Threads. Exercises

Exercise 1

Write.a program tht prints the arguments received by a thread.

Solution

```
#include <stdio.h>

#include <pthread.h>


void *f( void *arg){

int *v;

int i,j,k;

  printf ("Inicio ejecucion del thread\n");

   //Forma simple de acceder a los argumentos

   v=(int *)arg;

   printf ("TH:arg con v:%d, %d, %d:\n", v[0],v[1],v[2]);

    printf ("Fin ejecucion el thread\n");

}


Int main (){

 pthread_attr_t attr;

  int argu[4];

  pthread_t thid;


  argu[0]=99;

  argu[1]=11;
```

```
  argu[2]=22;
printf ("En el main:argu:%d, %d, %d:\n", argu[0],argu[1], argu[2]);


pthread_attr_init (&attr);
 //Puedes probar a crearlo DETACHED y esperar con un sleep
 // aunque es peor opcion
 //     pthread_attr_setdetachstate (&attr,  PTHREAD_CREATE_DETACHED);
         pthread_create (&thid, &attr, f, (void *)argu);
         pthread_join(thid,NULL);
}
```

Exercise 2

Write a program that creates 1 "print" caller thread that prints the message 3 times: "Thread says hi!", sleeps 1 second between each message and then indicates that it ends. The parent thread program must wait until the child thread terminates.

Solution

```
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

void * thread_function(void *arg) {
  int i;
  for ( i=0 ; i < 2 ; i++ ) {
    printf("Thread says hi!\n");
```

```
    sleep(1);
  }
      printf("Thread exists\n");
  return NULL;
}

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

  pthread_t mythread;
      printf("Launching new thread . . .\n");
  if ( pthread_create( &mythread, NULL, thread_function, NULL) ) {
    printf("error creating thread.");
    abort();
  }
      printf("Waiting on join\n");
  if ( pthread_join ( mythread, NULL ) ) {
    printf("error joining thread.\n");
    abort();
  }

  exit(0);
}
```

Exercise 3

This Exercise shows the problems that exist when a parent and child thread modify a global variable at the same time.

Threads. Exercises

Make a program that declares a global variable "myglobal" and creates 1 calling thread "print_point". The program then makes a for up to 20 in the increments myglobal by 1. Then it ends up printing the value of myglobal.

The function "print_point" executes a loop 20 where in each turn the value of myglobal is assigned to an auxiliary variable that is incremented. A "." Is printed. per lap. When finished, it sleeps 1sec and saves the auxiliary value in myglobal.

It asks:

a) Implement the program

b) Run the program and see what happens to the value of myglobal. Is it the same if it is run multiple times?

Solution

a)

```
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

int myglobal;

 void * imprimir_puntos(void *arg) {
  int i,j;
  for ( i=0; i<20; i++ ) {
    j=myglobal;
    j=j+1;
    printf(".");
    fflush(stdout);
   sleep(1);
    myglobal=j;
  }
  return NULL;
}

int main(void) {

  pthread_t mythread;
  int i;

  if ( pthread_create( &mythread, NULL, imprimir_puntos, NULL) )
{
```

```
    printf("error creating thread.");
    abort();
  }

  for ( i=0; i<20; i++) {
    myglobal=myglobal+1;
    printf("o");
    fflush(stdout);
    sleep(1);
  }

  if ( pthread_join ( mythread, NULL ) ) {
    printf("error joining thread.");
    abort();
  }

  printf("\nmyglobal equals %d\n",myglobal);

  exit(0);

}
```

b)

When modifying a global variable concurrently by parent and child thread, we cannot know what the final value will be. In each case it can be different, both in order of printing and in number. Below are 2 execution examples:

```
jesus$ ./a.out
o.o.o..o.o.o.o.o.o.oo..oo.o..o.oo..oo.o.
myglobal equals 21
jesus$ ./a.out
o.o..o.o.oo..oo..o.oo.o..o.o.o.o.oo.o..o
myglobal equals 23
```

Exercise 4

This Exercise shows how to create a variable number of threads, indicating the number with an argument. To do this, a function is created that prints "Hello from thread" and the thread number and ends.

