

# Instituto Politécnico Nacional Escuela Superior de Cómputo



## Sistemas Operativo Practica 5

### HILOS (THREADS) EN LENGUAJE C ENFOCADOS A LINUX

#### *Integrantes:*

*Ayona López Eugenio Milton*

*Morales Blas David Israel*

#### *Profesora:*

JUÁREZ MÉNDEZ ANA BELEM

Grupo: 2CM8

Fecha de entrega: 27 de marzo del 2020

# Objetivo

Utilizar los conocimientos de hilos en aplicaciones de sistemas operativos.

## Introducción

Un hilo es una manera de programar, que se puede considerar demasiado eficiente, debido al principal factor que nos proporciona, lo cual es correr tareas casi simultaneas, esto lo digo porque una computadora no puede ejecutar dos o mas tareas a la vez, hay un cierto margen en cada “proceso ligero”, pero es tan pequeño que nos da la ilusión de simultaneidad.

Si vamos a conocimientos anteriores y nos posicionamos en el punto en donde vimos el uso de la función `fork()`, notamos que hay varias similitudes y que hasta podríamos llegar a pensar que son las mismas cosas (lo cual yo llegue pensar al inicio), pero esto no se debe ver así, ya que puede ocasionar demasiados problemas a la hora de querer implementar ya sea los procesos y los hilos.

La mejor manera que yo encontré para encontrar las diferencias, fue claramente llenarme de información, que después utilizaba para visualizar estos conceptos al momento de empezar a programar. Pero en general quiero describir las diferencias que note en la parte de abajo...

Para crear un hilo es necesario usar la función siguiente:

**`int pthread_create(pthread_t *thread, pthread_attr_t *attr, void*(*start_routine)(void*),void *arg);`**

- ◆ El primer argumento apunta al identificador del hilo.
- ◆ El segundo argumento especifica los atributos asociados al nuevo hilo.
  - Si es NULL se utilizan los atributos predeterminados lo cual incluye ser un hilo dependiente.
- ◆ El tercer argumento indica el nombre de la función a ejecutar por el hilo.
- ◆ El cuarto argumento es un único parámetro que puede pasársele a la función.

Para poder observar el resultado de los hilos es necesario usar la función `pthread_join`, esta función cabe destacar sólo se usa en hilos independientes es decir el segundo parametro de `pthread_create` es NULL.

Por ultimo es necesario cerrar el hilo cuando se termine de usar esto se hace con la función **`int pthread_exit(void *value);`**

### DIFERENCIAS ENTRE UN PROCESO Y HILO:

- Un proceso es algo más complejo.
- En los procesos la memoria esta limitada, ya que se establece un bloque en donde este solo puede trabajar y en los hilos la memoria es independiente.
- Trabajar con hilos se me facilito mas debido a que podemos separar lo que nosotros queramos ejecutar en el en una función independiente, tanto podemos utilizar la misma función como crear otras para los hilos que creamos; Por tanto, la manera en como un proceso trabaja puede ser mas engorrosa es decir al llamar a `fork` n veces se crean  $2^N$  procesos donde n es el numero de llamadas a la función, controlar cuantos procesos hijos se requieren necesita 2 fors y su control es más complicado de cierta forma.

- El tiempo de ejecución en un proceso es mucho más lento que un hilo.

#### **SIMILITUDES ENTRE HILO Y PROCESO:**

- Ejecución de tareas casi simultáneas.
- Programación de los tiempos en que los hilos o procesos son sincronizados.
- Comunicación un poco más abstracta con nuestro sistema operativo.
- Procesamiento de señales.

En general los hilos y procesos son dos estilos que puedes utilizar independientemente a la hora de la resolución de problemas que implican el uso de tareas simultáneas.

## **DESARROLLO DE LA PRACTICA**

*El ejercicio 1 cuenta con dos soluciones debido a que en una no se aprecia bien como cada hilo imprime la palabra, en la segunda solución si es posible observar, además en uno los valores se piden desde el main y en otro se mandan desde ejecución.*

### **Ejercicio 1 Solución 1**

Se nos ha dado la tarea de resolver la practica 5 en la cual incluye 4 problemas, que implican el uso de hilos, los que a continuación se describen:

*Problema 1.* \_ Dado dos mensajes M1 y M2, los cuales respectivamente se deben pasar en dos hilos H1 y H2 y como último paso cada hilo debe imprimir en la pantalla el mensaje respectivo. Además de agregar al final la palabra FIN cuando se terminen de ejecutar los dos procesos.

-Este problema es fundamental, ya que nos enseña el uso, creación y el efecto que tiene el uso de hilos.

