Ejercicio 1

## Makefile

Ejercicio1:Ejercicio1.o

gcc -o Ejercicio1 Ejercicio1.o -lpthread

Ejercicio1.o:Ejercicio1.c

gcc -c Ejercicio1.c -lpthread

**Ejercicio1.c**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/time.h>

#include <unistd.h>

#include <sys/wait.h>

#include <signal.h>

#include <semaphore.h>

#include <errno.h>

#include <fcntl.h>

#include <time.h>

#include <string.h>

#include <sys/shm.h>

#include <sys/ipc.h>

#include <sys/stat.h>

## #define H 3

## void crearSemaforos( void );

## void cerrarSemaforos( void );

## void eliminarSemaforos( void );

## void leer\_Archivo( FILE \*pf);

## int M, W, D, E, S, A, i, j, k;

## pid\_t \*vecM, \*vecH, \*vecW, pid;

## sem\_t \*mutexD, \*semDP, \*semDC, \*mutexE, \*semEP, \*semEC, \*mutexS, \*semSP, \*semSC, \*mutexA, \*semAP, \*semAC;

## struct s\_memoria{

## int minero;

## int herrero;

## int herreroE;

## int herreroS;

## int herreroA;

## };

## ///// Manejador Señales /////

## void sig\_handler(int sigNum)

## {

## int i, j, k;

## for ( i=0; i<=M; i++)

## kill (vecM[i], SIGTERM);

## for ( j=0; j<=H; j++)

## kill (vecH[j], SIGTERM);

## for ( k=0; k<=W; k++)

## kill (vecW[k], SIGTERM);

## 

## eliminarSemaforos();

## free (vecM);

## free (vecH);

## free (vecW);

## exit(0);

## }

## /////////////////////// MAIN ////////////////////////

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

## {

## int i, j, k, espera, Id\_Memoria, val1, val2, time;

## pid\_t padre;

## key\_t clave;

## struct s\_memoria \*memoria;

## FILE \* pf;

## /////-Lectura Archivo Configuracion-/////

## if( ( pf = fopen( argv[1], "r" ) ) == NULL )

## {

## printf( "Error al abrir el archivo de configuracion\n" );

## exit( 1 );

## }

## leer\_Archivo( pf ); //Lee y valida el archivo de configuracion

## printf( "W: %d\n", W );

## printf( "D: %d\n", D );

## printf( "M: %d\n", M );

## printf( "H: %d\n", H );

## printf( "E: %d\n", E );

## printf( "S: %d\n", S );

## printf( "A: %d\n", A );

## /////Creacion e inicializacion de semaforos /////

## crearSemaforos();

## ///// Capturo señales /////

## signal(SIGTERM, sig\_handler);

## signal(SIGUSR1, sig\_handler);

## signal(SIGUSR2, sig\_handler);

## ///// Vector de pids /////

## vecM = (pid\_t \*) malloc(sizeof(pid\_t) \* M);

## if( vecM == NULL)

## {

## printf("\nMemoria insuficiente.\n");

## cerrarSemaforos();

## eliminarSemaforos();

## exit(1);

## }

## vecH = (pid\_t \*) malloc(sizeof(pid\_t) \* H);

## if( vecH == NULL)

## {

## printf("\nMemoria insuficiente.\n");

## cerrarSemaforos();

## eliminarSemaforos();

## exit(1);

## }

## vecW = (pid\_t \*) malloc(sizeof(pid\_t) \* W);

## if( vecW == NULL)

## {

## printf("\nMemoria insuficiente.\n");

## cerrarSemaforos();

## eliminarSemaforos();

## exit(1);

## }

## ///// Definicion Memoria Compartida /////

## clave = ftok( ".", 'M' );

## if( clave == -1 )

## {

## printf( "Error al crear la clave de la Memoria Compartida\n" );

## exit(0);

## }

## Id\_Memoria = shmget( clave, sizeof(struct s\_memoria ), IPC\_CREAT | 0660 );

## if( Id\_Memoria == -1 )

## {

## printf( "Error al crear la Memoria Compartida\n" );

## exit(0);

## }

## memoria = (struct s\_memoria\*) shmat( Id\_Memoria, 0, 0 );

## if( memoria == NULL )

## {

## printf( "Error, no hay Memoria\n" );

## exit(0);

## }

## ///// ------ /////

## memoria->minero = W \* 4;

## memoria->herrero = W \* 4;

## memoria->herreroE = memoria->herreroS = memoria->herreroA = W;

## //memoria->soldado = 3;

## //memoria->soldadoE = memoria->soldadoS = memoria->soldadoA = W;

## padre = getpid(); //Guardo el pid del proceso padre.

## printf( "Iniciamos, Soy Padre <%d>\n\n", padre );

## /////-Procesos Mineros-/////

## for (i=0 ; i<M; i++)

## {

## pid = fork();

## if( pid == 0 ) //Ejecuta el hijo.

## {

## vecM[i] = getpid();

## while( memoria->minero > 0 )

## {

## sem\_wait( semDP ); //P(semDP)

## sem\_wait( mutexD ); //P(mutexD)

## 

## if( memoria->minero <= 0 )

## {

## sem\_post( mutexD );

## break;

## }

## memoria->minero--; //Uso de RC(Mem Comp.)

