



INSTITUTO POLITÉCNICO NACIONAL.
ESCUELA SUPERIOR DE CÓMPUTO.



SISTEMAS OPERATIVOS.

PRÁCTICA 5

Hilos

Integrantes del equipo:

- Chavarría Vázquez Luis Enrique.
- Juárez Espinoza Ulises.
- Machorro Vences Ricardo Alberto.
- Pastrana Torres Víctor Norberto.





Índice de contenido.

Glosario de términos.	1
Hilos.	1
Hilo de usuario.	1
Hilos de kernel.	1
Procesamiento asíncrono.	1
Contenido (Investigación)	2
Consideraciones.	2
Importancia de los hilos.	2
Tipos de hilos.	2
Hilos del usuario.	2
Hilos del kernel.	3
Diferencia entre los hilos y los procesos.	3
Problema de usar hilos.	4
Usos comunes de los hilos.	4
• Procesamiento asíncrono.	4
• Aceleración de la ejecución.	4
• Trabajo interactivo y en segundo plano.	4
Creación de hilos.	4
• Identificación de nuestro hilo.	4
• Creación de nuestro hilo.	5
Gestión de hilos.	6
Modelos de subprocesos múltiples.	6
Modelo de muchos a muchos.	6
Modelo de muchos a uno.	7
Modelo de uno a uno.	8
Códigos y ventanas de ejecución	9
Programa51.c	9
Código explicado por partes.	9
Código completo.	12
Explicación de manera global del código.	14
Ejecución:	14
Programa52.c	16
Código explicado por partes.	16



Código completo.	18
Explicación de manera global del código.	19
Ejecución:	19
Programa53.c	20
Código explicado por partes.	20
Código completo.	22
Explicación de manera global del código.	23
Ejecución:	23
Programa54.c	24
Código explicado por partes.	24
Código completo.	27
Explicación de manera global del código.	30
Ejecución:	31
Programa55.c	33
Código explicado por partes.	33
Código completo.	36
Explicación de manera global del código.	37
Ejecución:	38
Conclusiones.	39
Chavarría Vázquez Luis Enrique.	39
Juárez Espinoza Ulises.	41
Machorro Vences Ricardo Alberto.	42
Pastrana Torres Victor Norberto.	43
Bibliografía	44



Índice de figuras

Ilustración 1 Creación de código.	5
Ilustración 2 Código completo.....	5
Ilustración 3 Foto referente a lo explicado arriba de muchos a muchos	7
Ilustración 4 Foto referente a lo explicado arriba de muchos a uno.	7
Ilustración 5 Foto referente a lo explicado arriba de uno a uno	8
Ilustración 6 Inclusión de bibliotecas y definición de la estructura.	9
Ilustración 7 Generando las referencias a los almacenes de las variables.....	10
Ilustración 8 Validación e impresión de los valores de los 2 hilos.	11
Ilustración 9 Impresión de los datos del HILO principal.	11
Ilustración 10 Ejecución del programa programa52.c	15
Ilustración 11 primera parte programa52.c	16
Ilustración 12 segunda parte del código del programa.	17
Ilustración 13 tercera parte del código del programa.	18
Ilustración 14 ejecución del programa en la terminal de UBUNTU	19
Ilustración 15 primera parte del código del programa53.c	20
Ilustración 16 segunda parte del código del programa53.c	20
Ilustración 17 tercera parte del código del programa53.c	21
Ilustración 18 cuarta parte del código del programa53.c.....	22
Ilustración 19 ejecución del código del programa53.c	23
Ilustración 20 cabeceras.....	24
Ilustración 21 Hilo 1 Programa54	25
Ilustración 22 Hilo2 Programa54	26
Ilustración 23 Hilo 3 Programa 54	26
Ilustración 24 main Programa 54	27
Ilustración 25 compilación	31
Ilustración 26 Ejecución	31
Ilustración 27 ejecución	31
Ilustración 28 archivo	32
Ilustración 29 reporte generado	32
Ilustración 30 archivos de cabecera	33
Ilustración 32 hilo 1 programa 55	34



Ilustración 31 hilo 2 programa 55	34
Ilustración 33 hilo 3 programa 55	35
Ilustración 34 main programa 55	35
Ilustración 35 Ejecución	38
Ilustración 36 número primo.....	38
Ilustración 37 número par	38

Índice de tablas

Tabla 1 Diferencias entre el nivel de usuario y nivel de kernel.	3
-----------------------------------------------------------------------	---



Glosario de términos.

Hilos.

Los hilos son básicamente una tarea que puede ser ejecutada en paralelo con otra tarea; teniendo en cuenta lo que es propio de cada hilo es el contador de programa, la pila de ejecución y el estado de la CPU (incluyendo el valor de los registros).

Hilo de usuario.

En una aplicación ULT pura, todo el trabajo de gestión de hilos lo realiza la aplicación y el núcleo o kernel no es consciente de la existencia de hilos. Es posible programar una aplicación como multihilo mediante una biblioteca de hilos. La misma contiene el código para crear y destruir hilos, intercambiar mensajes y datos entre hilos, para planificar la ejecución de hilos y para salvar y restaurar el contexto de los hilos.

Hilos de kernel.

Para cada hilo que existe en el espacio de usuario, hay un hilo del kernel correspondiente. Dado que estos subprocesos son administrados por el kernel, siguen una multitarea preventiva en la que el programador puede adelantarse a un subproceso en ejecución con un subproceso de mayor prioridad que está listo para la ejecución.

Procesamiento asíncrono.

Un proceso asíncrono es un proceso o una función que ejecuta una tarea "en segundo plano" sin que el usuario tenga que esperar a que finalice la tarea.



Contenido (Investigación)

Consideraciones.

Un hilo de ejecución se considera a menudo como la unidad más pequeña de procesamiento en la que trabaja un planificador.

Un proceso puede tener varios subprocesos de ejecución que se ejecutan de forma asincrónica.

Esta ejecución asincrónica brinda la capacidad de cada hilo de manejar un trabajo o servicio en particular de forma independiente. Por lo tanto, varios subprocesos que se ejecutan en un proceso manejan sus servicios, lo que en general constituye la capacidad completa del proceso. [1]

Importancia de los hilos.

Ahora, uno se preguntaría por qué necesitamos múltiples subprocesos en un proceso. ¿Por qué no se puede utilizar un proceso con un solo hilo principal (predeterminado) en cada situación?

Supongamos que hay un proceso, que recibe entradas en tiempo real y corresponde a cada entrada que tiene para producir una determinada salida. Ahora bien, si el proceso no tiene varios subprocesos, es decir, si el proceso no incluye varios subprocesos, todo el procesamiento del proceso se vuelve sincrónico. Esto significa que el proceso toma una entrada, la procesa y produce una salida.

La limitación en el diseño anterior es que el proceso no puede aceptar una entrada hasta que haya terminado de procesar la anterior y, en caso de que el procesamiento de una entrada tarde más de lo esperado, la aceptación de más entradas queda en espera.

Para considerar el impacto de la limitación anterior, si asignamos el ejemplo genérico anterior con un proceso de servidor de socket que puede aceptar la conexión de entrada, procese y proporcione al cliente de socket la salida. Ahora, si al procesar cualquier entrada, si el proceso del servidor toma más tiempo del esperado y mientras tanto, otra entrada (solicitud de conexión) llega al servidor de socket, entonces el proceso del servidor no podrá aceptar la nueva conexión de entrada ya que ya está bloqueada, procesando la antigua conexión de entrada. Esto puede dar lugar a un tiempo de espera de conexión en el cliente de socket que no se desea en absoluto. [2]

Esto muestra que el modelo de ejecución sincrónico no se puede aplicar en todas partes y, por lo tanto, se sintió el requisito del modelo de ejecución asincrónico que se implementa mediante el uso de subprocesos.

Tipos de hilos.

Hilos del usuario.

Estos subprocesos no son conocidos por el kernel y, por lo tanto, el kernel no está involucrado en su procesamiento. Estos subprocesos siguen la multitarea cooperativa en la que un subproceso libera la CPU por su propio deseo, es decir, el planificador no puede apropiarse del subproceso. [3]



Hilos del kernel.

Para cada hilo que existe en el espacio de usuario, hay un hilo del kernel correspondiente. Dado que estos subprocesos son administrados por el kernel, siguen una multitarea preventiva en la que el programador puede adelantarse a un subproceso en ejecución con un subproceso de mayor prioridad que está listo para la ejecución. [3]

En la siguiente tabla (tabla 1) podemos ver precisamente la comparativa de todo lo que hemos mencionado los puntos anteriores de manera un poco más desmenuzada y por tanto más simplificada.

Tabla 1 Diferencias entre el nivel de usuario y nivel de kernel.