The parent receives the number of threads (integer) as an argument and executes a for loop that creates those threads. Then wait for all threads to finish and finish.

Solution

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

#define MAX_THREAD 10

typedef struct {
        int id;
} parm;

void *hello(void *arg)
{
        parm *p=(parm *)arg;
        printf("Hello from thread %d\n", p->id);
        return (NULL);
}

int main(int argc, char* argv[]) {
        int n,i;
        pthread_t *threads;
        pthread_attr_t pthread_custom_attr;
        parm *p;

        if (argc != 2)
        {
                printf ("Usage: %s n\n  where n is no. of
threads\n",argv[0]);
                exit(1);
        }
        n=atoi(argv[1]);

        if ((n < 1) || (n > MAX_THREAD))
        {
                printf ("The no of thread should between 1 and
%d.\n",MAX_THREAD);
                exit(1);
        }

        threads=(pthread_t *)malloc(n*sizeof(*threads));
        pthread_attr_init(&pthread_custom_attr);

        p=(parm *)malloc(sizeof(parm)*n);
        /* Start up thread */
```

```
        for (i=0; i<n; i++)
        {
                p[i].id=i;
                pthread_create(&threads[i],
&pthread_custom_attr, hello, (void *)(p+i));
        }

        /* Synchronize the completion of each thread. */

        for (i=0; i<n; i++)
        {
                pthread_join(threads[i],NULL);
        }
        free(p);
        return 0;
}
```

Exercise 5

Write a program that creates 1 thread that adds the values passed to it as a parameter in an array of 10 integers and when it finishes it returns the calculated value.

Solution

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

void *suma(void *rango);



int main(){
```

```
pthread_attr_t attr;

pthread_t thread;

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

int *resultado;


    pthread_attr_init(&attr);

   // Creo el thread

    pthread_create(&thread,&attr,suma,&rango);

  // Espero la finalizaciÛn del thread

    pthread_join(thread,(void **)&resultado);

    printf("\nSuma  en Prog. Principal: %d\n",*resultado);

    return(0);

}


//Esta funciÛn la ejecuta el thread y realiza la suma del array
recibido en rango

void *suma(void *rango) {

   int i=0, *valores;

   int *suma; //variable de tipo puntero porque ser· devuelta al
main y si no lo fuera se eliminarÌa el valor calculado cuando
finalizara el thread


   valores= (int *)rango;

   suma=(int *)malloc (sizeof (int));

   *suma=0;

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

```
    *suma=*suma+valores[i];
  }
  printf("\tThread Suma : %d\n",*suma);
  pthread_exit(suma);
}
```

Exercise 6

Write a program that creates 10 threads. Each one of them calculates the value of the PI number using the Monte Carlo method and stores it in its corresponding position in an array. When all the threads have finished the main program calculates the average of the pi values stored in the array

Solution

```
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <math.h>


#define RADIO 5000

#define PUNTOS 1000000


//Variable global compartida por todos los threads, incluido el
main

float valoresPIthreads[10];


void *calcula_pi (void *kk);


main() {
```

```
	pthread_attr_t attr;
	pthread_t thread[10];
	int i;
	float *valorpi=0, suma=0, media=0;

	pthread_attr_init(&attr);
	for (i=0;i<10;i++) {
		pthread_create(&thread[i],&attr,calcula_pi,&i);
//Ponemos un sleep para que le de tiempo al thread a coger el
valor de i.
//Hay mejores mÈtodos que veremos m·s adelante.
//Puedes probar a quitar el sleep para ver que pasa
		sleep(1);
printf ("Creado thread %d\n",i);
	}
	for (i=0;i<10;i++) {
		pthread_join(thread[i],NULL);
	}
	for (i=0;i<10;i++) {
		printf("Valor del thread %d:
%f\n",i,valoresPIthreads[i]);
		suma=suma+valoresPIthreads[i];
	}
	media=suma/10.0;
	printf("El valor medio de Pi obtenido es: %f\n",media);
}
```