## printf("Minero <%d> genero 1 de hierro\n",vecM[i]);

## 

## sem\_post( mutexD ); //V(mutexD)

## sem\_post( semDC ); //V(semDC)

## 

## usleep( 500000 ); //Duerme 0.5 seg

## }

## sem\_close( mutexD );

## sem\_close( semDP );

## sem\_close( semDC );

## exit(0);

## }

## else

## {

## if( pid < 0)

## {

## printf("Error al crear el proceso Minero \n");

## exit(1);

## }

## }

## }

## /////-Proceso Herrero-/////

## for (j=0 ; j<H; j++)

## {

## pid = fork();

## if( pid == 0 ) //Ejecuta el hijo.

## {

## srand( getpid() );

## time = rand () % (3-0) + 1; //Para dormir entre 1 y 3 seg.

## vecH[j] = getpid();

## while( memoria->herrero > 0 )

## {

## sem\_wait( semDC );

## sem\_wait( mutexD );

## 

## if( memoria->herrero <= 0 )

## {

## sem\_post( mutexD );

## break;

## }

## memoria->herrero--; //Uso de RC(Mem Comp.)

## sem\_post( mutexD );

## sem\_post( semDP );

## 

## if( getpid() == vecH[0] )

## {

## while( memoria->herreroE > 0 )

## {

## sem\_wait( semEP );

## sem\_wait( mutexE );

## if( memoria->herreroE <= 0 )

## {

## sem\_post( mutexE );

## break;

## }

## printf("Herrero <%d> fabrico un escudo en %d segundos \n",vecH[j], time );

## memoria->herreroE--;

## sem\_post( mutexE );

## sem\_post( semEC );

## }

## }

## if( getpid() == vecH[1] )

## {

## while( memoria->herreroS > 0 )

## {

## sem\_wait( semSP );

## sem\_wait( mutexS );

## if( memoria->herreroS <= 0 )

## {

## sem\_post( mutexS );

## break;

## }

## printf("Herrero <%d> fabrico una espada en %d segundos \n",vecH[j], time );

## memoria->herreroS--;

## sem\_post( mutexS );

## sem\_post( semSC );

## }

## }

## if( getpid() == vecH[2] )

## {

## while( memoria->herreroA > 0 )

## {

## sem\_wait( semDC );

## sem\_wait( semAP );

## sem\_wait( mutexA );

## if( memoria->herreroA <= 0 )

## {

## sem\_post( mutexA );

## break;

## }

## printf("Herrero <%d> fabrico una armadura en %d segundos \n",vecH[j], time );

## memoria->herreroA--;

## sem\_post( mutexA );

## sem\_post( semAC );

## }

## }

## sleep( time ); //Duerme 1-3 seg

## 

## }

## 

## sem\_close( mutexD );

## sem\_close( semDP );

## sem\_close( semDC );

## if( getpid() == vecH[0] )

## {

## sem\_close( semEP );

## sem\_close( semEC );

## sem\_close( mutexE );

## exit(EXIT\_SUCCESS);

## }

## if( getpid() == vecH[1] )

## {

## sem\_close( semSP );

## sem\_close( semSC );

## sem\_close( mutexS );

## exit(EXIT\_SUCCESS);

## }

## if( getpid() == vecH[2] )

## {

## sem\_close( semAP );

## sem\_close( semAC );

## sem\_close( mutexA );

## exit(EXIT\_SUCCESS);

## }

## }

## else

## {

## if( pid < 0)

## {

## printf("Error al crear el proceso Herrero \n");

## exit(1);

## }

## }

## }

## /////-Proceso Soldado-/////

## for (k=0 ; k<W; k++)

## {

## pid = fork();

## if( pid == 0 ) //Ejecuta el hijo.

## {

## vecW[k] = getpid();

## 

## sem\_wait( semEC );

## sem\_wait( mutexE );

## printf( "Soldado <%d> retiro un escudo\n",vecW[k] );

## sem\_post( mutexE );

## sem\_post( semEP );

## 

## sem\_wait( semSC );

## sem\_wait( mutexS );

## printf( "Soldado <%d> retiro una espada\n", vecW[k] );

## sem\_post( mutexS );

## sem\_post( semSP );

## 

## sem\_wait( semAC );

## sem\_wait( mutexA );

## printf( "Soldado <%d> retiro una armadura\n", vecW[k] );

## sem\_post( mutexA );

## sem\_post( semAP );

## 

## sem\_close( semEP );

## sem\_close( semEC );

## sem\_close( mutexE );

## sem\_close( semSP );

## sem\_close( semSC );

## sem\_close( mutexS );

## sem\_close( semAP );

## sem\_close( semAC );

## sem\_close( mutexA );

## exit(EXIT\_SUCCESS);

## }

## else

## {

## if( pid < 0)

## {

## printf("Error al crear el proceso Soldado \n");

## exit(1);

## }

## }

## }

## ///////////////////////////////////////////////////////////////////////////

## for (espera=0 ; espera<(M+H+W); espera++)

## {

## wait(NULL);

## }

## shmdt( memoria );

## shmctl( Id\_Memoria, IPC\_RMID, 0 );

## eliminarSemaforos();

## //printf("fin de programa\n");

## exit(0);

## }

## //////////////////////////////////////////////////////////////////////////

## /////////////////////////////- FUNCIONES -////////////////////////////////

## void leer\_Archivo( FILE \*pf )

## {

## char linea[10];

## while( fgets( linea, sizeof( linea ), pf ) )

## {

## if( linea[1] != ':' )

## {

## printf( "Error en el archivo de configuracion\n" );

## exit( 0 );

## }

## switch( linea[0] )

## {

## case 'W':

## {

## W = atoi( (linea+2) );

## break;

## }

## case 'D':

## {

## D = atoi( (linea+2) );

## break;

## }

## case 'M':

## {

## M = atoi( (linea+2) );

## break;

## }

## case 'A':

## {

## A = atoi( (linea+2) );

## break;

## }

## case 'E':

## {

## E = atoi( (linea+2) );

## break;

## }

## case 'S':

## {

## S = atoi( (linea+2) );

## break;

## }

## default:

## {

## printf( "Error en el archivo de configuracion\n" );

## exit(0);

## }

## }

## }

## }

## void crearSemaforos()

## {

## mutexD = sem\_open ("/mutexD", O\_CREAT, 0644, 1); //Comprueba si existe el semáforo,y sino lo crea.