#### **Aspectos a tomar en cuenta**

- Como pasar parámetros al momento de crear el hilo

- Tiempo de espera para que un hilo realice su tarea **Código C**

```

1  #include <pthread.h>
2  #include <stdlib.h>
3  #include <stdio.h>
4  #include <string.h>
5  #include <unistd.h>
6  #include <pthread.h>
7
8  /*
9   struct parametros
10  {
11      int id hilo;
12      char M1[12];
13      char M2[12];
14  };
15
16
17      int cont = 0;
18  */
19
20
21  void* hilos_muestra(void* para)
22  {
23      //struct parametros* p = (struct parametros*)para;
24      int i=0;
25      for(i=0;i<strlen(para);++i){
26          printf("Soy tu char: %c\n", *(char*)(para+i));
27          fflush ( stdout );
28          usleep (10000);
29      }
30  }
31
32
33  int main()
34  {
35
36      //struct parametros datos;
37      char M1[12];
38      char M2[12];
39      /*Creacion de hilos*/
40      pthread_t thread1_id;
41      pthread_t thread2_id;
42
43      printf("Ingrese Primer mensaje\n");
44      scanf("%s",M1);
45
46      printf("Ingrese Segundo mensaje\n");
47      scanf("%s",M2);
48
49      /* Creamos un hilo y esta funcion genera un valor que usamos para identificar al hilo*/
50      pthread_create(&thread1_id, NULL, &hilos_muestra,M1);
51      pthread_create(&thread2_id, NULL, &hilos_muestra,M2);
52
53      /*Terminacion del hilo*/
54      pthread_join(thread1_id, NULL);
55      pthread_join(thread2_id, NULL);
56
57      printf ( " Fin \n ");
58
59  }

```

```

1  #include <pthread.h>
2  #include <stdlib.h>
3  #include <stdio.h>
4  #include <string.h>
5  #include <unistd.h>
6  #include <pthread.h>
7
8  /*
9  struct parametros
10 {
11     int id_hilo;
12     char M1[12];
13     char M2[12];
14 };
15
16 int cont = 0;
17 */
18
19 void* hilos_muestra(void* para)
20 {
21     //struct parametros* p = (struct parametros*)para;
22     int i=0;
23     for(i=0;i<strlen(para);i++){
24         printf("Soy tu char: %c\n", *(char*)(para+i));
25         fflush ( stdout );
26         usleep (1000);
27     }
28 }
29
30 int main()
31 {
32     //struct parametros datos;
33     char M1[12];
34     char M2[12];
35     /*Creacion de hilos*/
36     pthread_t thread1_id;
37     pthread_t thread2_id;
38
39     printf("Ingrese Primer mensaje\n");
40     scanf("%s",M1);
41
42     printf("Ingrese Segundo mensaje\n");
43     scanf("%s",M2);
44
45     /* Creamos un hilo y esta funcion genera un valor que usamos para identificar al hilo*/
46     pthread_create(&thread1_id, NULL, &hilos_muestra,M1);
47     pthread_create(&thread2_id, NULL, &hilos_muestra,M2);
48
49     /*Terminacion del hilo*/
50     pthread_join(thread1_id, NULL);
51     pthread_join(thread2_id, NULL);
52
53     printf ( " Fin \n ");
54 }

```

**Bibliotecas utilizadas en el programa**

**Función ejecutada por el hilo**

**Variables**

**Ingreso de datos**

**Invocación de los hilos y envío de datos**

**Terminación del hilo**

**Descripción:**

## Bibliotecas necesarias para uso de hilos

```
#include <pthread.h>
```

## Variables

Podemos observar que se encuentran uno que ya conocemos que es char y uno que no es muy conocido llamado pthread\_t.

pthread\_t: Variable ubicada en pthread utilizada para la invocación de un hilo.

## Ingreso de datos

Aquí ya es un uso mas funcional en nuestra práctica, ya que en esta sección pedimos los mensajes que nuestro usuario va ingresar, que este caso pueden tener una longitud máxima de 12 caracteres.

## Invocación de los hilos y envio de datos

Al igual que la variable `pthread_t`, nuestra funcion `pthread_create` se encuentra en la biblioteca `pthread`.

`pthread_create` : Funcion encargada de invocar nuestro hilo, recibe cuatro parametros, que se enlistan abajo:

1. Se encarga de recibir la dirección de nuestro hilo en especifico.
2. Personalizacion de los atributos que nuestro hilo va a tener, como damos un NULL, se generan los atributos que vienen por defecto.
3. Recibi la direccion de la función que nuestro hilo va ejecutar, en este la funcion es **hilos\_muestra**
4. Los parametros que va a mandar a nuestra funcion que se ejecuta en el hilo, en este caso son los mensajes que el usuario ha ingresado.

## Función ejecutada por el hilo

La parte mas importante de nuestro código es esta debido a que aquí se encuentra la lógica de nuestro programa.

