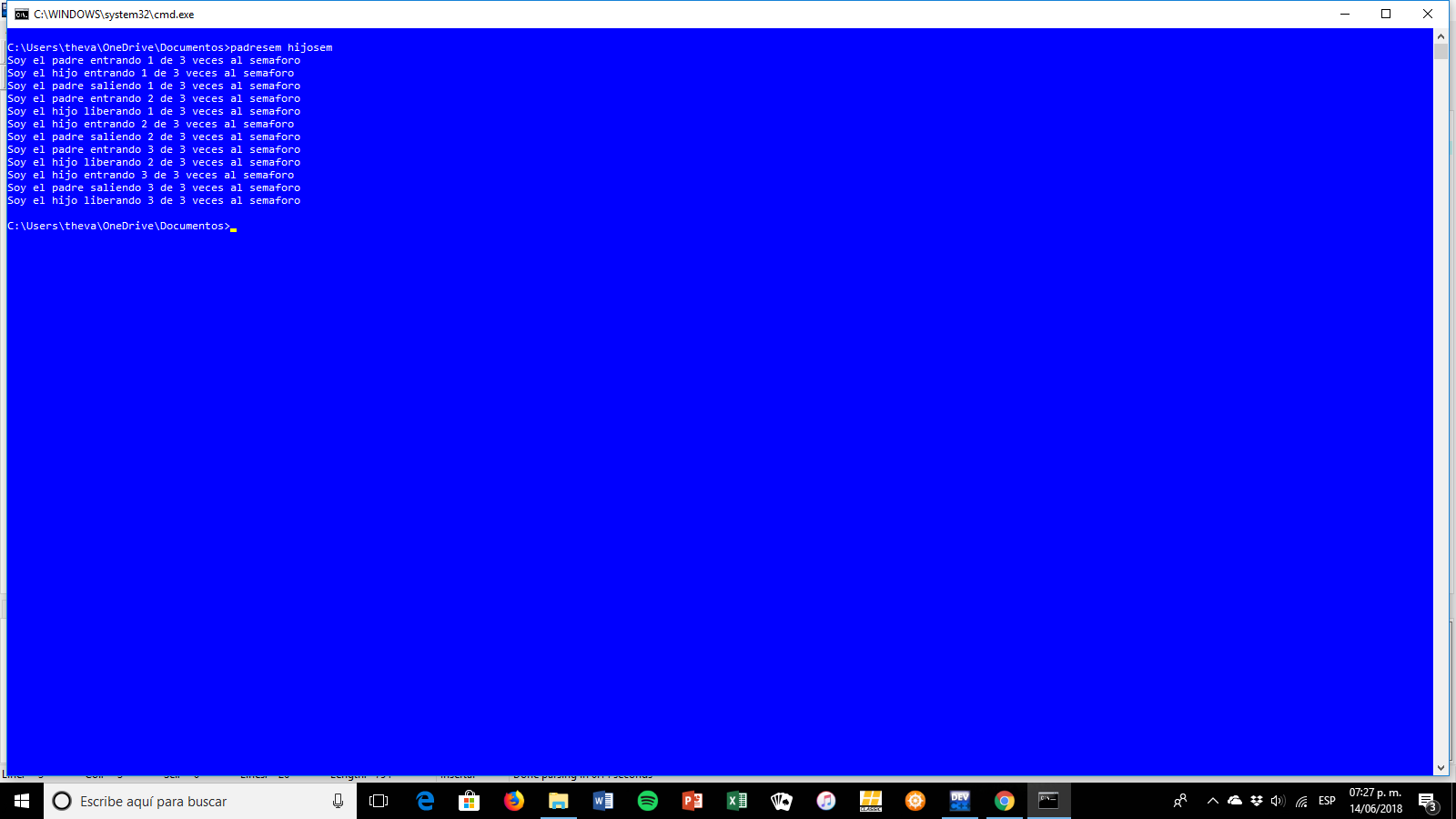
Sleep(5000);

i++;

}

}

Captura



Programa Matrices

#include <windows.h>

#include <stdio.h>

#include <time.h>

#include "matrix.c"

#define TAM\_MEM 400

//Escribimos sobre el texto las matrices

void writeFilex(matriz A, float \*p){

int i,j;

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

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

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

}

void readFilex(matriz A, float \*p){

int i,j;