## if( mutexD == (sem\_t \*) -1 )

## {

## perror( "Error al crear el semaforo mutexD\n"); //Validación

## exit(1);

## }

## semDP = sem\_open ("/semDP", O\_CREAT, 0644, D);

## if( semDP == (sem\_t \*) - 1 )

## {

## perror("Error al crear el semaforo semDP\n");

## sem\_close(mutexD);

## sem\_unlink("/mutexD");

## exit(1);

## }

## semDC = sem\_open ("/semDC", O\_CREAT, 0644, 0);

## if( semDC == (sem\_t \*)-1 )

## { perror("Error al crear el semaforo semDC\n");

## sem\_close(mutexD);

## sem\_unlink("/mutexD");

## sem\_close(semDP);

## sem\_unlink("/semDP");

## exit(1);

## }

## mutexE = sem\_open ("/mutexE", O\_CREAT, 0644, 1);

## if( mutexE == (sem\_t \*)-1 )

## { perror("Error al crear el semaforo mutexE\n");

## sem\_close(mutexD);

## sem\_unlink("/mutexD");

## sem\_close(semDP);

## sem\_unlink("/semDP");

## sem\_close(semDC);

## sem\_unlink("/semDC");

## exit(1);

## }

## semEP = sem\_open ("/semEP", O\_CREAT, 0644, E);

## if( semEP == (sem\_t \*)-1 )

## { perror("Error al crear el semaforo semEP\n");

## sem\_close(mutexD);

## sem\_unlink("/mutexD");

## sem\_close(semDP);

## sem\_unlink("/semDP");

## sem\_close(semDC);

## sem\_unlink("/semDC");

## sem\_close(mutexE);

## sem\_unlink("/mutexE");

## exit(1);

## }

## semEC = sem\_open ("/semEC", O\_CREAT, 0644, 0);

## if( semEC == (sem\_t \*)-1 )

## { perror("Error al crear el semaforo semEC\n");

## sem\_close(mutexD);

## sem\_unlink("/mutexD");

## sem\_close(semDP);

## sem\_unlink("/semDP");

## sem\_close(semDC);

## sem\_unlink("/semDC");

## sem\_close(mutexE);

## sem\_unlink("/mutexE");

## sem\_close(semEP);

## sem\_unlink("/semEP");

## exit(1);

## }

## mutexS = sem\_open ("/mutexS", O\_CREAT, 0644, 1);

## if( mutexS == (sem\_t \*)-1 )

## { perror("Error al crear el semaforo mutexS\n");

## sem\_close(mutexD);

## sem\_unlink("/mutexD");

## sem\_close(semDP);

## sem\_unlink("/semDP");

## sem\_close(semDC);

## sem\_unlink("/semDC");

## sem\_close(mutexE);

## sem\_unlink("/mutexE");

## sem\_close(semEP);

## sem\_unlink("/semEP");

## sem\_close(semEC);

## sem\_unlink("/semEC");

## exit(1);

## }

## semSP = sem\_open ("/semSP", O\_CREAT, 0644, S);

## if( semSP == (sem\_t \*)-1 )

## { perror("Error al crear el semaforo semSP\n");

## sem\_close(mutexD);

## sem\_unlink("/mutexD");

## sem\_close(semDP);

## sem\_unlink("/semDP");

## sem\_close(semDC);

## sem\_unlink("/semDC");

## sem\_close(mutexE);

## sem\_unlink("/mutexE");

## sem\_close(semEP);

## sem\_unlink("/semEP");

## sem\_close(semEC);

## sem\_unlink("/semEC");

## sem\_close(mutexS);

## sem\_unlink("/mutexS");

## exit(1);

## }

## semSC = sem\_open ("/semSC", O\_CREAT, 0644, 0);

## if( semSC == (sem\_t \*)-1 )

## { perror("Error al crear el semaforo semSC\n");

## sem\_close(mutexD);

## sem\_unlink("/mutexD");

## sem\_close(semDP);

## sem\_unlink("/semDP");

## sem\_close(semDC);

## sem\_unlink("/semDC");

## sem\_close(mutexE);

## sem\_unlink("/mutexE");

## sem\_close(semEP);

## sem\_unlink("/semEP");

## sem\_close(semEC);

## sem\_unlink("/semEC");

## sem\_close(mutexS);

## sem\_unlink("/mutexS");

## sem\_close(semSP);

## sem\_unlink("/semSP");

## exit(1);

## }

## mutexA = sem\_open ("/mutexA", O\_CREAT, 0644, 1);

## if( mutexA == (sem\_t \*)-1 )

## { perror("Error al crear el semaforo mutexA\n");

## sem\_close(mutexD);

## sem\_unlink("/mutexD");

## sem\_close(semDP);

## sem\_unlink("/semDP");

## sem\_close(semDC);

## sem\_unlink("/semDC");

## sem\_close(mutexE);

## sem\_unlink("/mutexE");

## sem\_close(semEP);

## sem\_unlink("/semEP");

## sem\_close(semEC);

## sem\_unlink("/semEC");

## sem\_close(mutexS);

## sem\_unlink("/mutexS");

## sem\_close(semSP);

## sem\_unlink("/semSP");