Uso: solo muestra en pantalla el mensaje que se envió en cada hilo.

Indicaciones

- Debido a que nuestro mensaje se encuentra en un apuntador a un tipo void, hay que castearlo para que se revele el mensaje lo cual hacemos con este código `*(char*)(para+i)`

Un aspecto a tomar en cuenta es `(para+i)` talvez esta no sea una manera muy conocida, pero tiene el mismo funcionamiento a si estuviéramos haciendo `M[indice]`.

- `strlen()` esta función nos devuelve el tamaño de nuestro mensaje, esta recibe como parámetros un apuntador en este caso al mensaje, nos daría la longitud del mensaje, la que usamos como condición en nuestro "for", para que imprima carácter por carácter, hasta el tamaño final.
- Y por ultimo esta las funciones `usleep()` y `fflush`, que se encarga de dormir a nuestro hilo unos milisegundos con el fin de observar el funcionamiento casi simultaneo de nuestro hilo y para limpiar la salida de nuestro buffer, respectivamente.

## SALIDA EN CONSOLA

```
C:\Users\eugen\OneDrive\Escritorio\Sistemas Operativos\P_5_1_MensajeHilos.exe

Ingrese Primer mensaje
hola
Ingrese Segundo mensaje
adios
Soy tu char: h
Soy tu char: a
Soy tu char: o
Soy tu char: d
Soy tu char: i
Soy tu char: l
Soy tu char: o
Soy tu char: a
Soy tu char: s
Fin

-----
Process exited after 17.58 seconds with return value 7
Presione una tecla para continuar . . .
```

### Ingreso de los mensajes

Se supone que la programación sin hilos, debería mostrar un mensaje parecido a "hola adios", pero como anteriormente explique, esto sucede debido al comportamiento del hilo, ya que como se están ejecutando simultáneamente, su salida es aleatoria, ya que la computadora pasa de manera casi simultánea de hilo a hilo, provocando este efecto.

## EJERCICIO 1 Solucion 2

Representación del mismo problema pero con un código donde se puede apreciar de mejor manera como cada hilo ejecuta la palabra carácter por carácter.

Código en c:

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

void gotoxy(int, int);
void *saludo(void *);

typedef struct data{
    char *cadena;
    int x,y;
}parametro;

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

    //char *cadena = "hola mundo";
    pthread_t hilo1;
    pthread_t hilo2;

    parametro p1;
    p1.cadena=argv[1];
    p1.x=5;
    p1.y=2;

    parametro p2;
    p2.cadena=argv[2];
    p2.x=30;
    p2.y=2;

    pthread_create(&hilo1, NULL, saludo, (void *)&p1);
    pthread_create(&hilo2, NULL, saludo, (void *)&p2);
    pthread_join(hilo1, NULL);
    pthread_join(hilo2, NULL);
    printf("\nFIN\n");

    return 0;
}

void gotoxy(int x, int y){
    printf("\033[%d;%df", y, x);
}

void *saludo(void *args){
    system("clear");
    parametro *par = (parametro *) args;
    //char *cadena= (char *) args;
    int i;
    for(i=0; i<strlen(par->cadena); i++){
        fflush(stdout);
        gotoxy(par->x, par->y);
        par->x++;
        printf("%c", par->cadena[i]);

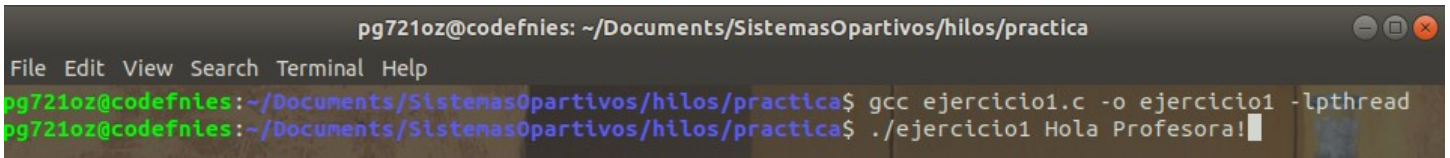
        sleep(1);
    }
    printf("\n");
    pthread_exit(NULL);
}
```

Como se puede observar la función gotoxy es la encargada de cambiar la posición de impresión de la función \*saludo

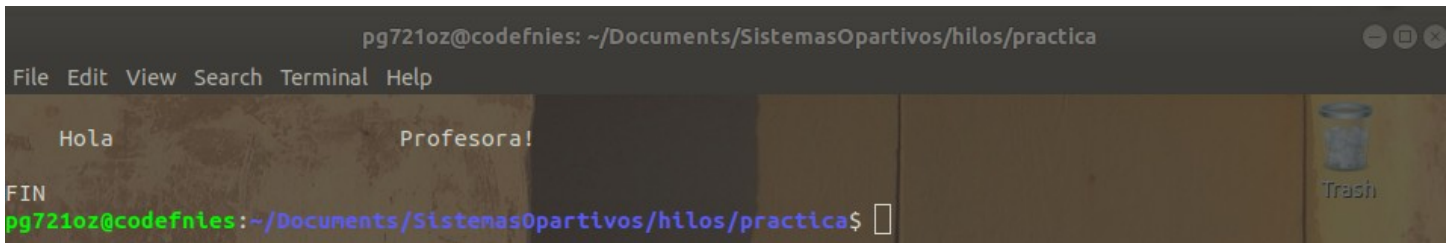
Aquí los valores se piden desde la ejecución es decir char \*argv[] es el encargado de tomar los dos valores.



## Resultados en pantalla código 1 solución 2.