```
void *calcula_pi (void *idthread)
{
      int j, y=0, x=0, cont=0,numthread;
      float pi=0, h=0;
      numthread=*((int *)idthread);
printf ("Inicio th %d\n", numthread);
      srandom(pthread_self());
      for (j=0;j<PUNTOS;j++) {
            y=(random()%((2*RADIO)+1)-RADIO);
            x=(random()%((2*RADIO)+1)-RADIO);
            h=sqrt((x*x)+(y*y));
            if ( h<=RADIO ) cont++;
      }
      valoresPIthreads[numthread]=(cont*4)/(float)PUNTOS;
      pthread_exit(&pi);
}
```

Exercise 7

Write a program that declares 2 functions 1 and 2, where the identity of the thread that is executing it is indicated. function 1 sleeps 2 seconds and function 2 sleeps 5 seconds. Next the main program must launch 2 threads, one for each function, write its identity property and terminate.

Write a new version where the parent waits for the children to finish before finishing.

Solution

```
// fichero crearthreads.c
//THREADS
```

```
// compilar con  gcc -lpthread ej7-crearthreads.c
//Jesús Carretero

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/types.h>

pthread_t thread1, thread2, thmain; /* Declaración de los
threads */
pthread_attr_t attr; /* atributos de los threads */

/* Definición de las funciones func1 y func2 */
void *func1 ()
{
  pthread_t tid = pthread_self(); /* identificador de thread*/
  printf("Soy el thread 1 y voy a ejecutar func1 \n");
  sleep(2);
  printf("Soy el thread 1 y he terminado de ejecutar la función
1\n");
  pthread_exit(NULL); /* Provoca la terminación del thread*/
}
void *func2 ()
{
  pthread_t tid = pthread_self(); /* identificador de thread*/
  printf("Soy el thread 2 y voy a ejecutar func2 \n");
  sleep(5);
  printf("Soy el thread 2 y he terminado de ejecutar la función
2\n");
  pthread_exit(NULL); /* Provoca la terminación del thread*/
}


/*Función main*/
int main(void)
{
  thmain = pthread_self();
  /*La propia función main es un thread*/
  /*inicializa los parámetros de los threads por defecto*/
  pthread_attr_init (&attr);
  printf("Soy la función main y voy a lanzar los dos threads
\n");
  pthread_create (&thread1, &attr, func1, NULL);
  pthread_create (&thread2, &attr, func2, NULL);
  printf("Soy main: he lanzado los dos threads y termino\n");
  pthread_exit (NULL);
}
```

Exercise 8

Write a program that declares a multiply function and that passes 2 numbers as parameters. Then the main program must prepare the parameters of the thread and launch 1 thread with multiply, write its identity property and terminate.

Solution

```
// fichero ej8-multiplicarthread.c
//THREADS
/* Realizar un programa que declare una función multiplicar y
que le pase como parámetros 2 números en una estructura
A continuación el programa principal debe preparar los
parámetros del thread y lanzar 1 thread con func1, escribir su
propiedad identidad y terminar. */
// compilar con  gcc -lpthread ej8-multiplicarthread.c
//Jesús Carretero

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

// Estructura que contiene los datos a pasar como parámetros
// Un único parámetro se puede pasar directamente con el
// operador &

typedef struct
{
   int dato1,dato2;
}datos;

pthread_t thread1, thmain; /* Declaración de los threads */
pthread_attr_t attr; /*atributos de los threads*/

/* Definición de las función func1 */
void *multiplicar (void *arg)
{
  int a,b;
  datos *p= (datos *) (arg);
  pthread_t tid = pthread_self(); /*identificador de thread*/

  a=(p->dato1);
  b=(p->dato2);
  printf("Soy el thread 1 y voy a multiplicar \n");
```