#	Hilos de nivel de usuario	A nivel de kernel
1	Los subprocesos a nivel de usuario son más rápidos de crear y administrar.	Los hilos a nivel de kernel son más lentos de crear y administrar.
2	La implementación se realiza mediante una biblioteca de subprocesos a nivel de usuario.	El sistema operativo admite la creación de hilos del kernel.
3	El hilo a nivel de usuario es genérico y puede ejecutarse en cualquier sistema operativo.	El hilo de nivel de kernel es específico del sistema operativo.
4	Las aplicaciones multiproceso no pueden aprovechar el multiprocesamiento.	Las propias rutinas del kernel pueden ser multiproceso.

Diferencia entre los hilos y los procesos

Mostraremos algunas de las principales diferencias entre el hilo y los procesos: [4]

- Los procesos no comparten su espacio de direcciones, mientras que los subprocesos que se ejecutan bajo el mismo proceso comparten el espacio de direcciones.
- Desde el punto anterior, está claro que los procesos se ejecutan de forma independiente entre sí y la sincronización entre procesos la realiza el kernel solo, mientras que, por otro lado, la sincronización de subprocesos debe ser atendida por el proceso bajo el cual se ejecutan los subprocesos.
- El cambio de contexto entre hilos es rápido en comparación con el cambio de contexto entre procesos.



- La interacción entre dos procesos se logra solo a través de la comunicación estándar entre procesos, mientras que los subprocesos que se ejecutan bajo el mismo proceso pueden comunicarse fácilmente ya que comparten la mayoría de los recursos como la memoria, el segmento de texto, etc.

Problema de usar hilos.

Muchos sistemas operativos no implementan subprocesos como procesos, sino que los ven como parte del proceso principal. Considerando otro posible problema podría surgir son los problemas de concurrencia. Dado que los subprocesos comparten todos los segmentos (excepto el segmento de la pila) y el programador puede adelantarlos en cualquier etapa, cualquier variable global o estructura de datos que pueda dejarse en un estado inconsistente al adelantar un subproceso podría causar problemas graves cuando la siguiente prioridad alta thread ejecuta la misma función y usa las mismas variables o estructuras de datos. [5]

Usos comunes de los hilos.

Acorde con lo encontrado en [6] podemos ver los diversos usos que tienen los hilos.

- **Procesamiento asíncrono.**

Un ejemplo es como los software de procesamiento de texto guardan archivos temporales cuando se está trabajando en dicho programa. Se crea un hilo que tiene como función guardar una copia de respaldo mientras se continúa con la operación de escritura por el usuario sin interferir en la misma.

- **Aceleración de la ejecución.**

Se ejecutan lotes de código de forma totalmente simultánea.

- **Trabajo interactivo y en segundo plano.**

En un programa de hoja de cálculo un hilo puede estar visualizando los menús y leer la entrada del usuario mientras que otro hilo ejecuta las órdenes y actualiza la hoja de cálculo.

- **Estructuración modular de los programas.**

Puede ser un mecanismo eficiente para un programa que ejecuta una gran variedad de actividades, teniendo las mismas bien separadas mediante a hilos que realizan cada una de ellas.

Creación de hilos.

- **Identificación de nuestro hilo.**

Así como un proceso se identifica a través de un ID de proceso, un hilo se identifica mediante un ID de hilo. Pero curiosamente, la similitud entre los dos termina aquí. [2]

- 1) Un ID de proceso es único en todo el sistema, mientras que un ID de subproceso es único solo en el contexto de un único proceso.
- 2) Un ID de proceso es un valor entero, pero el ID de subproceso no es necesariamente un valor entero. Bien podría ser una estructura.



3) Un ID de proceso se puede imprimir muy fácilmente, mientras que un ID de hilo no es fácil de imprimir.

- Creación de nuestro hilo.

Basado en lo encontrado en [7], normalmente, cuando un programa se inicia y se convierte en un proceso, comienza con un hilo predeterminado. Entonces podemos decir que cada proceso tiene al menos un hilo de control. Un proceso puede crear subprocesos adicionales utilizando la siguiente función mostrada en la imagen llamada ilustración 1:

```
#include <pthread.h>
int pthread_create(pthread_t *restrict tidp, const pthread_attr_t *restrict attr, void *(*start_rtn)(void), void *restrict arg)
```

Ilustración 1 Creación de código.

El primer argumento es una dirección de tipo pthread_t. Una vez que la función se llama con éxito, la variable cuya dirección se pasa como primer argumento contendrá el ID del hilo del hilo recién creado. El segundo argumento puede contener ciertos atributos que queremos que contenga el nuevo hilo. Podría ser una prioridad, etc. El tercer argumento es un puntero de función. Esto es algo a tener en cuenta que cada hilo comienza con una función y que la dirección de las funciones se pasa aquí como tercer argumento para que el kernel sepa desde qué función iniciar el hilo. Como la función (cuya dirección se pasa en el tercer argumento anterior) puede aceptar algunos argumentos también, podemos pasar estos argumentos en forma de puntero a un tipo void.

Ahora procedemos a aplicar lo visto en el siguiente programa que aparece en la ilustración número 2.

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

pthread_t tid[2];

void* doSomething(void *arg){
    unsigned long i = 0;
    pthread_t id = pthread_self();

    if(pthread_equal(id,tid[0])){
        printf("\n Primer hilo procesado\n");
    }else{
        printf("\n Segundo hilo procesado\n");
    }

    for(i=0; i<(0xFFFFFFFF);i++){
        return NULL;
    }
}

int main(void){
    int i = 0;
    int err;
    while(i < 2){
        err = pthread_create(&tid[i], NULL, &doSomething, NULL);
        if (err != 0)
            printf("\nNo podemos leer el hilo :[%s]", strerror(err));
        else
            printf("\n Hilo creado de manera satisfactoria.\n");

        i++;
    }
    sleep(5);
    return 0;
}
```

Ilustración 2 Código completo.



De manera resumida, lo que el programa mostrado en la parte superior hace es lo siguiente.

- 1) Utiliza la función `pthread_create ()` para crear dos hilos.
- 2) La función de inicio para ambos hilos se mantiene igual.
- 3) Dentro de la función `'doSomething ()'`, el hilo usa las funciones `pthread_self ()` y `pthread_equal ()` para identificar si el hilo en ejecución es el primero o el segundo creado.
- 4) Además, dentro de la misma función `'doSomething ()'` se ejecuta un bucle `for` para simular un trabajo que consume mucho tiempo.

Ahora bien, ejecutan el código tenemos lo siguiente.