```
pg721oz@codefnies: ~/Documents/SistemasOpartivos/hilos/practica
File Edit View Search Terminal Help
pg721oz@codefnies:~/Documents/SistemasOpartivos/hilos/practica$ gcc ejercicio1.c -o ejercicio1 -lpthread
pg721oz@codefnies:~/Documents/SistemasOpartivos/hilos/practica$ ./ejercicio1 Hola Profesora!
```



```
pg721oz@codefnies: ~/Documents/SistemasOpartivos/hilos/practica
File Edit View Search Terminal Help
Hola Profesora!
FIN
pg721oz@codefnies:~/Documents/SistemasOpartivos/hilos/practica$
```

## Conclusión ejercicio 1 solucion 2

Como se puede observar el código limpia la pantalla con la función `system("clear")`; por ello en la screenshot se observa que se escribe lo de los dos hilos en la parte de arriba y posterior escribe la palabra fin una vez que los hilos se terminaron de ejecutar, cabe destacar que dar un tamaño de palabra como argumento muy grande debido a que la impresión se encuentran en la misma línea sobrescribirá en la salida la letra, es decir no se podría apreciar bien, pero su funcionamiento sigue siendo correcto.

## Ejercicio 2.

### CÓDIGO C

```
1  #include<stdlib.h>
2  #include<stdio.h>
3  #include<string.h>
4  #include<unistd.h>
5  #include<pthread.h>
6  #include<math.h>
7
8  struct datos{
9      int suma; //almacena la suma total de cada hilo
10     int ciclo; //Veces en que E1 se sumara
11     int E1; //Numero de E1 para que el valor que se sumara n ciclos
12 };
13 int hilo=0;
14 void* hilos_muestra(void* para)
15 {
16     struct datos* p = (struct datos*)para;
17     printf("E1= %d se sumara: %d veces en este hilo\n ",p->E1, p->ciclo);
18
19     int i;
20     for(i=1;i<=(p->ciclo); ++i){
21         p->suma+= p->E1;
22     }
23     ++hilo;
24     printf("HILO:%d Suma= %d\n\n",hilo,(p->E1*p->ciclo));
25 }
26
27 int main(int argc, char *argv[])
28 {
29     struct datos parametros;
30     int N[3]; //usado para almacenar los datos que recibimos desde consola
31
32     int cad, contcad, reinicio;
33     char numeros[3][10]; //el primer arreglo indica los numeros que vamos a recibir el segundo son los digitos de cada numero
34
35     for(cad=0, contcad=0; argv[1][contcad] != 0; ) {
36         for(reinicio=0; argv[1][contcad] != '*'; ++reinicio)
37         {
38             numeros[cad][reinicio] = argv[1][contcad];
39             ++contcad;
40         }
41         ++contcad;
42         ++cad;
43     }
44     // Use c as you need
45
46     int num;
47     for(num=0; num<3; ++num){
48         N[num] = atoi(numeros[num]);
49     }
50     printf("E1: %d E2: %d N: %d\n", N[0], N[1], N[2]);
51 }
```

**Bibliotecas que utilizamos**

**Estructura con los datos que utiliza el hilo.(datos)**

**Funcion que muestra la suma de los hilos y realiza el "ciclo de trabajo" mostrando el aporte en cada hilo.(hilos\_muestra)**

**Variables**

**Analiza los datos ingresados desde consola, guardandolo en un arreglo.**

**Convierte el arreglo que obtuvimos desde la consola a un tipo util (int) y muestra los datos que ingresamos.**

## ANALIZA LOS DATOS INGRESADOS DESDE CONSOLA, GUARDANDOLO EN UN ARREGLO

Esta es una parte fundamental de nuestro programa la cual se debe poner demasiada atención para

**Ejemplo:**

Tenemos la cadena 343\*234\*123\*

En nuestro arreglo `argv[1][contcad]` se vería de esta manera

3	4	3	*	2	3	4	*	1	2	3	*	0
---	---	---	---	---	---	---	---	---	---	---	---	---

En cada ciclo del for externo recorre esa cadena, y dentro el for interno el arreglo *números* se *visualiza de esta manera*:

[illegible]

Pero en general esta es la forma en que trabaja:

1er ciclo:

Diagram illustrating a sequence of numbers and symbols: 3, 4, 3, \*, 2, 3, 4, \*, 1, 2, 3, \*, 0. An orange arrow points to the first '3'. Below this sequence is a 3x10 grid. The first cell of the first row contains the number '3'. The rest of the grid is empty.

Los demás recorridos del arreglo argv los hace el for interno con ++cont

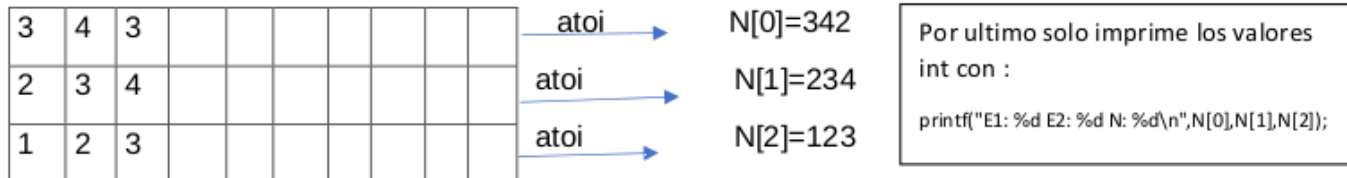
[illegible]