## sem\_close(semSC);

## sem\_unlink("/semSC");

## exit(1);

## }

## semAP = sem\_open ("/semAP", O\_CREAT, 0644, A);

## if( semAP == (sem\_t \*)-1 )

## { perror("Error al crear el semaforo semAP\n");

## sem\_close(mutexD);

## sem\_unlink("/mutexD");

## sem\_close(semDP);

## sem\_unlink("/semDP");

## sem\_close(semDC);

## sem\_unlink("/semDC");

## sem\_close(mutexE);

## sem\_unlink("/mutexE");

## sem\_close(semEP);

## sem\_unlink("/semEP");

## sem\_close(semEC);

## sem\_unlink("/semEC");

## sem\_close(mutexS);

## sem\_unlink("/mutexS");

## sem\_close(semSP);

## sem\_unlink("/semSP");

## sem\_close(semSC);

## sem\_unlink("/semSC");

## sem\_close(mutexA);

## sem\_unlink("/mutexA");

## exit(1);

## }

## semAC = sem\_open ("/semAC", O\_CREAT, 0644, 0);

## if( semAC == (sem\_t \*)-1 )

## { perror("Error al crear el semaforo semAC\n");

## sem\_close(mutexD);

## sem\_unlink("/mutexD");

## sem\_close(semDP);

## sem\_unlink("/semDP");

## sem\_close(semDC);

## sem\_unlink("/semDC");

## sem\_close(mutexE);

## sem\_unlink("/mutexE");

## sem\_close(semEP);

## sem\_unlink("/semEP");

## sem\_close(semEC);

## sem\_unlink("/semEC");

## sem\_close(mutexS);

## sem\_unlink("/mutexS");

## sem\_close(semSP);

## sem\_unlink("/semSP");

## sem\_close(semSC);

## sem\_unlink("/semSC");

## sem\_close(mutexA);

## sem\_unlink("/mutexA");

## sem\_close(semAP);

## sem\_unlink("/semAP");

## exit(1);

## }

## }

## void cerrarSemaforos()

## {

## sem\_close( mutexD );

## sem\_close( semDP );

## sem\_close( semDC );

## sem\_close( mutexE );

## sem\_close( semEP );

## sem\_close( semEC );

## sem\_close( mutexS );

## sem\_close( semSP );

## sem\_close( semSC );

## sem\_close( mutexA );

## sem\_close( semAP );

## sem\_close( semAC );

## }

## void eliminarSemaforos()

## {

## sem\_unlink("/mutexD");

## sem\_unlink("/semDP");

## sem\_unlink("/semDC");

## sem\_unlink("/mutexE");

## sem\_unlink("/semEP");

## sem\_unlink("/semEC");

## sem\_unlink("/mutexS");

## sem\_unlink("/semSP");

## sem\_unlink("/semSC");

## sem\_unlink("/mutexA");

## sem\_unlink("/semAP");

## sem\_unlink("/semAC");

## }

## Manual 1

###################### MANUAL DE USO #########################

0)Posicionarse en el directorio /Ejercicio1

1)Ejecutar make

2)Ejecutar ./Ejercicio1 <Archivo\_Configuracion.txt>

NOTA: "Archivo\_Configuracion.txt" es el nombre del artchivo que contiene la configuracion del programa.

El Archivo de configuracion debera contener la siguiente informacion:

M: Cantidad de Mineros

D: Capacidad del deposito de hierro

S: Capacidad del deposito de espadas

E: Capacidad del deposito de escudos

A: Capacidad del deposito de armaduras

W: Numero de Soldados

Ejemplo del archivo de configuracion:

M:3

D:4

S:3

E:2

A:2

W:5

Ejercicio 2

## makefile

## Ejercicio2:Ejercicio2.o

## gcc -o Ejercicio2 Ejercicio2.o -lpthread

## Ejercicio2.o:Ejercicio2.c

## gcc -c Ejercicio2.c –lpthread

**TP3\_Ej2.c**

## /\*###############################################

## # Trabajo Práctico N°3 #

## # Ejercicio 2 #

## # #

## # #

## # Integrantes: #

## # #

## # Maximiliano Toledo 31982343 #

## # Marcelo Romero 34140911 #

## # Juan Jose Martinez 30944263 #

## # Oscar Emilio Pereyra 32382585 #

## # Javier Collazo 31632425 #

## # #

## # Entrega N°1 #

## ###############################################\*/

## #include <stdio.h>

## #include <stdlib.h>

## #include <sys/ipc.h>

## #include <sys/types.h>

## #include <sys/shm.h>

## #include <sys/sem.h>

## #include <signal.h>

## #include <string.h>

## #define SUMA 1

## #define RESTA 1

## #define DIVI 1

## #define MULTI 1

## typedef struct operaciones

## {

## int operando1;

## int operando2;

## char operacion[3];

## int hay\_op;//variable auxiliar podria no ir

## int hay\_resultado;

## float resultado;//variable auxiliar podria no ir

## int hijo\_suma;//variable auxiliar podria no ir

## }t\_operaciones;

## typedef struct operaciones2

## {

## int operando1;

## int operando2;

## char operacion[3];

## }t\_ope;

## int SALIDA=1;

## t\_operaciones \*suma;

## t\_operaciones \*divi;

## t\_operaciones \*multi;

## t\_operaciones \*resta;

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Semaforos\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## union semun {