```
./threads  
  
Hilo creado de manera satisfactoria.  
Primer hilo procesado.  
Hilo creado de manera satisfactoria.  
Segundo hilo procesado.
```

Gestión de hilos.

Modelos de subprocesos múltiples.

Algunos sistemas operativos proporcionan un subproceso de nivel de usuario combinado y un servicio de subproceso de nivel de kernel. Solaris es un buen ejemplo de este enfoque combinado. En un sistema combinado, varios subprocesos dentro de la misma aplicación pueden ejecutarse en paralelo en varios procesadores y una llamada al sistema de bloqueo no necesita bloquear todo el proceso. Los modelos de subprocesos múltiples son de tres tipos [8]:

- Relación de muchos a muchos.
- Relación de muchos a uno.
- Relación uno a uno.

Modelo de muchos a muchos.

El modelo de muchos a muchos multiplexa cualquier número de subprocesos de usuario en un número igual o menor de subprocesos del núcleo [8].

El siguiente diagrama muestra el modelo de subprocesos de muchos a muchos donde 6 subprocesos a nivel de usuario se multiplexan con 6 subprocesos a nivel de kernel. En este modelo, los desarrolladores pueden crear tantos subprocesos de usuario como sea necesario y los subprocesos de Kernel correspondientes pueden ejecutarse en paralelo en una máquina multiprocesador. Este modelo proporciona la mejor precisión en la concurrencia y cuando un hilo realiza una llamada al sistema de bloqueo, el kernel puede programar otro hilo para su ejecución. En la ilustración 3 podemos ver este ejemplificado a detalle

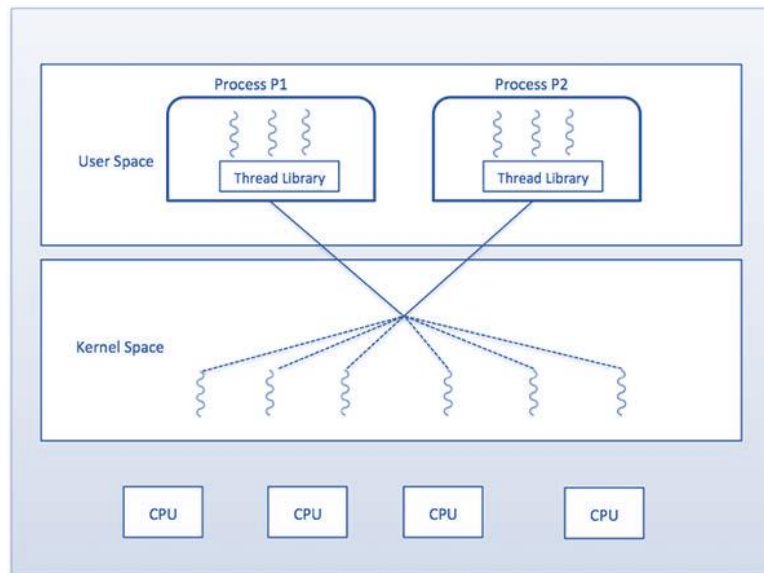


Ilustración 3 Foto referente a lo explicado arriba de muchos a muchos

Modelo de muchos a uno.

El modelo de varios a uno asigna muchos subprocesos de nivel de usuario a un subproceso de nivel de Kernel. La gestión de subprocesos se realiza en el espacio del usuario mediante la biblioteca de subprocesos. Cuando el hilo realiza una llamada al sistema de bloqueo, todo el proceso se bloqueará. Solo un subproceso puede acceder al Kernel a la vez, por lo que varios subprocesos no pueden ejecutarse en paralelo en multiprocesadores. [8]

Si las bibliotecas de subprocesos a nivel de usuario se implementan en el sistema operativo de tal manera que el sistema no las admite, entonces los subprocesos del Kernel utilizan los modos de relación de muchos a uno. La siguiente foto (ilustración 4) vemos el ejemplo de lo mencionado.

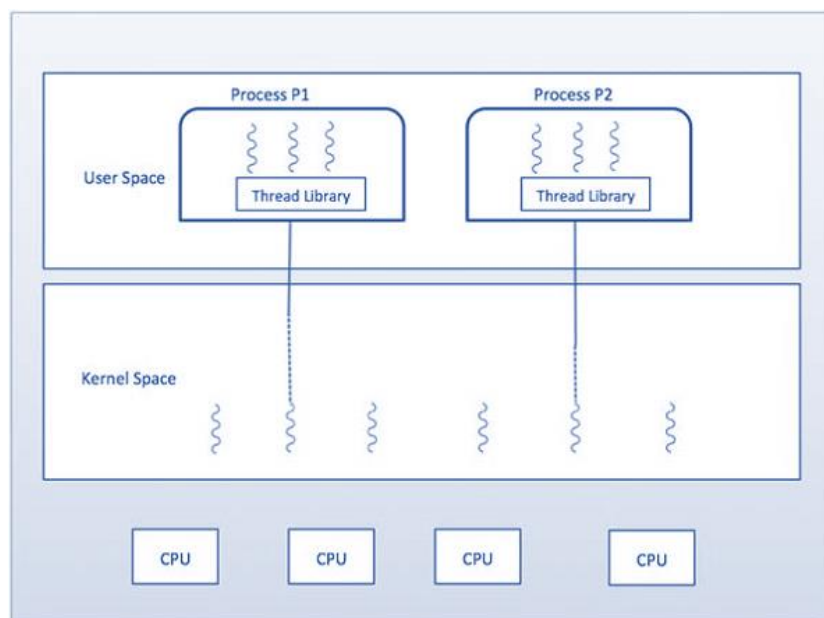


Ilustración 4 Foto referente a lo explicado arriba de muchos a uno.



Modelo de uno a uno.

Existe una relación de uno a uno entre el subproceso a nivel de usuario y el subproceso a nivel de kernel. Este modelo proporciona más simultaneidad que el modelo de varios a uno. También permite que se ejecute otro hilo cuando un hilo realiza una llamada al sistema de bloqueo. Admite múltiples subprocesos para ejecutarse en paralelo en microprocesadores. [8]

La desventaja de este modelo es que la creación de un hilo de usuario requiere el hilo de Kernel correspondiente. OS / 2, Windows NT y Windows 2000 utilizan un modelo de relación uno a uno. En la siguiente foto (ilustración 5) vemos el ejemplo de lo que hemos mencionado.

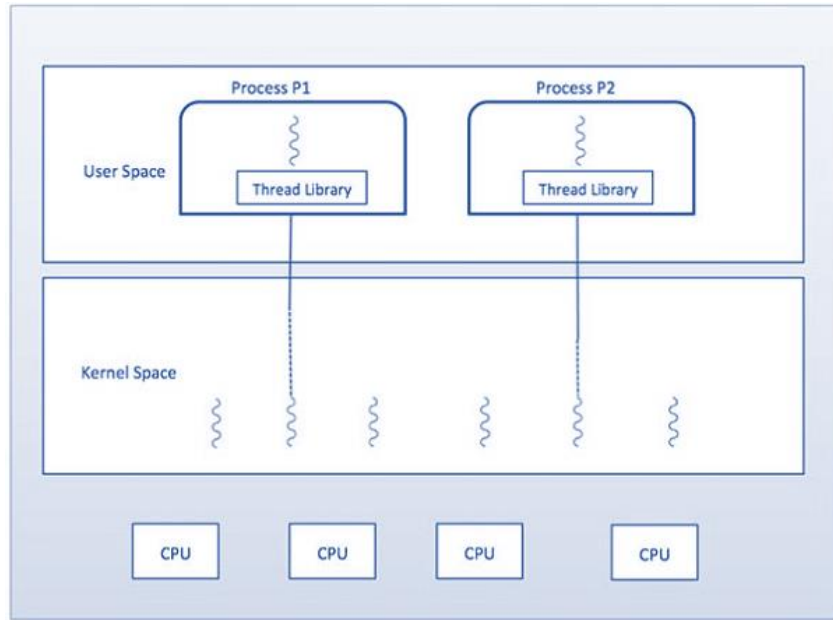


Ilustración 5 Foto referente a lo explicado arriba de uno a uno



Códigos y ventanas de ejecución

Programa51.c

Realizar un programa que inicie un hilo principal que a su vez crea dos hilos. El hilo principal espera hasta que ambos hilos terminen y después finaliza. Los hilos sólo deben de mostrar algún mensaje en pantalla y terminar.

Código explicado por partes.

En esta primera parte del código podemos ver como se hicieron todas las importaciones de las librerías necesarias para la ejecución del programa, dentro de las cuales destacan una que se emplea para el uso de precisamente la implementación de los hilos en nuestro código, también tenemos otro para el manejo de cadenas de caracteres y algunas del sistema. Cabe destacar que también hemos hecho una definición de GNU_SOURCE el cual nos ayuda a acceder a diversas funciones que son omitidas en POSIX para el manejo precisamente de los hilos y nos permite el acceso algunas funcionalidades de bajo nivel dentro de nuestro sistema operativo, esto lo podemos ver en la ilustración 6 que aparece debajo de este párrafo.

```
#define _GNU_SOURCE

/*USAMOS GNU SOURCE PARA acceder a diversas funciones
que son omitidas en POSIX, del mismo modo si queremos
acceder a funciones de bajo nivel que no son portables
a otro sistema, si buscamos portabilidad
usariamos _POSIX_C_SOURCE=200809L, pero en este caso no
es necesario.*/

#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>

#include <stdlib.h>
#include <pthread.h>
#include <sys/syscall.h>
#include <string.h>

typedef struct{
    int contador;
    /*Debemos acceder a la parte de la pid
    el identificador de nuestro hilo y a
    los pthreads*/
    pid_t identificador_pid;
    pid_t identificador_tid;
    pthread_t ptid;
} datos;
```

Ilustración 6 Inclusión de bibliotecas y definición de la estructura.

Ahora bien, también la primera imagen se muestra como se hizo la definición de nuestra estructura en la cual vamos a ingresar los datos básicos como un contador, y los identificadores de cada uno de



los hilos que en mostraremos esto con la finalidad de poder recordar y visualizar en consola que hilo corresponde acorde con su identificador.

También en ese de mencionar qué necesitamos precisamente apuntar a dichos identificadores ,para lo cual hemos implementado el siguiente código mostrado en la ilustración 7 que aparece abajo.