The diagram illustrates a dynamic programming problem. At the top, a 1D array of size 12 is shown with the following elements: 3, 4, 3, \*, 2, 3, 4, \*, 1, 2, 3, \*. An orange arrow points to the third element (3). Below the array is a 3x10 grid. The first row of the grid contains the values 3, 4, 3, followed by seven empty cells. The remaining two rows are entirely empty.



## CONVIERTE EL ARREGLO QUE OBTUVIMOS DESDE LA CONSOLA A UN TIPO UTIL (INT) Y MUESTRA LOS DATOS QUE INGRESAMOS.

Aquí utilizamos el arreglo números para convertirlo a un tipo más útil en este caso la función atoi resulta de mucha utilidad, ya que convierte un apuntador a char que es la dirección de una cadena a un número tipo int; por tanto nosotros al hacer esto `numeros[num]`, le estamos pasando las direcciones de nuestras cadenas.



## PRUEBA QUE EL NUMERO N NO SEA MAYOR QUE E2

Esto es muy importante debido que si no cumple esta condición  $E2/N$  siempre dará un valor 0 lo cual hace que siempre su ciclo de trabajo siempre sea 0, lo que indica que siempre sumara 0.

## VARIABLES UTILIZADA EN EL ANALISIS DE LAS SUMAS

Estas variables se utilizan en la parte mas importante de nuestro código la cual hace que se divida el ciclo de trabajo para cada hilo, a continuación, se describen:

*i*: Variable que indica en que hilo se encuentra nuestro programa va desde  $0 \dots N$ .

**partent**: No es de tanta importancia en este programa, solo es un requisito que la función `modf()`, no pide para almacenar la parte entera de un número con decimales.

<b>parametros.ciclo</b> <b>parametros.E1</b> <b>parametros.suma</b>	}	Variables previamente explicadas en la estructura datos, aquí solo las inicializamos, para no obtener valores erróneos durante su uso, de la siguiente manera:  <code>parametros.ciclo=N[1]/N[2]; parametros.E1=N[0]; parametros.suma=0;</code>
---	---	---

## PARTE ENGARGADA DE DIVIDIR EL TRABAJO DE CADA HILO

Esta sección del código es la fundamental, para que el programa, realice correctamente su funcionalidad, se divide en las siguientes partes.

- Si el modulo de E2 y N es 0, entonces el ciclo de trabajo será igual en todos los hilos.

$$H1 \quad \dots \quad N$$

E1	E1	E1	E1	E1	E1	E1	E1
----	----	----	----	----	----	----	----

- Si el módulo de E2 y N es diferente a 0, el ciclo de trabajo será igual hasta N-1, en N el ciclo de traba es diferente y es dado por:

$$Ciclo_{Hn} = Ciclo_{Hn-1} + (N * P_d) + 1e-9$$

*Ciclo<sub>Hn</sub> = Cantidad el ciclo de trabajo en el ultimo ciclo*

*Ciclo<sub>Hn-1</sub> = Cantidad del ciclo de trabajo cuando el trabajo es el mismo en cada ciclo*

*Ciclo<sub>Hn</sub> = Cantidad el ciclo de trabajo en el ultimo ciclo*

*N = cantidad de hilo ingresados*

*P<sub>d</sub> = Parte decimal del ciclo de trabajo que se trunca, cuando se hace la division entera*

*1e - 9 = Factor de correcion para que se redonde nuestro numero*

Visualmente quedaría así nuestro ciclo de trabajo para cada hilo

$H_{1.....H_{N-1}} = parte\_enter(\frac{E2}{N})$							$H_N = Nparte\_decima(\frac{E2}{N})$
E1	E1	E1	E1	E1	E1	E1	E1

Esto queda demasiado claro en los if que están en el código en las líneas de código que está aquí:

-parametros.ciclo=N[1]/N[2], para ciclo de trabajo iguales

- parametros.ciclo= parametros.ciclo + (N[2]\*modf((double)N[1]/N[2], &partent)) + 1e-9, para el último ciclo de trabajo, cuando el modulo es diferente a 0 y se encuentra en el ultimo hilo.

- En cada if y else se manda invocar el hilo pthread\_create(&thread[i], NULL, &hilos\_muestra,&parametros), con parámetros como la dirección de cada hilo( la cual cambia mientras i varia), atributos (en este caso los por defecto), la dirección de nuestra función que se describirá su funcionamiento abajo y por ultimo los parámetros que se encuentran en nuestra estructura.
- Hay que tomar en cuenta como son varios hilos que se están ejecutando casi simultáneos, es necesario que se le ponga un sleep para que cada hilo haga su tarea en el debido tiempo.

### **FUNCION QUE MUESTRA LA SUMA DE LOS HILOS Y REALIZA EL "CICLO DE TRABAJO" MOSTRANDO EL APORTE EN CADA HILO.(HILOS\_MUESTRA)**

Esta función solo se encarga de sumar todas las aportaciones de los hilos, en la variable que suma que se encuentra adentro de la estructura datos, lo cual hacemos aquí:

```
int i;
```

```
for(i=1;i<=(p->ciclo); ++i){
```

```
p->suma+= p->E1;
```

```
}
```

, mostrar cuantas veces realizara cada ciclo de trabajo, lo cual se encuentra aquí.

```
printf("E1= %d se sumara: %d veces en este hilo\n",p->E1, p->ciclo);
```

y por ultimo mostrar la aportación actual de cada hilo a la variable suma, que lo hacemos en esta parte:

```
printf("HILO:%d Suma= %d\n\n",hilo,(p->E1*p->ciclo));
```

la variable hilo solo se utiliza para mostrar el hilo actual.



## SALIDA EN PANTALLA DEL PROGRAMA

Compilación:

```
ayonx12@ubuntu: ~/Desktop/Sistemas
File Edit View Search Terminal Help
ayonx12@ubuntu:~/Desktop/Sistemas$ ls
Cont ContadorHilos.c Sumas SumasHilos.c term
ayonx12@ubuntu:~/Desktop/Sistemas$ gcc -pthread -o Sumas SumasHilos.c
ayonx12@ubuntu:~/Desktop/Sistemas$
```

Ejecución con datos  $34 \times 12 \times 10^*$

```
File Edit View Search Terminal Help
Cont ContadorHilos.c Sumas SumasHilos.c term
ayonx12@ubuntu:~/Desktop/Sistemas$ ./Sumas 34*12*10*
E1= 34 E2: 12 N: 10
E1= 34 se sumara: 1 veces en este hilo
HILO:1 Suma= 34

E1= 34 se sumara: 1 veces en este hilo
HILO:2 Suma= 34

E1= 34 se sumara: 1 veces en este hilo
HILO:3 Suma= 34

E1= 34 se sumara: 1 veces en este hilo
HILO:4 Suma= 34

E1= 34 se sumara: 1 veces en este hilo
HILO:5 Suma= 34

E1= 34 se sumara: 1 veces en este hilo
HILO:6 Suma= 34

E1= 34 se sumara: 1 veces en este hilo
HILO:7 Suma= 34
```

```
E1= 34 se sumara: 1 veces en este hilo
HILO:8 Suma= 34

E1= 34 se sumara: 1 veces en este hilo
HILO:9 Suma= 34

E1= 34 se sumara: 3 veces en este hilo
HILO:10 Suma= 102

La suma total de los 10 hilos es 408ayonx12@ubuntu:~/Desktop/Sistemas$
```

Ejecución con datos  $12 \times 34 \times 17^*$

```
ayonx12@ubuntu: ~/Desktop/Siste
File Edit View Search Terminal Help
Cont ContadorHilos.c Sumas SumasHilos.c term
ayonx12@ubuntu:~/Desktop/Sistemas$ ./Sumas 12*34*17*
E1= 12 E2: 34 N: 17
E1= 12 se sumara: 2 veces en este hilo
HILO:1 Suma= 24