## int val;

## struct semid\_ds \*buf;

## unsigned short \*array;

## struct seminfo \*\_\_buf;

## };

## int obtenerMutex(key\_t clave);

## int obtenerSemaforo(key\_t clave, int valor);

## void pedirSemaforo(int IdSemaforo);

## void devolverSemaforo(int IdSemaforo);

## void eliminarSemaforo(int IdSemaforo);

## void eliminarMutex(int IdSemaforo);

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

## {

## int p=0;

## key\_t mutex\_suma,mutex\_prod,mutex\_cons;

## key\_t mutex\_multi,multi\_prod,multi\_cons;

## key\_t mutex\_divi,divi\_prod,divi\_cons;

## key\_t mutex\_resta,resta\_prod,resta\_cons;

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## int id\_suma,prod\_suma,cons\_suma,hay\_ope;//varibles para los semaforos de la suma

## int id\_multi,prod\_multi,cons\_multi;//varibles para los semaforos de la multiplicacion

## int id\_resta,prod\_resta,cons\_resta;//varibles para los semaforos de resta

## int id\_divi,prod\_divi,cons\_divi;//varibles para los semaforos de division

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## t\_operaciones vec\_ope[2];//vector donde guardo los datos en la mem compartida

## t\_ope vec[100];//vector de estrucutura donde se guardaran todas las operaciones que hay en el archivo

## char operaciones[3][3];//con esta variable guardo las divisiones de las operaciones al vector de estrucutra

## char cadena[20];//variable donde guardo la linea(operacion) que leo desde el archivo

## int i,j=-1,estado,x=0;

## FILE \*pf,\*graba\_rdos;//puntero a archivo

## pid\_t pid; //guardo el pid

## char \* pch;//para dividir las operaciones y poder guardarlas en el vector estructura

## //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

## ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

## /\*inicializamos los contadores\*/

## int contador\_gral=0,cont\_suma=0,cs=0,cont\_resta=0,cr=0,cont\_divi=0,cd=0,cont\_multi=0,cm=0,cant\_op=0;

## char codigo;

## key\_t clave\_suma,clave\_resta,clave\_multi,clave\_divi;

## int Id\_Memoria\_suma,Id\_Memoria\_resta,Id\_Memoria\_multi,Id\_Memoria\_divi;

## //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

## /////////////////////////////////////\*Abro el archivo y guardo las operaciones en un vector\*//////////////////////////////////

## //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

## if(strcmp(argv[1],"-?" )==0|| strcmp(argv[1],"-h")==0 ||strcmp(argv[1],"help" )==0)

## {

## printf("Debe ingresar por parametro un archivo con operaciones separadas por un espacio. Ej. 4 + 7 etc. el programa calculara las operaciones y guardara los resultado en un archivo de salida llamado resultados.txt \n");

## exit(1);

## }

## if((pf=fopen(argv[1], "r"))==NULL)

## {

## printf("Error al abrir archivo de configuracion del servidor.");

## exit(1);

## }

## if((graba\_rdos=fopen("resultados.txt", "w"))==NULL)

## {

## printf("Error al abrir archivo de resultados");

## exit(1);

## }

## ///\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*///

## //////////////\*Semaforos para la suma\*////////////////

## mutex\_suma= ftok("/", 6000);

## mutex\_prod= ftok("/", 6001);

## mutex\_cons= ftok("/", 6002);

## id\_suma = obtenerMutex(mutex\_suma);

## prod\_suma=obtenerSemaforo(mutex\_prod,1);

## cons\_suma=obtenerSemaforo(mutex\_cons,0);

## //////////////////////////////////////////////////////////

## //\*\*\*Semaforos para la multiplicacion\*\*\*\*/////////////////

## mutex\_multi= ftok("/", 6003);

## multi\_prod= ftok("/", 6004);

## multi\_cons= ftok("/", 6005);

## id\_multi = obtenerMutex(mutex\_multi);

## prod\_multi=obtenerSemaforo(multi\_prod,1);

## cons\_multi=obtenerSemaforo(multi\_cons,0);

## //////////////////////////////////////////////////////////

## //\*\*\*Semaforos para la resta\*\*\*\*/////////////////

## mutex\_resta= ftok("/", 6006);

## resta\_prod= ftok("/", 6007);

## resta\_cons= ftok("/", 6008);

## id\_resta = obtenerMutex(mutex\_resta);

## prod\_resta=obtenerSemaforo(resta\_prod,1);

## cons\_resta=obtenerSemaforo(resta\_cons,0);

## //////////////////////////////////////////////////////////

## //\*\*\*Semaforos para la division\*\*\*\*///////////////////////

## mutex\_divi= ftok("/", 6009);

## divi\_prod= ftok("/", 6010);

## divi\_cons= ftok("/", 6011);

## id\_divi = obtenerMutex(mutex\_divi);

## prod\_divi=obtenerSemaforo(divi\_prod,1);

## cons\_divi=obtenerSemaforo(divi\_cons,0);

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## //\*\*Creo la memoria compartida para las 4 operaciones\*\*///////////////////////////////////////////////////////////////////////

## /////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

## /\*Memoria para la suma\*/

## clave\_suma = ftok ("/bin/ls", 33);

## if (clave\_suma == -1)

## {

## printf ("No consigo clave para memoria compartida \n");

## exit(0);

## }

## Id\_Memoria\_suma = shmget (clave\_suma, sizeof(suma[2]), 0777 | IPC\_CREAT);

## if (Id\_Memoria\_suma == -1)

## {

## printf ("No consigo clave para memoria compartida \n");

## exit (0);

## }

## suma = (t\_operaciones\*)shmat(Id\_Memoria\_suma, NULL, 0);//Buffer de memoria para las operaciones de suma

## if (suma == NULL)

## {

## printf ("No consigo clave para memoria compartida \n");

## exit (0);

## }

## /\*Memoria para la multiplicacion\*/

## clave\_multi = ftok ("/bin/ls", 35);

## if (clave\_multi == -1)

## {

## printf ("No consigo clave para memoria compartida \n");

## exit(0);

## }

## Id\_Memoria\_multi = shmget (clave\_multi, sizeof multi[2], 0777 | IPC\_CREAT);

## if (Id\_Memoria\_multi == -1)

## {

## printf ("No consigo clave para memoria compartida \n");

## exit (0);

## }

## multi = (t\_operaciones\*)shmat(Id\_Memoria\_multi,NULL, 0);//Buffer de memoria para las operaciones de multiplicacion