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

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

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

}

float \*mem1(char \*idMem){

HANDLE hMem = CreateFileMapping(INVALID\_HANDLE\_VALUE, NULL, PAGE\_READWRITE, 0, TAM\_MEM, idMem);

return MapViewOfFile(hMem, FILE\_MAP\_ALL\_ACCESS, 0, 0, TAM\_MEM);

}

float \*mem2(char \*idMem){

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

return MapViewOfFile(hMem, FILE\_MAP\_ALL\_ACCESS, 0, 0, TAM\_MEM);

}

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

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, proc);

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

}

HANDLE hSem[5];

void operaciones(int i){

srand(time(NULL));

if(i == 0){

//Proceso abuelo

float \*p = mem1("memoria");

matriz A = matrizAleatoria(), B = matrizAleatoria();

matriz AB = nueva(), C\_D = nueva();

writeFilex(A, p);

ReleaseSemaphore(hSem[0], 1, NULL);

WaitForSingleObject(hSem[1], INFINITE);

writeFilex(B, p);

ReleaseSemaphore(hSem[0], 1, NULL);

WaitForSingleObject(hSem[1], INFINITE);

printf("Producto AB:\n");

readFilex(AB, p);

printMatriz(AB);

ReleaseSemaphore(hSem[0], 1, NULL);

WaitForSingleObject(hSem[4], INFINITE);

printf("Suma CD:\n");

readFilex(C\_D, p);

printMatriz(C\_D);

ReleaseSemaphore(hSem[0], 1, NULL);

}else if(i == 1){

//Proceso padre

float \*p = mem2("memoria");

matriz A = nueva(), B = nueva();

matriz C = matrizAleatoria(), D = matrizAleatoria();

C = matrizAleatoria(), D = matrizAleatoria();

WaitForSingleObject(hSem[0], INFINITE);

printf("Matriz A:\n");

readFilex(A, p);

printMatriz(A);

ReleaseSemaphore(hSem[1], 1, NULL);

WaitForSingleObject(hSem[0], INFINITE);

printf("Matriz B:\n");

readFilex(B, p);

printMatriz(B);

matriz AB = multiplicacion(A, B);

writeFilex(AB, p);

ReleaseSemaphore(hSem[1], 1, NULL);

WaitForSingleObject(hSem[0], INFINITE);

writeFilex(C, p);

ReleaseSemaphore(hSem[2], 1, NULL);

WaitForSingleObject(hSem[3], INFINITE);

writeFilex(D, p);

ReleaseSemaphore(hSem[2], 1, NULL);

WaitForSingleObject(hSem[3], INFINITE);

}else if(i == 2){

//Proceso hijo

float \*p = mem2("memoria");

matriz C = nueva();

matriz D = nueva();

WaitForSingleObject(hSem[2], INFINITE);

printf("Matriz C:\n");

readFilex(C, p);

printMatriz(C);

ReleaseSemaphore(hSem[3], 1, NULL);

WaitForSingleObject(hSem[2], INFINITE);

printf("Matriz D:\n");

readFilex(D, p);

printMatriz(D);

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

writeFilex(C\_D, p);

ReleaseSemaphore(hSem[4], 1, NULL);

WaitForSingleObject(hSem[0], INFINITE);

ReleaseSemaphore(hSem[3], 1, NULL);

}

}

void semaforo(int proc){

int i;

for( i = 0; i < 5; ++i){

char name[100];

sprintf(name, "sem%d", i);

if(proc == 0){

hSem[i] = CreateSemaphore(NULL, 0, 1, name);

}

else

hSem[i] = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, name);

}

}

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

srand(time(NULL));

int proc = 0;

if(argc > 1) sscanf(argv[1], "%d", &proc);

semaforo(proc);