```
void *foo(void *args){  
    datos *referencia = (datos *) args;  
  
    referencia->ptid = pthread_self();  
    referencia->identificador_pid = getpid();  
    referencia->identificador_tid = syscall(SYS_gettid);  
  
    return(referencia);  
}
```

Ilustración 7 Generando las referencias a los almacenes de las variables.

Ahora ya dentro de nuestra función para poder mostrar los hilos, necesitamos primero que nada ingresar a la memoria dinámica y posteriormente por medio de un ciclo iremos realizando un conteo y las iteraciones para poder acceder a número de hilos que nosotros hemos definido en un comienzo, ya que este programa nos ayuda a poder determinar la cantidad de ideas que queremos que se muestra en pantalla y en nuestro caso se trata solamente de dos hilos derivados.

Cabe destacar que en esta sección básicamente se hicieron todas la validaciones como por ejemplo en caso de que no tengamos una definición de los creada lo mostro mensaje de error, pero también como lo es el caso al detener una definición de la cantidad de hilos que queremos entonces se limitará a mostrarnos simplemente el mensaje que nosotros queremos con información que nosotros le estamos pidiendo.

Al estar trabajando directamente lenguaje de programación C, en la impresión directamente se pusieron los símbolos de carácter guión con la finalidad de poder mantener todo la consola un poco más organizado y no confundir los datos al momento del impresión, desta manera tenerlos bien identificados en todo momento.

Queremos recalcar que hemos ingresado estos datos y los hemos impreso en la terminal del sistema operativo por el simple motivo de que es imprescindible que tengamos bien identificados a cada uno de los hilos y desde luego también tenemos datos del padre para poder verificar que en efecto estos serán derivados de un hilo principal, siendo todo que acabamos de mencionar visible en la ilustración 8 de la parte inferior.



```
void spawnThreads(unsigned int numeroHilosDefinidos){
    int referente;
    pthread_t *identificadores_tids = malloc(sizeof(pthread_t) * numeroHilosDefinidos);

    int contador;

    for (contador = 0; contador < numeroHilosDefinidos; contador++){
        datos *referenciaExterna = malloc(sizeof(datos) * numeroHilosDefinidos);
        memset(referenciaExterna, '\0', sizeof(*referenciaExterna));

        referenciaExterna->contador = contador;

        referente = pthread_create(&identificadores_tids[contador], NULL, foo, (void *) referenciaExterna);

        if ( referente != 0)
            perror("Error!!!");
    }

    for (int contador = 0; contador < numeroHilosDefinidos; ++contador){
        datos *estadoActual;

        referente = pthread_join(identificadores_tids[contador], (void *) &estadoActual);

        if ( referente != 0)
            perror("Error!!");
        else{
            printf("-----HILO NÚMERO [[%d]]-----\n"
                "pthreadId del proceso referido a [%lu]\n"
                "identificador [pid] == [%d]\n"
                "identificador [tid] == [%d]\n",
                estadoActual->contador, estadoActual->ptid, estadoActual->identificador_pid, estadoActual->identificador_tid);

            free(estadoActual);
        }
    }

    free(identificadores_tids);
}
```

Ilustración 8 Validación e impresión de los valores de los 2 hilos.

Finalmente en nuestro main, es suficiente con que haga moned impresión del hilo principal, lo cual es bastante sencillo porque simplemente tenemos que hacer referencia a ellos en la impresión respetando la estructura que hemos nosotros definido y por último simplemente hacemos la llamada a nuestra función para hacer aparecer los hilos que nosotros queramos, es importante recalcar que debemos pasar del parámetro de la cantidad de hilos que queramos generar partiendo del primer hilo, lo bonito de este código es que nos permite y nos da la posibilidad de crear una cantidad n de hilos lo cual está bastante bien si es que queremos ver cómo es que se generan y poder entender mucho mejor los datos y la información que el programa que hemos creado nos proporciona como puede apreciarse en la ilustración 9 ubicada en la parte inferior a este párrafo.

```
int main(int argc, char *argv[]){
    printf("___HILO PRINCIPAL___\n --pthreadId == %lu \n --pid == %d \n --tid == %ld\n"
        "-----\n"
        "-----SIGUIENTES_DOS_HILOS-----\n\n",
        pthread_self(), getpid(), syscall(SYS_gettid));

    spawnThreads(2);

    return(0);
}
```

Ilustración 9 Impresión de los datos del HILO principal.



Código completo.

Es destacable que este es nuestro código completo, en formato de texto puesto dentro de nuestra práctica, pero la explicación general del código se encuentre la parte de abajo, aunque ya lo hemos desmenuzado en la sección de arriba la cual contiene el código explicado por partes.

```
#define _GNU_SOURCE

/*USAMOS GNU SOURCE PARA acceder a diversas funciones
que son omitidas en POSIX, del mismo modo si queremos
acceder a funciones de bajo nivel que no son portables
a otro sistema, si buscamos portabilidad
usaremos _POSIX_C_SOURCE=200809L, pero en este caso no
es necesario.*/

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

typedef struct{
    int contador;
    /*Debemos acceder a la parte de la pid
    el identificador de nuestro hilo y a
    los pthreads*/
    pid_t identificador_pid;
    pid_t identificador_tid;
    pthread_t ptid;
} data;

void *foo(void *args){
    data *referencia = (data *) args;

    referencia->ptid = pthread_self();
    referencia->identificador_pid = getpid();
    referencia->identificador_tid = syscall(SYS_gettid);

    return(referencia);
}

void spawnThreads(unsigned int númeroHilosDefinidos)
{
    int referente;
    pthread_t *identificadores_tids = malloc(sizeof(pthread_t) * númeroHilosDefinidos);
}
```



```
int contador;

for (contador = 0; contador < númeroHilosDefinidos; contador++)
{
    data *referenciaExterna = malloc(sizeof(data) * númeroHilosDefinidos);
    memset(referenciaExterna, '\0', sizeof(*referenciaExterna));

    referenciaExterna->contador = contador;

    referente = pthread_create(&identificadores_tids[contador], NULL, foo, (
void *) referenciaExterna);

    if ( referente != 0)
        perror("Error!!!");
}

for (int contador = 0; contador < númeroHilosDefinidos; ++contador)
{
    data *estadoActual;

    referente = pthread_join(identificadores_tids[contador], (void *) &estad
oActual);

    if ( referente != 0)
        perror("Error!!!");
    else
    {
        printf("_____\\n"
"-----HILO NÚMERO [[%d]]-----\\n"
"pthreadId del proceso referido a [%lu]\\n"
"identifcardor [pid] == [%d]\\n"
"identificador [tid] == [%d]\\n",
estadoActual->contador, estadoActual->ptid, estadoActual->ide
ntificador_pid, estadoActual->identificador_tid);

        free(estadoActual);
    }
}

free(identificadores_tids);
}

int main(int argc, char *argv[])
{
    printf("__HILO PRINCIPAL__\\n --pthreadId == %lu \\n --pid == %d \\n --
tid == %ld\\n"
"_____\\n"
```



```
"_____SIGUIENTES_DOS_HILOS_____\n\n",  
pthread_self(), getpid(), syscall(SYS_gettid));  
  
spawnThreads(2);  
  
return(0);  
}
```

Explicación de manera global del código.

Como se mencionó anteriormente en nuestro código, lo primero que se hace es la implementación de las diversas librerías y desde luego de una definición especial para poder tener acceso funciones de bajo nivel y también poder manejar algunas funcionalidades omitidas en POSIX, todo con la mera finalidad de poder trabajar mucho mejor con nuestros hilos. Es bastante interesante ver también que en no solamente se puede crear un programa estático en el cual se generan dos hilos y ya esta, sino que también es posible como fue nuestro caso crear un programa que tienen una cantidad determinada por el mismo usuario de hilos, lo cual es bastante interesante porque ya así se mejora la implementación que nosotros deseamos, podemos inclusive pedir al usuario que no se diera una cantidad por medio de la terminal y entonces le mostrará en un ciclo cierta cantidad de hilos en una determinada cantidad ocasiones mientras se lo decida con un menú y un interfaz más detallada, pero en este caso nuestra finalidad era mostrar simplemente como en el funcionamiento de un hilo principal y dos hilos secundarios, tal cual estaba pedido los requerimientos de la práctica. Nos aseguramos de que el mensaje que mostraron estos signos fuera precisamente mensajes con información de alto valor los cuales son identificadores que nos permite entender precisamente de dónde vienen y también poder tener bien visualizado cuáles son esos hilos porque supongamos que queremos hacer que este programa se pudiera utilizar para genera mucho más hilos desde luego que tendríamos que llevar un control mucho más detallado de cada uno de los valores identificadores que estos tienen dentro de nuestro sistema operativo.

Ejecución:

En la siguiente imagen podemos apreciar la ejecución de nuestro código, a nosotros como equipo nos llamó bastante la atención como es que se compilan los programas de hilos ya que es muy interesante ver que en realidad no solamente basta con hacer nuestra compilación de forma tradicional como lo haríamos en este lenguaje programación, sino que además de ello tenemos que ingresar la directiva `-lpthread` para indicarle de manera directa al sistema que estaremos trabajando precisamente con hilos los cuales son por así decirlo funciones de un tanto bajo nivel, por lo que es necesario indicarlo además de que también estamos utilizando cierta tuberías que nos permiten el manejo de los mismos por lo cual es importante que lo hagamos esta forma, lo que ocurre sino usamos dicha directiva es que simplemente nuestro programa nos va compilar o posiblemente nos dé un archivo un formato `.out` el cual tendrá que ser ejecutado con esa terminación, pero nuestra intención es tener el ejecutable directamente con el código compilado de forma tradicional como lo tenemos en un programa común y corriente en C, entonces ya entendido esto ahora sí procedemos explica que es lo que estamos viendo en la terminal.

Principalmente tenemos la impresión de nuestro primer hilo el cual evidentemente tiene una id y única, pero también tiene un pid y tid que lo hacen también único, esto es porque se trata del hilo principal pero cuando nosotros vemos los otros dos hilos que hemos definido los cuales ya se han ingresado por medio del parámetro que le pasamos a la función de la creación de hilos el cual fue dos, podemos ver que precisamente el pid coincide con el del hilo principal y a su vez los otros



identificadores que simplemente porque están ocupando por así decirlo un espacio diferente dentro del sistema operativo, también cabe destacar que cada uno de los hilos tiene un identificador referido único.

Como lo mencionamos hace unos momentos es bastante interesante porque este programa que hemos arrollado sí lo pensamos lo podemos escalar muchas más cosas o inclusive tener una gestión mucho más eficiente de una cantidad limitada, claro está limitada en términos de los recursos que nuestra computadora nos permita, esto lo vemos en la ilustración 10 de la parte de abajo.