E1= 12 se sumara: 2 veces en este hilo
HILO:2 Suma= 24

E1= 12 se sumara: 2 veces en este hilo
HILO:3 Suma= 24

E1= 12 se sumara: 2 veces en este hilo
HILO:4 Suma= 24

E1= 12 se sumara: 2 veces en este hilo
HILO:5 Suma= 24

E1= 12 se sumara: 2 veces en este hilo
HILO:6 Suma= 24

E1= 12 se sumara: 2 veces en este hilo
HILO:7 Suma= 24
```

```
E1= 12 se sumara: 2 veces en este hilo
HILO:8 Suma= 24

E1= 12 se sumara: 2 veces en este hilo
HILO:9 Suma= 24

E1= 12 se sumara: 2 veces en este hilo
HILO:10 Suma= 24

E1= 12 se sumara: 2 veces en este hilo
HILO:11 Suma= 24

E1= 12 se sumara: 2 veces en este hilo
HILO:12 Suma= 24

E1= 12 se sumara: 2 veces en este hilo
HILO:13 Suma= 24

E1= 12 se sumara: 2 veces en este hilo
HILO:14 Suma= 24
```

```
E1= 12 se sumara: 2 veces en este hilo
HILO:15 Suma= 24

E1= 12 se sumara: 2 veces en este hilo
HILO:16 Suma= 24

E1= 12 se sumara: 2 veces en este hilo
HILO:17 Suma= 24

La suma total de los 17 hilos es 408ayonx12@ubuntu:~/Desktop/Sistemas$
```

- No importa cuantos hilos metamos si los factores de la multiplicación dan el mismo resultado, el trabajo de los hilos realizara correctamente la suma.



## Ejercicio 3.

### *!PROFESORA IMPORTANTE!*

*//Las unidades de incremento y decremento no supimos si tenian que ser ya declaradas o se tenian que mandar desde la ejecución o desde el main.*

*//Las unidades de decremento no sabemos si las mandamos igual pueden ser positivas por lo cual esta el for que rectifica que siempre el segundo valor sea negativo y el primero siempre sea positivo.*

Realizar un programa con una variable entera global con un valor inicial de cero. Crear un hilo que

incremente la variable global en A unidades. Crear otro hilo que la disminuya en B unidades. Al final

el hilo principal(main) imprimirá el valor de la variable global.

### **Aspectos a tomar en cuenta:**

\* Uso de hilos funcion pthread\_create

### **Bibliotecas necesarias:**

#include <pthread.h>

### **Variables:**

Struct data:

El cual contiene un id y un valor el cual se le asignara por medio del hilo creado para aumentar o disminuir el valor de la variable global

numerG:

Variable global

decremento:

Variable de decremento.

incremento:

Variable de incremento.

statush1:

Valor tipo int devuelto por pthread\_create para verificar si es correcto la creacion del hilo.

statush2:

Valor tipo int devuelto por pthread\_create para verificar si es correcto la creacion del hilo.

## Código:

```
pg721oz@codefnies: ~/Documents/SistemasOpartivos/hilos/practica
File Edit View Search Terminal Help
GNU nano 2.9.3                                ejercicio3.c

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
void funcionHilos(int, int);
void *incrOrDecrVal(void *);

typedef struct data{
    int id; //No es necesario pero por si uno me fallaba lo meti.
    int valor;
}contenedorHilo;

int numerG=0; // VARIABLE GLOBAL

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

    pthread_t hilo1;
    pthread_t hilo2;
    int incremento, decremento, statush1, statush2;

    if(argc<2){
        printf("No ingresaste valores para aumentar o disminuir en cada hilo\n");
        printf("Valores por defecto +20 y -10 \n");
        incremento = 20;
        decremento = -10;
    }
    if (argc>=2){
        incremento = atoi(argv[1]);
        decremento = atoi(argv[2]);
        if (incremento < 0){
            incremento*=-1;
        }
        if (decremento > 0)
            decremento*=-1;
    }

    contenedorHilo p1;
    p1.id = 0;
    p1.valor = incremento;
    statush1= pthread_create(&hilo1, NULL, incrOrDecrVal, (void *)&p1);

    contenedorHilo p2;
    p2.id = 1;
    p2.valor = decremento;

    statush2=pthread_create(&hilo2, NULL, incrOrDecrVal, (void *)&p2);
    if(statush1 || statush2 != 0){
        return -1;
    }
    pthread_join(hilo1, NULL);
    pthread_join(hilo2, NULL);
    printf("Valor de la variable global: %d", numerG);
    return 0;
}

void *incrOrDecrVal(void *args){
    contenedorHilo *p = (contenedorHilo *) args;
    numerG += p->valor;
    usleep(10000);
}
```

*/\*Ya que no sabiamos bien que queria en el 3 por defecto deja un numero de aumento de 20 y uno de decremento de 10\*/*

*/\*Estos dos if hacen que siempre el primer valor sea positivo y el segundo negativo \*/*

## Salida por defecto es decir al no mandar valores:

```
pg721oz@codefnies: ~/Documents/SistemasOpartivos/hilos/practica
File Edit View Search Terminal Help
pg721oz@codefnies:~/Documents/SistemasOpartivos/hilos/practica$ gcc ejercicio3.c -o ejercicio3 -lpthread
pg721oz@codefnies:~/Documents/SistemasOpartivos/hilos/practica$ nano ejercicio3.c
pg721oz@codefnies:~/Documents/SistemasOpartivos/hilos/practica$ ./ejercicio3
No ingresaste valores para aumentar o disminuir en cada hilo
Valores por defecto +20 y -10
Valor de la variable global: 10pg721oz@codefnies:~/Documents/SistemasOpartivos/hilos/practica$
```

Como se aprecia se queda por defecto un 20 y un -10 y el valor de la variable incrementa en 20, valiendo 20 y despues decrementa en 10 valiendo entonces +10

## Salida dando valores

```
Valor de la variable global: 10pg721oz@codefnies:~/Documents/SistemasOpartivos/hilos/practica$ ./ejercicio3 34 7
Valor de la variable global: 27pg721oz@codefnies:~/Documents/SistemasOpartivos/hilos/practica$ ./ejercicio3 2 7
Valor de la variable global: -5pg721oz@codefnies:~/Documents/SistemasOpartivos/hilos/practica$ ./ejercicio3 3 3
Valor de la variable global: 0pg721oz@codefnies:~/Documents/SistemasOpartivos/hilos/practica$
```

Como se muestra siempre el primer valor lo toma como positivo y el segundo valor negativo.

## Ejercicio 4

NOTA Si el segundo argumento de la función `pthread_create` es `NULL` se utilizan los atributos predeterminados, lo cual implica ser un hilo dependiente. Si un hilo es dependiente no se liberarán sus

recursos a menos que otro hilo espere su finalización, esto mediante `pthread_join`. Se pueden crear hilos sin necesidad de esperar a que dichos hilos terminen, de ser así deben tener el atributo de

independientes (`PTHREAD_CREATE_DETACHED`).

La función `pthread_attr_setdetachstate(pthread_attr_t*attr,intdetachstate)`

permite establecer el estado de terminación de un hilo, si el segundo argumento es

`PTHREAD_CREATE_DETACHED`, el hilo se considera como independiente.

## Bibliotecas a usar:

#include <pthread.h> → Necesaria para manejar los hilos

#include <unistd.h> → Para el sleep

## Variables:

hilos[] → Contenedor para crear los 10 hilos.

NHILOS → int con valor de 10-

Codigo:

```
pg721oz@codefnies: ~/Documents/SistemasOpartivos/hilos/practica
File Edit View Search Terminal Help
GNU nano 2.9.3 ejercicio4.c

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

#define NHILOS 10

void funcion_hilos(void){
    printf("\nHilo %u \n", pthread_self());
    pthread_exit(0);
}

void main(){
    int i;
    pthread_attr_t atributos;
    pthread_t hilos[NHILOS];

    pthread_attr_init(&atributos);
    pthread_attr_setdetachstate(&atributos,PTHREAD_CREATE_DETACHED);
    // pthread_t devuelve el identificador del hilo
    for(i=0; i<NHILOS; i++){
        pthread_create(&hilos[i], &atributos, (void *) funcion_hilos,NULL);
    }

    sleep(4);
}
```

Salida del ejercicio 4

```
pg721oz@codefnies: ~/Documents/SistemasOpartivos/hilos/practica
File Edit View Search Terminal Help
pg721oz@codefnies:~/Documents/SistemasOpartivos/hilos/practica$ gcc ejercicio4.c -o ejercicio4 -lpthread
pg721oz@codefnies:~/Documents/SistemasOpartivos/hilos/practica$ ./ejercicio4

Hilo 140487388694272
Hilo 140487397086976
Hilo 140487380301568
Hilo 140487355123456
Hilo 140487371908864
Hilo 140487363516160
Hilo 140487237691136
Hilo 140487327749888
Hilo 140487336142592
Hilo 140487346730752
pg721oz@codefnies:~/Documents/SistemasOpartivos/hilos/practica$
```

Como se puede observar se muestran los 10 hilos con su respectivo identificador obtenido por `pthread_self`, el cual es necesario imprimirlo con `%lu` ya que sino tirará warnings.