## if (multi == NULL)

## {

## printf ("No consigo clave para memoria compartida \n");

## exit (0);

## }

## //\*\*\*\*\*\*\*\*\*\*\*\*\*\*Memoria para la resta\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## clave\_resta = ftok ("/bin/ls", 34);

## if (clave\_resta == -1)

## {

## printf ("No consigo clave para memoria compartida \n");

## exit(0);

## }

## Id\_Memoria\_resta = shmget (clave\_resta, sizeof resta[2], 0777 | IPC\_CREAT);

## if (Id\_Memoria\_resta == -1)

## {

## printf ("No consigo clave para memoria compartida \n");

## exit (0);

## }

## resta = (t\_operaciones\*)shmat (Id\_Memoria\_resta,NULL, 0);

## if (resta == NULL)

## {

## printf ("No consigo clave para memoria compartida \n");

## exit (0);

## }

## /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Memoria para la division\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## 

## clave\_divi = ftok ("/bin/ls", 36);

## if (clave\_divi == -1)

## {

## printf ("No consigo clave para memoria compartida \n");

## exit(0);

## }

## Id\_Memoria\_divi = shmget (clave\_divi, sizeof(int)\*100, 0777 | IPC\_CREAT);

## if (Id\_Memoria\_divi == -1)

## {

## printf ("No consigo clave para memoria compartida \n");

## exit (0);

## }

## divi = (t\_operaciones\*)shmat (Id\_Memoria\_divi,NULL, 0);

## if (divi == NULL)

## {

## printf ("No consigo clave para memoria compartida \n");

## exit (0);

## }

## ///////////////////////////////////////////////////////////////////////////////////////////

## //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

## //\*\*Lectura del archivo y realizacion de las validaciones\*\*//////////////////////////////////////////////////////

## while (fgets(cadena,12, pf) != NULL) //leo el archivo y guardamos los datos en un vector de estructura "vec"

## {

## p=0;

## pch = strtok (cadena," ");

## while (pch != NULL)

## {

## strcpy(operaciones[p],pch);

## p++;

## pch = strtok (NULL, " ,.");

## }

## vec[cant\_op].operando1=atoi(operaciones[0]);

## if(vec[cant\_op].operando1 < 0 || vec[cant\_op].operando1 > 100)//Valida que el operando dos este entre 0 y 99

## {

## printf("Error de numero , debe ingresar un numero entre 0 y 99 \n");

## exit(1);

## }

## strcpy(vec[cant\_op].operacion,operaciones[1]);

## if(strcmp(operaciones[1],"+")!=0&&strcmp(operaciones[1],"-")!=0&&strcmp(operaciones[1],"\*")!=0&&strcmp(operaciones[1],"/")!=0)//Valida que la operacion sea una valida

## {

## printf("Error de Operacion , debe ingresar una operacion valida +, - , \* o / \n");

## exit(1);

## }

## if(strcmp(operaciones[1],"+")==0) cont\_suma++;//contador que cuenta las operaciones de suma que hay en el archivo

## if(strcmp(operaciones[1],"\*")==0) cont\_multi++;//contador que cuenta las operaciones de multiplicacion que hay en el archivo

## if(strcmp(operaciones[1],"/")==0) cont\_divi++;//contador que cuenta las operaciones de division que hay en el archivo