```
luis@luis-compu:~/Desktop/Programas Luis/51$ gcc 51.c -o 51 -lpthread
luis@luis-compu:~/Desktop/Programas Luis/51$ ./51
____HILO PRINCIPAL____
--pthreadId == 139647210829632
--pid == 9754
--tid == 9754

=====SIGUIENTES_DOS_HILOS=====

-----HILO NÚMERO [[0]]-----
pthreadId del proceso referido a [139647210825472]
identifcardor [pid] == [9754]
identificador [tid] == [9755]

-----HILO NÚMERO [[1]]-----
pthreadId del proceso referido a [139647202432768]
identifcardor [pid] == [9754]
identificador [tid] == [9756]
```

Ilustración 10 Ejecución del programa programa52.c



Programa52.c

Realizar un programa con una variable entera global (fuera de main()) con un valor inicial de cero. Crear un hilo que incremente la variable global en a unidades y crear otro hilo que la disminuya en b unidades. Al final el hilo principal imprimirá el valor de la variable global.

Código explicado por partes.

En esta primera parte del código, requerimos hacer las inclusiones de algunas de las bibliotecas estándar para el manejo usual de lenguaje pero también para el uso de los hilos, también podemos destacar que se ha hecho la declaración de la variable N la cual se inicializa en cero para cumplir con el requerimiento inicial de la variable global se inicializa fuera del main, también hemos utilizado dos funciones una para el hilo uno y otra para el hilo dos, posteriormente vemos uso de ellas pero lo explicaremos con mucho más detalle en las partes posteriores de nuestro programa. También cabe destacar que la parte de las salidas de los hilos los tenemos como nulos, se puede apreciar lo mencionado en la ilustración 11 de la zona inferior.

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

//se define la variable global en 0
int N=0;

//el primer hilo hace un incremento a la variable global
void *Hilo1(void *argumentos)
{
    N=N+5;
    pthread_exit(NULL);
}

//el segundo hilo hace un decremento a la variable global
void *Hilo2(void *argumentos)
{
    N=N-2;
    pthread_exit(NULL);
}
```

Ilustración 11 primera parte programa52.c



Ahora bien en la parte principal del programa hacemos uso de la librería de hilos y en este caso establecemos la parte de las id's o identificadores, posteriormente la parte la creación basta con que pasemos los atributos necesarios los cuales son el identificador del hilo y también el nombre como declaramos nuestro hilo, finalmente basta con utilizar join hasta que el momento de finalización se concluye y por último debemos imprimir el resultado la operación de nuestros hilos y basta con simplemente utiliza la variable en donde almacenamos el resultado la variable global para imprimir, lo que hemos mencionado puede ser visto en la ilustración número 12.

```
int main ()
{
    pthread_t id_hilo1, id_hilo2;
    printf("\nCreacion del hilo 1\n");
    //se hace primero el incremento de 5
    pthread_create(&id_hilo1, NULL, Hilo1, NULL);
    sleep(2);
    printf("\nCreacion del hilo 2\n");
    //se hace el decremento de 2
    pthread_create(&id_hilo2, NULL, Hilo2, NULL);
    sleep(2);

    printf("\nHilos esperando su finalizacion... \n");
    pthread_join(id_hilo1, NULL);
    pthread_join(id_hilo2, NULL);
    printf("\nHilos finalizados\n");
    <<<<<< HEAD
    //se imprime el resultado de la operacion de los hilos
    =====
    //se imprimir el resultado de la operacion de los hilos
    >>>>>> 17c8a9f6c2f20a6a70c0f1ec34499e535d62898c
    //si todo es correcto se deberia mostrar 3
    printf("Valor de la variable global: %d \n", N);

    return 0;
}
```

Ilustración 12 segunda parte del código del programa.



Código completo.

En la siguiente imagen (ilustración 13), podemos ver una captura de todo nuestro código que ya hemos implementado.

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

//se define la variable global en 0
int N=0;

//el primer hilo hace un incremento a la variable global
void *Hilo1(void *argumentos)
{
    N=N+5;
    pthread_exit(NULL);
}

//el segundo hilo hace un decremento a la variable global
void *Hilo2(void *argumentos)
{
    N=N-2;
    pthread_exit(NULL);
}

int main ()
{
    pthread_t id_hilo1, id_hilo2;
    printf("\nCreacion del hilo 1\n");
    //se hace primero el incremento de 5
    pthread_create(&id_hilo1,NULL,Hilo1,NULL);
    sleep(2);
    printf("\nCreacion del hilo 2\n");
    //se hace el decremento de 2
    pthread_create(&id_hilo2,NULL,Hilo2,NULL);
    sleep(2);

    printf("\nHilos esperando su finalizacion ... \n");
    pthread_join(id_hilo1,NULL);
    pthread_join(id_hilo2,NULL);
    printf("\nHilos finalizados\n");
    <<<<<< HEAD
    //se imprime el resultado de la operacion de los hilos
    =====
    //se imprimir el resultado de la operacion de los hilos
    >>>>>> 17c8a9f6c2f20a6a70c0f1ec34499e535d62898c
    //si todo es correcto se debería mostrar 3
    printf("Valor de la variable global: %d \n",N);

    return 0;
}
```

Ilustración 13 tercera parte del código del programa.

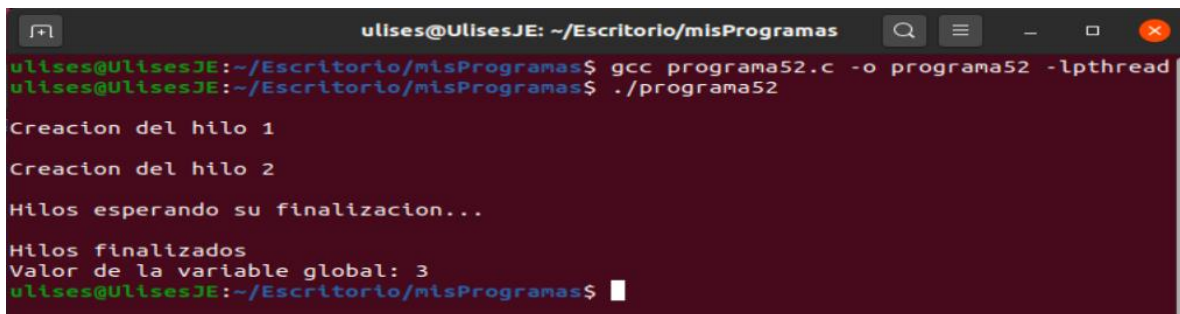


Explicación de manera global del código.

Básicamente lo que hacemos en el programa es tomar una variable global declarada en cero, y partiendo de dos hilos uno que va incrementar y otro que va disminuir poder con dicho número imprimir el resultado nuestra operación, esto es de vital importancia debido a que en muchas ocasiones que debemos hacer diversas operaciones por medio del uso de hilos, de forma totalmente paralela o para segmentar nuestro código, también podemos pensar en aplicaciones de ejecución asíncrona en donde estamos realizando múltiples operaciones por lo cual aunque es un ejemplo bastante sencillo, nos ayuda a entender mucho mejor como es que podemos realizar múltiples operaciones con múltiples hilos todo al mismo tiempo para aprovechar mucho mejor los recursos o inclusive mientras el usuario ejecuta algún otro proceso nosotros podamos hacer operaciones de fondo en nuestro programa.

Ejecución:

Ahora, ya compilando y ejecutando completamente el código podemos ver a continuación en la ilustración 14 el mensaje en donde se crea tanto el hilo uno como el hilo dos, la posterior finalización de la operación y la impresión del resultado en pantalla consiguiendo así resultado esperado.