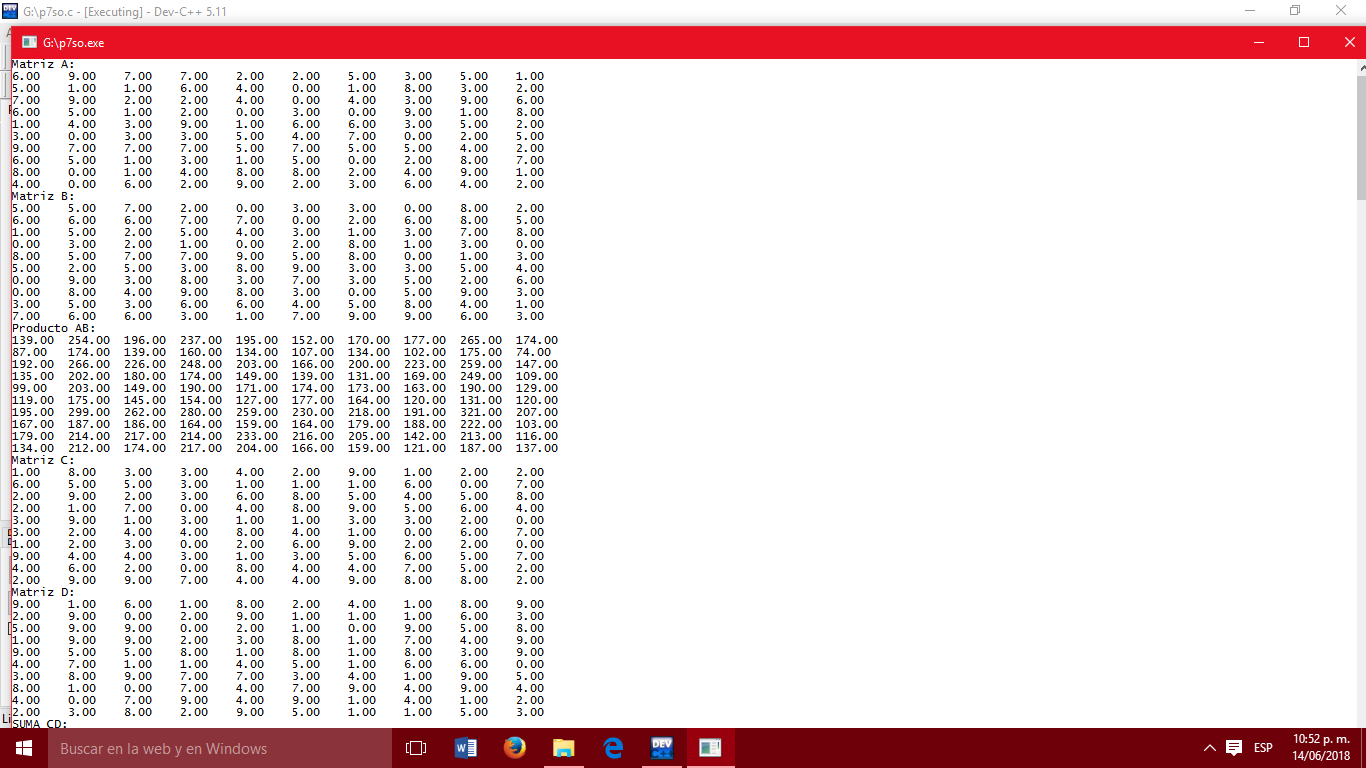
if(proc < 2) proceso(argv[0], proc + 1);

operaciones(proc);

return 0;

}

Captura



**Conclusiones**

**Reyes Valenzuela:** Pudimos comprobar la funcionalidad de los semáforos tanto en Windows como en Linux, en donde vimos que su implementación es menos engorrosa, a pesar de tener la misma función.

**Guerra Vargas Irving Cristóbal:** En esta práctica, como en las anteriores, compartimos informaciones entre procesos de una manera más fácil, y mas rápida que otros métodos para compartir informacion.

**Dávila Méndez Juan Manuel:** De manera similar a las prácticas hechas anteriormente, fue posible observar esta ocasión cómo compartir recursos pero utilizando mecanismos de sincronización. En el caso de esta práctica, los semáforos sirvieron para no tener que usar más matrices así como para no permitir que, por ejemplo, una función, altere los datos que otra función pueda estar utilizando. Sin embargo, quizá el uso de los mutex sea más fácil, pues los administra el sistema operativo y, en cambio, con semáforos, uno tiene que asegurarse de que cada semáforo esté controlado en orden, lo cual es algo tedioso.