## if(strcmp(operaciones[1],"-")==0) cont\_resta++;//contador que cuenta las operaciones de resta que hay en el archivo

## vec[cant\_op].operando2=atoi(operaciones[2]);

## if(vec[cant\_op].operando2 < 0 || vec[cant\_op].operando2 > 100)//Valida que el operando dos este entre 0 y 99

## {

## printf("Error de numero , debe ingresar un numero entre 0 y 99 \n");

## exit(1);

## }

## cant\_op++;//cuenta la cantidad de operaciones que hay en el archivo

## }

## ///////////////////////////////////////////////////////////////////////////////////////////

## pid = fork();

## 

## if(pid==0)

## { /\*Hijo que se encarga de hacer las operaciones de multiplicacion\*/

## while( cm < cont\_multi )

## {

## pedirSemaforo(cons\_multi);

## pedirSemaforo(id\_multi);

## multi->resultado=multi->operando1 \* multi->operando2;

## fprintf(graba\_rdos,"%d %s %d = %f \n",multi->operando1,multi->operacion,multi->operando2,multi->resultado);

## devolverSemaforo(id\_multi);

## devolverSemaforo(prod\_multi);

## cm++;

## }

## exit(1);

## }

## 

## else {

## pid = fork();

## if(pid==0)

## {/\*Hijo que se encarga de hacer las operaciones de suma\*/

## while( cs < cont\_suma )

## {

## pedirSemaforo(cons\_suma);

## pedirSemaforo(id\_suma);

## suma->resultado=suma->operando1 + suma->operando2;

## fprintf(graba\_rdos,"%d %s %d = %f \n",suma->operando1,suma->operacion,suma->operando2,suma->resultado);

## suma->hay\_resultado=1;

## devolverSemaforo(id\_suma);

## devolverSemaforo(prod\_suma);

## cs++;

## }

## exit(1);

## }

## else

## {

## pid = fork();

## if(pid==0)

## {/\*Hijo que se encarga de hacer las operaciones de division\*/

## while( cd < cont\_divi )

## {

## pedirSemaforo(cons\_divi);

## pedirSemaforo(id\_divi);

## divi->resultado=divi->operando1 / divi->operando2;

## fprintf(graba\_rdos,"%d %s %d = %f \n",divi->operando1,divi->operacion,divi->operando2,divi->resultado);

## divi->hay\_resultado=1;

## devolverSemaforo(id\_divi);

## devolverSemaforo(prod\_divi);

## cd++;

## }

## exit(1);

## }

## 

## else

## {

## pid = fork();

## if(pid==0)

## {

## /\*Hijo que se encarga de hacer las operaciones de resta\*/

## while( cr < cont\_resta )

## {

## 

## pedirSemaforo(cons\_resta);

## pedirSemaforo(id\_resta);

## resta->resultado=resta->operando1 - resta->operando2;

## 

## fprintf(graba\_rdos,"%d %s %d = %f \n",resta->operando1,resta->operacion,resta->operando2,resta->resultado);

## 

## devolverSemaforo(id\_resta);

## devolverSemaforo(prod\_resta);

## cr++;

## }

## exit(1);

## }

## else

## {

## /\*Padre que se encargara las guardar las operaciones en la memoria compartida \*/

## while(contador\_gral < cant\_op )

## {

## 

## vec[contador\_gral].operacion;

## if(strcmp(vec[contador\_gral].operacion,"\*")==0)//Si es operacion multiplicacion pasa

## {

## pedirSemaforo(prod\_multi);

## pedirSemaforo(id\_multi);

## multi->operando1=vec[contador\_gral].operando1;

## multi->operando2=vec[contador\_gral].operando2;

## strcpy(multi->operacion,vec[contador\_gral].operacion);

## devolverSemaforo(id\_multi);

## devolverSemaforo(cons\_multi);

## }

## 

## 

## if(strcmp(vec[contador\_gral].operacion,"+")==0)//Si es operacion suma pasa

## {

## pedirSemaforo(prod\_suma);

## pedirSemaforo(id\_suma);

## suma->operando1=vec[contador\_gral].operando1;

## suma->operando2=vec[contador\_gral].operando2;

## strcpy(suma->operacion,vec[contador\_gral].operacion);

## devolverSemaforo(id\_suma);

## devolverSemaforo(cons\_suma);

## }

## 

## if(strcmp(vec[contador\_gral].operacion,"/")==0)//Si es operacion multiplicacion pasa

## {

## 

## pedirSemaforo(prod\_divi);

## pedirSemaforo(id\_divi);

## divi->operando1=vec[contador\_gral].operando1;

## divi->operando2=vec[contador\_gral].operando2;

## strcpy(divi->operacion,vec[contador\_gral].operacion);

## 

## devolverSemaforo(id\_divi);

## devolverSemaforo(cons\_divi);

## }

## if(strcmp(vec[contador\_gral].operacion,"-")==0)//Si es operacion multiplicacion pasa

## {

## 

## pedirSemaforo(prod\_resta);

## pedirSemaforo(id\_resta);

## resta->operando1=vec[contador\_gral].operando1;

## resta->operando2=vec[contador\_gral].operando2;

## strcpy(resta->operacion,vec[contador\_gral].operacion);

## 

## devolverSemaforo(id\_resta);

## devolverSemaforo(cons\_resta);

## }

## contador\_gral++;

## }

## 

## }

## //\*\*Cosas del padre\*\*/////////// ////////////////////////////////////

## 

## //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/////////////////////

## }

## }

## }

## for (i=0 ; i<4;i++)

## {

## wait(NULL);

## }

## /////////////////////\*Elimina la memoria compartida\*//////////////////////////////

## shmdt ((t\_operaciones\*)suma);

## shmctl (Id\_Memoria\_suma, IPC\_RMID, (struct shmid\_ds \*)NULL);

## shmdt ((t\_operaciones\*)multi);

## shmctl (Id\_Memoria\_multi, IPC\_RMID, (struct shmid\_ds \*)NULL);

## shmdt ((t\_operaciones\*)resta);

## shmctl (Id\_Memoria\_resta, IPC\_RMID, (struct shmid\_ds \*)NULL);

## shmdt ((t\_operaciones\*)divi);

## shmctl (Id\_Memoria\_divi, IPC\_RMID, (struct shmid\_ds \*)NULL);

## /\*\*Elimino los semaforos\*\*/

## eliminarSemaforo(id\_suma);

## eliminarSemaforo(prod\_suma);

## eliminarSemaforo(cons\_suma);

## 

## eliminarSemaforo(id\_multi);

## eliminarSemaforo(prod\_multi);

## eliminarSemaforo(cons\_multi);

## 

## eliminarSemaforo(id\_resta);

## eliminarSemaforo(prod\_resta);

## eliminarSemaforo(cons\_resta);

## eliminarSemaforo(id\_divi);

## eliminarSemaforo(prod\_divi);

## eliminarSemaforo(cons\_divi);

## printf("Los resultado se guardan en el archivo : resultados.txt\n");

## return 1;

## }

## ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

## ///////////////////////////////////////\*\*\*Declaro las funciones\*\*\*//////////////////////////////////////////////////////////////////////////////////////

## ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

## int obtenerMutex(key\_t clave)

## {

## int IdSemaforo;

## union semun CtlSem;

## IdSemaforo = semget(clave, 1, IPC\_CREAT | 0600);

## CtlSem.val = 1;

## semctl(IdSemaforo, 0, SETVAL, CtlSem);

## return IdSemaforo;

## }

## int obtenerSemaforo(key\_t clave, int valor)

## {

## int IdSemaforo;

## union semun CtlSem;

## IdSemaforo = semget(clave, 1, IPC\_CREAT | 0600);

## CtlSem.val = valor;

## semctl(IdSemaforo, 0, SETVAL, CtlSem);

## return IdSemaforo;

## }

## void pedirSemaforo(int IdSemaforo)

## {

## struct sembuf OpSem;

## OpSem.sem\_num = 0;

## OpSem.sem\_op = -1;

## OpSem.sem\_flg = 0;

## semop(IdSemaforo, &OpSem, 1);

## }

## void devolverSemaforo(int IdSemaforo)

## {

## struct sembuf OpSem;

## OpSem.sem\_num = 0;

## OpSem.sem\_op = 1;

## OpSem.sem\_flg = 0;

## semop(IdSemaforo, &OpSem, 1);

## }

## void eliminarSemaforo(int IdSemaforo)

## {

## semctl(IdSemaforo, 0, IPC\_RMID);

## }

## void eliminarMutex(int IdSemaforo)

## {

## semctl(IdSemaforo, 0, IPC\_RMID);

## }

## ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

## 

## ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

## Manual

###################### MANUAL DE USO #########################

0)Posicionarse en el directorio /Ejercicio2

1)Ejecutar make

2)Ejecutar ./launcher <Archivo de operaciones>

NOTA: <Arch> Es el archivo que contendra todas las operaciones a realizar

a)Si el modo de ejecucion y los parametros ingresados son correctos, el

programa generará el archivo con los resultados de las operaciones que fueron pasadas por medio del archivo de operaciones.

b)Si el modo de uso fue incorrecto, el programa emitira un mensaje de ayuda.

Ejercicio 3

## Makefile

######################################################

# TP 3 Ejercicio 3 #

#Integrantes: #

#Maximiliano Toledo 31982343 #

#Marcelo Julio Romero 34140911 #

#Juan Jose Martinez 30944263 #

#Oscar Emilio Pereyra 32382585 #

#Javier Collazo 31632425 #

######################################################

# Makefile

CC= gcc

LD= gcc

CFLAGS= -c -g -I ./include/

LFLAGS= -lpthread -L ./libs/

OBJS= Ej3.o

main: launcher

echo "OK Listo."

launcher: $(OBJS)

$(LD) $(LFLAGS) $(OBJS) -o Ej3

launcher.o: Ej3.c

$(CC) $(CFLAGS) Ej3.c

## Manual

###################### MANUAL DE USO #########################

0)Posicionarse en el directorio /Ejercicio3

1)Ejecutar make

2)Ejecutar ./Ej3 [Path]

NOTA: [Path] Es el directorio en donde se desea analizar los archivos .c, .cpp, .h, .sh, .ini y makefile.

Si se lo deja en blanco se examina el directorio actual.

a) Si el directorio ingresado es valido, el programa mostrara por pantalla el nombre del archivo generado

con el resultado del análisis. Dicho archivo se genera siempre en el directorio en donde esta alojado el

programa.

b) Si el directorio es invalido, el programa emitira un mensaje indicativo.

c) Para mas ayuda puede ejecutar ./Ej3 -h, ./Ej3 -? o ./Ej3 -help

Ejercicio 4

## Makefile

Entrar en la carpeta Cliente y Servidor para visualizarlos

## Manual Cliente

---------------------------------------------------------------------

Manual de Configuracion del Ejercicio 4 del Trabajo Practico número 3

---------------------------------------------------------------------

Compilacion y linkeo:

- Para la etapa de compilacion y linkeo del ejecutable a generar se debe ejecutar "make" desde un terminal estando ubicado en el mismo directorio del ejercicio 3.

Ej: $ make

Ejecucion del ejecutable "cliente":

- Para ejecutar "cliente" se deberá colocar "./cliente <ip\_del\_servidor>" en un terminal ubicado en la misma direccion que el ejecutable.

Parametros:

- El programa se le coloca como parametro la ip del servidor (Ej: 192.168.1.153) en formato ipv4.

Comportamiento:

- El cliente se conecta al ip indicado como parametro.

- Mantiene la conexion

- La conexion con el servidor la mantiene hasta que alguno de los dos la cierre

- Se empieza a comunicar con los movimientos que desea mandar, para luego dar el resultado de las filas y las columnas

- Se visualiza la matriz parcial de resultados

Envio de señales:

- Para terminar con el proceso correctamente se le debe enviar la señal SIGINT (ctrl+c)

## Manual Servidor

---------------------------------------------------------------------

Manual de Configuracion del Ejercicio 4 del Trabajo Practico número 3

---------------------------------------------------------------------

Compilacion y linkeo:

- Para la etapa de compilacion y linkeo del ejecutable a generar se debe ejecutar "make" desde un terminal estando ubicado en el mismo directorio del ejercicio 4.

Ej: $ make

Ejecucion del ejecutable "servidor":

- Para ejecutar "servidor" se deberá colocar "./servidor" en un terminal ubicado en la misma direccion que el ejecutable.

Parametros:

- El programa no lleva parametros, si se le coloca alguno no provocará ningun efecto en el programa.

Comportamiento:

- El servidor establece un enlace tcp/ip aceptando cualquier conexion.

- Las conexiones con los clientes las mantiene hasta que alguno de los dos no la cierre

- Cuando termina envia un informe con todos los datos refereridos a la partida

Envio de señales:

- Para terminar con el proceso correctamente se le debe enviar la señal SIGINT (ctrl+c)