```
ulises@UlisesJE: ~/Escritorio/misProgramas
ulises@UlisesJE:~/Escritorio/misProgramas$ gcc programa52.c -o programa52 -lpthread
ulises@UlisesJE:~/Escritorio/misProgramas$ ./programa52

Creacion del hilo 1
Creacion del hilo 2
Hilos esperando su finalizacion...
Hilos finalizados
Valor de la variable global: 3
ulises@UlisesJE:~/Escritorio/misProgramas$
```

Ilustración 14 ejecución del programa en la terminal de UBUNTU



Programa53.c

Realizar un programa cree un proceso hijo que a su vez creará tres hilos. Cada uno de los tres hilos creará dos hilos más. Cada uno de los hilos creados imprimirá en pantalla sus identificadores.

Código explicado por partes.

Lo primero que tenemos que hacer es incluir las distintas bibliotecas básicas en el uso este lenguaje y también la biblioteca para el manejo de los hilos, definiremos la cantidad de hilos y por medio de la función hilo1 comenzaremos nuestro programa, esto se ve claramente en la ilustración 15.

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

//numero de hilos del proceso hijo
#define MAX_THREADS 3

//nos servira para imprimir los identificadores de los hilos
void Hilo1(void)
{
    printf("\nHilo: %ld\n",pthread_self());
    pthread_exit(0);
}
```

Ilustración 15 primera parte del código del programa53.c

Paso esta primera parte, requerimos implementar hilo2, en donde queremos los identificadores y las inicializaciones necesarias, recordemos que requerimos crear 3 hilos más, para la que usaremos un simple bucle que se tendrá llegado al límite establecido por nosotros, lo cual es visible en la ilustración 16.

```
//dos hilos mas creados por cada uno de los 3 primeros
void Hilo2(void){
    pthread_attr_t atributos;
    pthread_t identificadores[2];
    pthread_attr_init(&atributos);
    pthread_attr_setdetachstate(&atributos,PTHREAD_CREATE_DETACHED);
    for(int k=0;k<2;k++){
        pthread_create(&identificadores[k],&atributos,(void*)Hilo1,NULL);
    }
    sleep(2);
}
```

Ilustración 16 segunda parte del código del programa53.c



Ahora bien, en la parte más interesante del código en la cual hacemos algo muy similar en la inicialización de nuestros hilos, también es importante recalcar que hemos creado algunas cuantas validaciones para cuando existan errores al generar el proceso, y algo muy interesante es que la definición que hemos hecho en un comienzo del número máximo de hilos la cual es igual a tres nos ayudara definir los límites para nuestro ciclo, lo mencionado es visible en la ilustración 17 de la parte inferior.

```
int main(){

    pid_t pid;
    pthread_attr_t atributos;
    pthread_t identificadores[MAX_THREADS];
    pthread_attr_init(&atributos);
    pthread_attr_setdetachstate(&atributos, PTHREAD_CREATE_DETACHED);

    //se crea el proceso hijo
    pid= fork();
    if(pid==-1)
    {
        perror("\nError al crear el proceso\n");
        exit(-1);
    }
    //si esta creado entonces entra al ciclo
    if(pid==0)
    {
        for(int j=0; j<MAX_THREADS; j++)
        {
            //se crean 3 hilos a partir de el proceso hijo
            pthread_create(&identificadores[j], &atributos, (void*)Hilo2, NULL);
            sleep(2);
        }
    }
    return 0;
}
```

Ilustración 17 tercera parte del código del programa53.c



Código completo.

La siguiente imagen (ilustración 18) podemos apreciar todo el desarrollo de nuestro código completo, y en la parte de abajo encontrará una explicación a detalle del mismo.

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

//numero de hilos del proceso hijo
#define MAX_THREADS 3

//nos servira para imprimir los identificadores de los hilos
void Hilo1(void)
{
    printf("\nHilo: %ld\n",pthread_self());
    pthread_exit(0);
}

//dos hilos mas creados por cada uno de los 3 primeros
void Hilo2(void){
    pthread_attr_t atributos;
    pthread_t identificadores[2];
    pthread_attr_init(&atributos);
    pthread_attr_setdetachstate(&atributos,PTHREAD_CREATE_DETACHED);
    for(int k=0;k<2;k++){
        pthread_create(&identificadores[k],&atributos,(void*)Hilo1,NULL);
    }
    sleep(2);
}

int main()
{
    pid_t pid;
    pthread_attr_t atributos;
    pthread_t identificadores[MAX_THREADS];
    pthread_attr_init(&atributos);
    pthread_attr_setdetachstate(&atributos,PTHREAD_CREATE_DETACHED);

    //se crea el proceso hijo
    pid= fork();
    if(pid==-1)
    {
        perror("\nError al crear el proceso\n");
        exit(-1);
    }
    //si esta creado entonces entra al ciclo
    if(pid==0)
    {
        for(int j=0;j<MAX_THREADS;j++)
        {
            //se crean 3 hilos a partir de el proceso hijo
            pthread_create(&identificadores[j],&atributos,(void*)Hilo2,NULL);
            sleep(2);
        }
    }

    return 0;
}
```

Ilustración 18 cuarta parte del código del programa53.c



Explicación de manera global del código.

Como lo mencionamos en el código del programa anterior resulta bastante interesante ver como por medio del uso de hilos se pueden crear un montón de cosas y desde luego optimizar programas para que hagan ejecuciones en segundo plano, lo que resulta aún más interesante y emocionante de este programa es que hemos implementado una solución multihilo, la cual sí lo pensamos al detalle podría inclusive permitirnos tener la capacidad de implementar soluciones a problemas bastante complicados en donde se requiere ejecución en segundo plano, pero que a su vez genere aún más ejecución este hilos dentro de ese mismo segundo plano, con lo cual se podría tener una gestión bastante eficiente de los recursos en un sistema y ofrecer soluciones bastante complejas en sistemas que probablemente no son los mejores o tal vez no tienen todos los recursos a la disposición y con el uso de estas técnicas se podría implementar una optimización tremenda para que dicha solución es lleguen a muchos más usuarios y desde luego se aprovechan los recursos de la mejor manera posible en todos los ámbitos.

Ejecución:

Si bien es cierto, ya hemos mencionado la importancia que tienen los hilos, pero ahora viéndolo bien en nuestra terminal (la cual es visible en la ilustración 19) vemos que en efecto hemos podido crear múltiples hilos y poder mostrar su identificador directamente la terminal de nuestro sistema operativo Ubuntu, consiguiendo con ello los resultados esperados y en adición a ello teniendo muchas más expectativas sobre los alcances que el uso de los hilos pueden tener.

```
ulises@UlisesJE:~/Escritorio/c$ cd ..
ulises@UlisesJE:~/Escritorio$ cd misProgramas
ulises@UlisesJE:~/Escritorio/misProgramas$ gcc programa53.c -o programa53 -lpthread
ulises@UlisesJE:~/Escritorio/misProgramas$ ./programa53
ulises@UlisesJE:~/Escritorio/misProgramas$
Hilo: 139962917844736

Hilo: 139962994607872

Hilo: 139962917844736

Hilo: 139963003000576

Hilo: 139962917844736

Hilo: 139962994607872
ulises@UlisesJE:~/Escritorio/misProgramas$
```

Ilustración 19 ejecución del código del programa53.c



Programa54.c

Realizar un programa que cree tres hilos. El primer hilo se encargará de contabilizar las ocurrencias de una cadena dentro de un archivo específico y devolver el resultado al programa principal; el segundo hilo copiará los archivos de su directorio actual a un subdirectorio que usted elija devolviendo al programa principal el número de archivos copiados; el tercer hilo generará un archivo donde se reportarán los resultados devueltos por los otros dos hilos.

Código explicado por partes.

Para comenzar tenemos las bibliotecas que utilizamos para este programa, están las de entrada y salida de datos, la que nos permite el uso de hilos, la que permite realizar acciones a strings y también utilizamos algunas bibliotecas que sirven para realizar llamadas sistema. Otro elemento que observamos son variables globales, en este caso son dos apuntadores y dos variables que fungirán como contadores, estas variables las usamos para la comunicación de los hilos con el proceso padre, esto puede ser visto en la ilustración 20.

```
1  #include <pthread.h>
2  #include <stdlib.h>
3  #include <stdio.h>
4  #include <unistd.h>
5  #include <sys/types.h>
6  #include <sys/stat.h>
7  #include <fcntl.h>
8  #include <string.h>
9
10 int *p;
11 int *q;
12 int konta=0;
13 int cont=0;
```

Ilustración 20 cabeceras

En el código del hilo 1 tenemos que buscar las ocurrencias de una palabra en un archivo dado, para eso utilizamos una variable de tipo entero para abrir el archivo, además de las funciones fopen, fgets, fclose y getchar. Nos auxiliamos de variable temporales que se encargaran de buscar si la palabra ingresada por el usuario se encuentra en el archivo y además cuenta cuantas veces esta en el archivo. Al final se retorna un puntero global al cual se le direccionó la variable contador con el número de incidencias de la palabra en el archivo, lo cual se ejemplifica en la ilustración 21.