```
  printf("La multiplicación es %d\n",a*b);
  printf("Soy el thread 1 y he terminado de multiplicar \n");
  pthread_exit (NULL);
}


/*Función main*/
int main(int argc, char* argv[])
{
  datos param;
  param.dato1=atoi(argv[1]);
  param.dato2=atoi(argv[2]);

  thmain = pthread_self();

  pthread_attr_init (&attr);
  printf("Soy la función main y voy a lanzar el thread \n");
  pthread_create (&thread1, &attr, func1, &param);
  printf("Soy main: he lanzado el thread y termino\n");
  pthread_exit (NULL);
}
```

Exercise 9

Write a program that declares a multiply function and passes 2 numbers to multiply as parameters. Next, the main program must take the number whose table we want, and execute a loop launching threads that print its multiplication table. In each iteration you must prepare the thread parameters and launch 1 thread with multiply. At the end write your identity property and finish.

Solution

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

// Estructura que contiene los datos a pasar como parámetros
// Un único parámetro se puede pasar directamente con el
// operador &

typedef struct
```

```
{
   int dato1,dato2;
}datos;

pthread_t thread1, thmain; /* Declaración de los threads */
pthread_attr_t attr; /*atributos de los threads*/


/* Definición de las función multiplicar */
void *multiplicar (void *arg)
{
  int a,b;
  datos *p= (datos *) (arg);
  pthread_t tid = pthread_self(); /*identificador de thread*/

  a=(p->dato1);
  b=(p->dato2);
  printf("%d por %d es %d\n",a, b, a*b);
  pthread_exit (NULL);
}


/*Función main*/
int main(int argc, char* argv[])
{
  datos param;
  int i;

  param.dato1=atoi(argv[1]);
  thmain = pthread_self();

  pthread_attr_init (&attr);
  printf("Soy la funcion main. Tabla de multiplicar del %d \n",
param.dato1);
  for (i=0; i<= 10; i++) {
      param.dato2=i;
      pthread_create (&thread1, &attr, multiplicar, &param);
      sleep(1);
  }
  printf("Soy main pid= %d: he lanzado los thread y termino\n",
getpid());
  pthread_exit (NULL);
}
```

Exercise 10

Make a program that declares a print function and passes it as parameters 1 string to print. Then the main program must prepare the parameters with 2 strings "hello" and "world" and launch 2 threads that try to print "hello world" and finish.

Solution

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>


pthread_t thread1, thread2; /* Declaración de los threads */
pthread_attr_t attr; /*atributos de los threads*/

/* Definición de las función imprimir */
void *imprimir (void *arg)
{
  char a[12];
  strcpy(a, (char*)arg);

  printf("%s \n",a);
  pthread_exit (NULL);
}

/*Función main*/
int main (void)
{
  char cadena_hola[]="Hola ";
  char cadena_mundo[]="mundo ";

  pthread_attr_init (&attr);
  pthread_create(&thread1, &attr, imprimir, (void
*)cadena_hola);
  pthread_create(&thread2, &attr, imprimir, (void
*)cadena_mundo);
  pthread_exit (NULL);
}
```

Exercise 11

Write a program that adds values in concurrency using threads. The program declares a global variable sum_total and an addition procedure that increments sum_total by 100 using an intermediate local variable, sleeps for a second, and assigns the internal variable to sum_total.

The principal then creates 10 add threads, waits for them to finish, prints the computed sum value, and terminates.

As a variant, eliminate sleep time and indicate what happens.

Solution

a) Programa sumador con concurrencia

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/types.h>


#define NUMTH 10
int suma_total = 0;

void sumar() {
  int i;
  int suma=suma_total;
  suma = suma + 100;
  sleep(1);
  printf("Pthread =%d  despierta \n", (int)pthread_self());
  suma_total=suma;
}


int main() {
  pthread_t th[NUMTH];
  int i;
  for (i=0;i<NUMTH;i++) {
    pthread_create(&th[i], NULL,(void*)sumar, NULL);
  }

  for (i=0;i<NUMTH;i++) {
    pthread_join(th[i], NULL);
  }

  printf("Suma total = %d\n",
    suma_total);
}
```