Código sección Linux:

#include <stdio.h>

#include "operadores.h"

#include <stdlib.h>

#define SIZ\_MAX 3

int main(int argc, char \*\*argv)

{

sem\_t \*ptrsemaforo; //Apuntador de tipo semaforo

if(sem\_unlink("semaforochevere")!=0) //Tratar de desenlazar un semaforo ya existente

printf("[info: no habia semaforos anteriormente]\n");

if ((ptrsemaforo = sem\_open("semaforochevere", O\_CREAT, 0644, 1)) == SEM\_FAILED) {

printf("Error al inicializar semaforo\n"); //Tratar de crear un nuevo semaforo con nombre

exit(1);

}

printf("[info: dir\_semaforo: %p]\n",ptrsemaforo);

float a[SIZ\_MAX][SIZ\_MAX],b[SIZ\_MAX][SIZ\_MAX],res[SIZ\_MAX][SIZ\_MAX];;

printf("Ingrese datos en A:\n");

InsEnMat(a,SIZ\_MAX,SIZ\_MAX);

CompartirMat(a,1111);

printf("Ingrese datos en B:\n");

InsEnMat(b,SIZ\_MAX,SIZ\_MAX);

CompartirMat(b,1112);

if(fork()==0) //Primer hijo

{

if((sem\_wait(ptrsemaforo))==0) //Chequeo del primers semaforo

{

printf("[info-hijo]: hijo ha entrado a la region critica\n");

ObtenerMatCompartida(1111,a);

ObtenerMatCompartida(1112,b);

Multiplicar(a,b,res);

CompartirMat(res,1113); //Aqui, esta matriz la recibe el padre

printf("CALCULO DE DATOS HIJO 1\n");

}

sleep(0);

if(sem\_post(ptrsemaforo)==0) //Dejar que el abuelo continue(1), a partir de este punto

printf("[info-hijo] hijo salio de la seccion critica\n");//es posible reutilizar las matrices

sem\_wait(ptrsemaforo); //poner en "rojo" el semaforo para poder meter datos despues de regresar el resultado

printf("Ingrese datos en A:\n");

InsEnMat(a,SIZ\_MAX,SIZ\_MAX);

CompartirMat(a,1111);

printf("Ingrese datos en B:\n");

InsEnMat(b,SIZ\_MAX,SIZ\_MAX);

CompartirMat(b,1112);

sem\_post(ptrsemaforo);//liberar el semaforo para el nieto

if(fork()==0)

{

if((sem\_wait(ptrsemaforo))==0) //Esperar autorizacion del abuelo (2)

{

printf("[info-nieto] nieto ha entrado a la seccion critica\n");//es posible reutilizar las matrices

ObtenerMatCompartida(1111,a);

ObtenerMatCompartida(1112,b);

SumaOResta(a,b,res,1);

//ImpMat(res,NULL,1,1,0);

CompartirMat(res,1113);

}

sleep(0);

if(sem\_post(ptrsemaforo)==0) //Dejar que el abuelo continue(3)

printf("[info-nieto] nieto salio de la seccion critica\n");

exit(0);

}

exit(0);

}

sleep(0);

sem\_wait(ptrsemaforo); //Esperar a que el hijo termine de liberar datos (1)

ObtenerMatCompartida(1113,res);

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

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

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

{

TreInversaCaller(res,a,SIZ\_MAX);

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

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

}

else

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

sem\_post(ptrsemaforo); //Permitir que el nieto se ejecute (2)

sem\_wait(ptrsemaforo); //Esperar a que el nieto realice operaciones(3)

ObtenerMatCompartida(1113,res);

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

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

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

{

TreInversaCaller(res,a,SIZ\_MAX);

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

ImpMat(a,"Inversa de la suma .txt",0,0,0);

}

else

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

exit(0);

}

Archivo de biblioteca:

#include<stdio.h>

#include<curses.h>

#include<math.h>

#include<string.h>

#include<semaphore.h>

#include<pthread.h>

#include<stdlib.h>

#include<unistd.h>

#include<fcntl.h>

#include<sys/shm.h>

#include<sys/wait.h>

#include<unistd.h>

#include<sys/syscall.h>

#define SIZ\_MAX 3

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);

}

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];

}

Las capturas de pantalla se encuentran en la página siguiente: ejecución (de manera similar a las prácticas anteriores, no se utilizaron matrices de 10x10 debido a lo tedioso que es meter 200 datos y que además sean linealmente independientes).

Así mismo, un aspecto importante de esta parte de la práctica es que se utilizaron semáforos con nombre, mismos que, haciendo investigación, implementan las mismas funciones que aquellos que no tienen nombre pero con la ventaja de que a estos se les puede dar un nombre para bloquear o, ya sea, desbloquear, similar al uso de los mutex.