```
1 void *Codigo_Hilo1(void *arg)//Hilo 1 que busca una cadena en un archivo dado
2 {
3
4 FILE *Fd;
5
6 char palabra[30],texto[80];
7 char fitxizen[]="//home//victor//Documentos//Pràcticas//Prueba.txt";
8
9 int i,tmp1,tmp2;
10 p = &konta;
11
12 printf("\t-----Hola!!!, soy el hilo 1-----\n");
13 printf("Ingrese la palabra a buscar en el fichero: \n");
14 fgets(palabra,30,stdin);
15 //home/victor/Documentos/Pràcticas/Prueba.txt
16
17 Fd=fopen(fitxizen, "r");
18
19 if (Fd==NULL)
20     printf("Error abriendo el fichero");
21
22 while (feof(Fd)==0)
23 {
24     fgets(texto,80,Fd);
25
26     for(i=0;i<strlen(texto);i++)
27     {
28         if (palabra[0]==texto[i])
29         {
30             tmp1=0;
31
32             tmp2=i;
33
34             while ((palabra[tmp1]==texto[tmp2])&&(tmp2<strlen(texto))&&(tmp1!=strlen(palabra)))
35             {
36                 tmp1++;
37                 tmp2++;
38
39                 if (tmp1==strlen(palabra))
40                     konta++;
41             }
42         }
43     }
44 }
45
46 printf("La palabra se repite en el texto %d veces\n",konta);
47 printf("Pulse Enter para continuar...\n");
48 getchar();
49
50 fclose(Fd);
51
52 return(p);
53
54 pthread_exit(NULL);
55 }
```

Ilustración 21 Hilo 1 Programa54

En el código del hilo 2 nos encargamos de copiar archivos de un directorio a otro, para esto utilizamos la llamada a sistema cp, y le damos como parámetros la dirección actual de archivo a copiar y después la dirección destino. El hilo retornara el número de archivos copiados, para esto al igual que en el hilo 1 devolvemos un puntero que esta direccionado a la variable contador del hilo 2, cabe destacar que lo mencionado es mostrado en la ilustración 22.



```
1 void *Codigo_Hilo2(void *arg)//Hilo 2 que copia archivos de un directorio a otro
2 {
3     int i;
4     q = &cont;
5
6     printf("\n\t-----Hola!!!, soy el hilo 2-----\n");
7     printf("Copiando archivos del directorio...");
8
9     system("cp //home//victor//Documentos//Pràcticas//filosofos.c //home//victor//Documentos");cont
10 ++;
11     system("cp //home//victor//Documentos//Pràcticas//filosofos //home//victor//Documentos");cont
12 ++;
13     system("cp //home//victor//Documentos//Pràcticas//Programa54.c //home//victor//Documentos"
14 );cont++;
15     system("cp //home//victor//Documentos//Pràcticas//Programa54 //home//victor//Documentos
16 ");cont++;
17     system("
18 cp //home//victor//Documentos//Pràcticas//Programa55.c //home//victor//Documentos");cont++;
19     system("
20 cp //home//victor//Documentos//Pràcticas//Programa55 //home//victor//Documentos");cont++;
21
22     printf("\nArchivos copiados\n\n");
23
24     return(q);
25
26     pthread_exit(NULL);
27 }
```

Ilustración 22 Hilo2 Programa54

En el hilo 3 recibimos como parámetros los punteros de los hilos anteriores, estos valores los mandamos a un archivo de texto en donde se imprimirán por medio de la función fprintf y en la terminal únicamente imprimimos un mensaje diciendo que el reporte con los resultados se ha generado lo cual se ve en la ilustración 23 con el código que lo implementa.

```
1 void *Codigo_Hilo3(void *argumentos)//Hilo 3 Crea el reporte con los resultados del hilo 1 y 2
2 {
3     int *datos = (int*)argumentos;
4     FILE *Fd;
5
6     printf("\n\n\t-----Hola!!!, soy el hilo 3-----\n");
7
8     Fd=fopen("//home//victor//Documentos//Pràcticas//ResultadosPrograma54.txt", "w");
9
10    if(Fd==NULL){
11        printf("\nError en la creacion del archivo");
12    }
13    else{
14        printf("\nReporte creado exitosamente, revise el directorio\n\n");
15    }
16
17    fprintf(Fd,"\t*****Resultados del programa 54*****\n");
18    fprintf(Fd,"\nEl numero de concurrencias del hilo 1 es: %d\n",*datos);
19    fprintf(Fd,"\nEl numero de elementos copiados del hilo 2 es: %d\n",*(datos+1));
20
21
22    fclose(Fd);
23
24    pthread_exit(NULL);
25 }
```

Ilustración 23 Hilo 3 Programa 54



Para el caso de la función principal encontramos la declaración de los 3 hilos y el arreglo de argumentos donde se guardan los valores retornados por los hilos, cada hilo es creado mediante la función `pthread_create` donde le pasamos el identificador del hilo, la rutina donde se encontrará y los valores que se le enviarán a los hilos, esto es para el caso del hilo 3, si nos fijamos lo descrito se ve en términos de código en la ilustración 24.

```
1  int main()
2  {
3      pthread_t hilo1;
4      pthread_t hilo2;
5      pthread_t hilo3;
6      int argumentos[2];
7
8      pthread_create(&hilo1, NULL,Codigo_Hilo1,NULL);
9      pthread_join(hilo1,NULL);
10     printf("\t*****Hilo numero 1 terminado*****\n");
11     printf("El valor retornado por el hilo 1 fue: %d\n\n",*p);
12
13     pthread_create(&hilo2, NULL,Codigo_Hilo2,NULL);
14     pthread_join(hilo2,NULL);
15     printf("\t*****Hilo numero 2 terminado*****\n");
16
17     argumentos[0] = *p;
18     argumentos[1] = *q;
19     pthread_create(&hilo3, NULL,Codigo_Hilo3,(void*)argumentos);
20     pthread_join(hilo3,NULL);
21     printf("\t*****Hilo numero 3 terminado*****\n");
22
23     printf("\nFinalizando el hilo principal\n");
24     pthread_exit(NULL);
25 }
```

Ilustración 24 main Programa 54

Código completo.

A continuación, podemos apreciar en formato de texto todo nuestro código implementado, la explicación general de funcionamiento se encuentra en la parte de abajo del mismo, la cual está dedicada para estos menesteres.

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

```
int *p;
int *q;
int konta=0;
```




```
int cont=0;

void *Codigo_Hilo1(void *arg)//Hilo 1 que busca una cadena en un archivo dad
o
{
FILE *Fd;

    char palabra[30],texto[80];
    char fitxizen[]="//home//victor//Documentos//Pràcticas//Prueba.txt";

    int i,tmp1,tmp2;
    p = &konta;

    printf("\t-----Hola!!!, soy el hilo 1-----
\n");
    printf("Ingrese la palabra a buscar en el fichero: \n");
    fgets(palabra,30,stdin);
    //home/victor/Documentos/Pràcticas/Prueba.txt

    Fd=fopen(fitxizen, "r");

    if (Fd==NULL)
        printf("Error abriendo el fichero");

    while (feof(Fd)==0)
    {
        fgets(texto,80,Fd);

        for(i=0;i<strlen(texto);i++)
        {
            if (palabra[0]==texto[i])
            {
                tmp1=0;

                tmp2=i;

                while ((palabra[tmp1]==texto[tmp2])&&(tmp2<strlen(texto))&
&(tmp1!=strlen(palabra)))
                {
                    tmp1++;
                    tmp2++;

                    if (tmp1==strlen(palabra))
                        konta++;
                }
            }
        }
    }

    printf("La palabra se repite en el texto %d veces\n",konta);
    printf("Pulse Enter para continuar...\n");
```



```
        getchar();

        fclose(Fd);

        return(p);

    pthread_exit(NULL);
}

void *Codigo_Hilo2(void *arg)//Hilo 2 que copia archivos de un directorio a
otro
{
    int i;
    q = &cont;

    printf("\n\t-----Hola!!!, soy el hilo 2-----\n");
    printf("Copiando archivos del directorio...");

    system("cp //home//victor//Documentos//Pràcticas//filosofos.c //home//vic
tor//Documentos");cont++;
    system("cp //home//victor//Documentos//Pràcticas//filosofos //home//vi
ctor//Documentos");cont++;
    system("cp //home//victor//Documentos//Pràcticas//Programa54.c //ho
me//victor//Documentos");cont++;
    system("cp //home//victor//Documentos//Pràcticas//Programa54 //h
ome//victor//Documentos");cont++;
    system("cp //home//victor//Documentos//Pràcticas//Programa55.
c //home//victor//Documentos");cont++;
    system("cp //home//victor//Documentos//Pràcticas//Programa
55 //home//victor//Documentos");cont++;

    printf("\nArchivos copiados\n\n");

    return(q);

    pthread_exit(NULL);
}

void *Codigo_Hilo3(void *argumentos)//Hilo 3 Crea el reporte con los resulta
dos del hilo 1 y 2
{
    int *datos = (int*)argumentos;
    FILE *Fd;

    printf("\n\n\t-----Hola!!!, soy el hilo 3-----\n");