a) Sin sleep hay problemas de concurrencia. Como se puede ver a continuación.

```
Pthread =61386752  despierta
Pthread =61923328  despierta
Pthread =62459904  despierta
Pthread =62996480  despierta
Pthread =63533056  despierta
Pthread =64069632  despierta
Pthread =64606208  despierta
Pthread =65142784  despierta
Pthread =65679360  despierta
Pthread =66215936  despierta
Suma total = 200
```

EXERCISE 12

Make a program that creates 1 thread that adds the values passed to it as a parameter in an array of 10 integers (1,2,3,4,5,6,7,8,9,10) and when it finishes it returns the value calculated.

SOLUTION

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>


void *suma(void *rango);


int main() {

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

    int *resultado;


    pthread_t thread;
```

```
    pthread_attr_t attr;

    pthread_attr_init(&attr);

    pthread_create(&thread, &attr, suma, &rango);   /*Creo el
thread*/

    pthread_join(thread, (void **)&resultado); /*Espero la
finalizacion del thread*/

    printf("\nSuma  en Prog. Principal: %d\n", *resultado);

    return(0);

}


void *suma(void *rango) {

    int i = 0;

    int *valores;

    int *suma;

    valores = (int *)rango;

    suma = (int *)malloc (sizeof(int));

    *suma = 0;

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

     *suma = *suma + valores[i];

    }

    printf("\tThread Suma : %d\n", *suma);

    pthread_exit(suma);

}
```

EXERCISE 13

Threads. Exercises

Make a program that creates 10 threads. Each thread receives by parameter a number n that it must print on the screen. The number n received will be in the range [0-9] according to the thread created.

SOLUTION

```
#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <pthread.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>


#define MAX_THREADS 10

void *funcionThread(void *numero);

int main(){

    int j;

    int k;

    pthread_t thid[MAX_THREADS]; /*array con los identificadores
de los threads que se van a crear*/

    pthread_attr_t attr; /*variable con los atributos de los
threads que se van a crear*/

    pthread_attr_init(&attr); /*se inicializan los atributos (a
por defecto)*/

    /*bucle para crear los distintos threads*/

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

    {
```

```
     pthread_create(&thid[j], &attr, (void *) funcionThread,
&j);

     sleep(1); /*se duerme el proceso para que el thread recien
creado le de tiempo a ejecutar*/

    }

    /*bucle para esperar por la finalizacion de los distintos
threads*/

    for(k = 0; k < MAX_THREADS; k++)

    {

     pthread_join(thid[k], NULL);

    }

    return(0);

}


void *funcionThread(void *numero) {

    sleep(1); /*si se duerme el thread, no le da tiempo a coger
el valor del parametro*/

    int valor = *((int *)(numero));

    printf("Thread con numero %d \n", valor); /*imprime el
numero que recibe por parametro*/

    pthread_exit(0); /*hace que termine el thread y se mande el
estado de terminacion al padre*/

}
```

EXERCISE 14

Given the following code that:

- ♣ Create a (determined) number of threads. Each thread prints its identifier and ends.

♣ The parent process waits with the join function for the completion of the threads.

You are asked to modify this program to calculate the sum of randomly generated integers. First each child generates a random integer and stores it in a global array. When the children finish the father calculates the sum of all the integers generated by the threads and prints it on the screen. Use the random () function to generate a random number:

int numeroAleatorio = (int)(random( ) % 10);

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define MAX_THREADS 10

void *funcionThread(void *numero);

int main(){
{
    int j, k;
    pthread_t thid[MAX_THREADS]; //array con los identificadores de los threads que se van a crear
   pthread_attr_t attr; //variable con los atributos de los threads que se van a crear
   pthread_attr_init(&attr); //se inicializan los atributos (a por defecto)
   //bucle para crear los distintos threads
```

```
    for(j = 0; j < MAX_THREADS; j++)
    {
       pthread_create(&thid[j], &attr, (void *)funcionThread, NULL);
    }
    //bucle para esperar por la finalización de los distintos threads
    for(k = 0; k < MAX_THREADS; k++)
    {
       pthread_join(thid[k], NULL);
    }
    return(0);
}

void *funcionThread(void)
{
    printf("Thread con identificador  = %ud \n", (int) pthread_self());
//imprime el id del thread
    pthread_exit(0); //hace que termine el thread y se mande el estado
de terminación al padre
}
```

SOLUTION

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```
#define MAX_THREADS 10


int arrayEnteros[MAX_THREADS];


void *funcionThread(void *numero);


int main() {

    int j, k;

    int sumaAcumulada = 0;


    pthread_t thid[MAX_THREADS]; //array con los identificadores
de los threads que se van a crear


    pthread_attr_t attr; //variable con los atributos de los
threads que se van a crear

    pthread_attr_init(&attr); //se inicializan los atributos (a
por defecto)


    //bucle para crear los distintos threads

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

    {

     pthread_create(&thid[j], &attr, (void *)funcionThread, &j);


     sleep(1); //se duerme el proceso para que el thread recien
creado le de tiempo a ejecutar

    }
```

```
    //bucle para esperar por la finalizacion de los distintos
threads
    for(k = 0; k < MAX_THREADS; k++)
    {
     pthread_join(thid[k], NULL);

     sumaAcumulada += arrayEnteros[k];
    }

    printf("La suma acumulada es: %d\n", sumaAcumulada);

    return(0);
}

void *funcionThread(void *numero) {

    int posicion = *((int *)(numero));

    int numeroAleatorio = (int)(random() % 10);

    printf("El thread de la posicion %d genero el numero: %d\n",
posicion, numeroAleatorio);

    arrayEnteros[posicion] = numeroAleatorio;
```

```
    pthread_exit(0); //hace que termine el thread y se mande el
estado de terminacion al padre

}
```

Exercise 15

Write a program that creates 10 threads to add the data from a file "numbers.dat" that contains 1000 whole numbers. Each thread must add 100 numbers from the file. Thread n will add the numbers between n * 100 and (n * 100) +99 from the array (n varies from 0 to 9). When they finish the main program will write the total sum.

Solution

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
void *suma(void *rango);

pthread_attr_t attr;
int f=0;
pthread_t thread[10];

int main() {
    int i=0, n=0, rango=0, *estado, pestado=0, nbytes=0, nreg=0;
    estado=&pestado;
    pthread_attr_init(&attr);

    if((f=open("numeros.dat", O_RDONLY))==-1) {
        fprintf(stderr,"Error en la apertura del fichero\n");
        return(-1);
    }
    nbytes=lseek(f,0,SEEK_END);
    nreg=nbytes/sizeof(int);
    for(i=0;i<10;i++) {
        pthread_create(&thread[i],&attr,suma,&rango);
        sleep (1);
        rango+=100;
    }
```

```
    for(i=0;i<10;i++) {
      pthread_join(thread[i],(void **)&estado);
      printf("Suma Parciales en Prog. Principal: %d\n",*estado);
      n+=*estado;
    }
    printf("Suma Total: %d\n",n);
    printf("Total numeros sumados: %d\n",nreg);
    close(f);
    return(0);
}



void *suma(void *rango) {
int j=0, valor, *suma, num=0;

//sleep(1);
    valor=* ((int *) rango);
    suma=(int *)malloc (sizeof (int));
   *suma=0;
    printf("Rango: %d a %d\n",valor+1,valor+100);
    lseek(f,valor * sizeof(int),SEEK_SET);
    for(j=0;j<100;j++) {
        read(f,&num,sizeof(int));
        *suma+=num;
    }
    printf("\tSuma Parcial: %d\n",*suma);
    pthread_exit(suma);
}
```