    Fd=fopen("//home//victor//Documentos//Pràcticas//ResultadosPrograma54.txt
", "w");

    if(Fd==NULL){
        printf("\nError en la creacion del archivo");
    }
}
```



```
}
else{
    printf("\nReporte creado exitosamente, revise el directorio\n\n");
}

fprintf(Fd, "\t*****Resultados del programa 54*****\n"
);
fprintf(Fd, "\nEl número de concurrencias del hilo 1 es: %d\n", *datos);
fprintf(Fd, "\nEl número de elementos copiados del hilo 2 es: %d\n", *(dato
s+1));

fclose(Fd);

pthread_exit(NULL);
}

int main()
{
    pthread_t hilo1;
    pthread_t hilo2;
    pthread_t hilo3;
    int argumentos[2];

    pthread_create(&hilo1, NULL, Codigo_Hilo1, NULL);
    pthread_join(hilo1, NULL);
    printf("\t*****Hilo número 1 terminado-*****\n");
    printf("El valor retornado por el hilo 1 fue: %d\n\n", *p);

    pthread_create(&hilo2, NULL, Codigo_Hilo2, NULL);
    pthread_join(hilo2, NULL);
    printf("\t*****Hilo número 2 terminado*****\n");

    argumentos[0] = *p;
    argumentos[1] = *q;
    pthread_create(&hilo3, NULL, Codigo_Hilo3, (void*)argumentos);
    pthread_join(hilo3, NULL);
    printf("\t*****Hilo número 3 terminado*****\n");

    printf("\nFinalizando el hilo principal\n");
    pthread_exit(NULL);
}
```

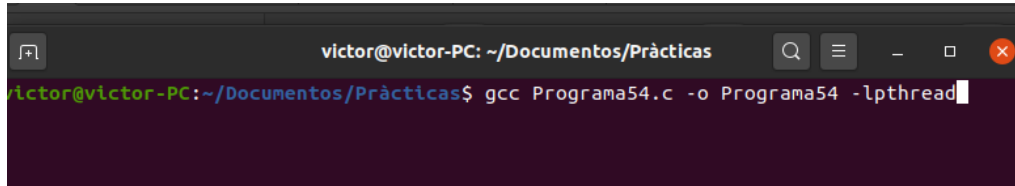
Explicación de manera global del código.

El programa 54 se trata de un proceso que crea tres hilos, cada uno de ellos se encargara de realizar una tarea en específico, el primer hilo se encarga de buscar una cadena ingresada por el usuario y buscarla en un archivo de texto dado, después retorna el número de coincidencias. El segundo hilo se encarga de copiar archivos de un directorio a otro y retornar el número de archivos copiados y finalmente el hilo 3 recibe los parámetros de los hilos anteriores y crea un archivo de texto donde manda los resultados obtenidos.



Ejecución:

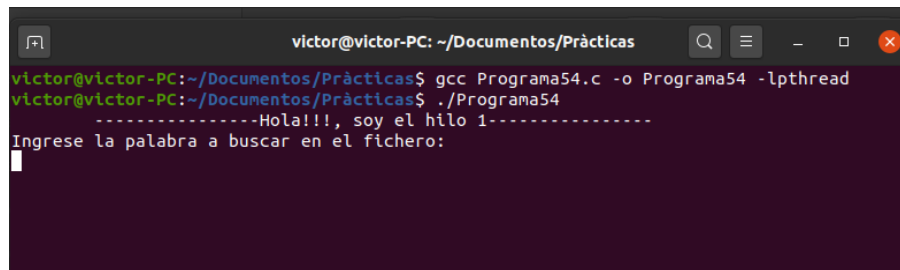
Para la ejecución utilizamos el comando gcc con el nombre del programa y además -lpthread, con esto podremos compilar archivos que hagan uso de hilos, de lo contrario no se podrá realizar la compilación, lo anteriormente descrito se aprecia en la imagen de la ilustración 25 de la terminal .



```
victor@victor-PC: ~/Documentos/Prácticas
victor@victor-PC:~/Documentos/Prácticas$ gcc Programa54.c -o Programa54 -lpthread
```

Ilustración 25 compilación

Para la ejecución del programa únicamente escribimos el comando ./Programa54 y el programa se ejecutó. Primero nos pregunta la palabra a buscar en el archivo de texto, esto se puede ver con facilidad en la ilustración 26 que aparece en la zona inferior a este párrafo.

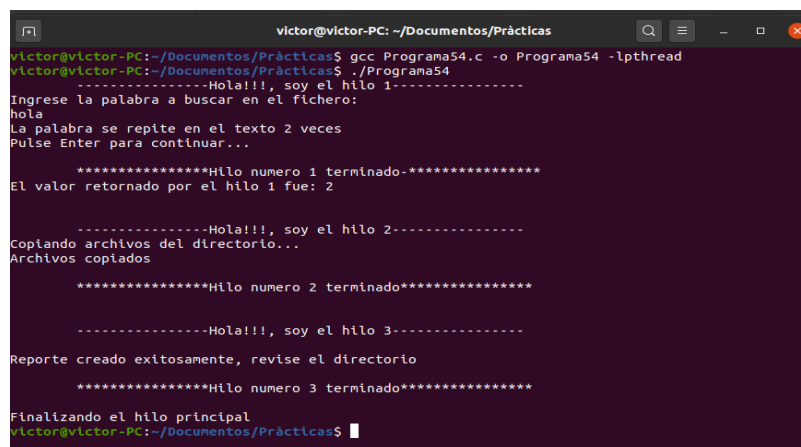


```
victor@victor-PC: ~/Documentos/Prácticas
victor@victor-PC:~/Documentos/Prácticas$ gcc Programa54.c -o Programa54 -lpthread
victor@victor-PC:~/Documentos/Prácticas$ ./Programa54
-----Hola!!!, soy el hilo 1-----
Ingrese la palabra a buscar en el fichero:

```

Ilustración 26 Ejecución

Después de ingresar la palabra realizara la búsqueda, si lo encontró enviara el número de coincidencias. Al terminar envía un mensaje al usuario diciendo que el reporte de resultados fue generado, en la zona inferior podemos ver la ilustración 27, que muestra a la perfección lo descrito anteriormente.



```
victor@victor-PC: ~/Documentos/Prácticas
victor@victor-PC:~/Documentos/Prácticas$ gcc Programa54.c -o Programa54 -lpthread
victor@victor-PC:~/Documentos/Prácticas$ ./Programa54
-----Hola!!!, soy el hilo 1-----
Ingrese la palabra a buscar en el fichero:
hola
La palabra se repite en el texto 2 veces
Pulse Enter para continuar...

*****Hilo numero 1 terminado*****
El valor retornado por el hilo 1 fue: 2

-----Hola!!!, soy el hilo 2-----
Copiando archivos del directorio...
Archivos copiados

*****Hilo numero 2 terminado*****

-----Hola!!!, soy el hilo 3-----
Reporte creado exitosamente, revise el directorio

*****Hilo numero 3 terminado*****

Finalizando el hilo principal
victor@victor-PC:~/Documentos/Prácticas$
```

Ilustración 27 ejecución



Revisamos el fichero y el archivo txt generado, lo cual se puede apreciar en la ilustración 28 y 29, donde tenemos el archivo y su posterior lectura.

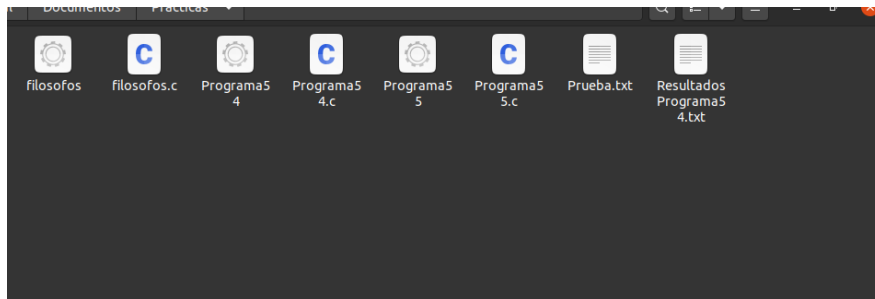


Ilustración 28 archivo

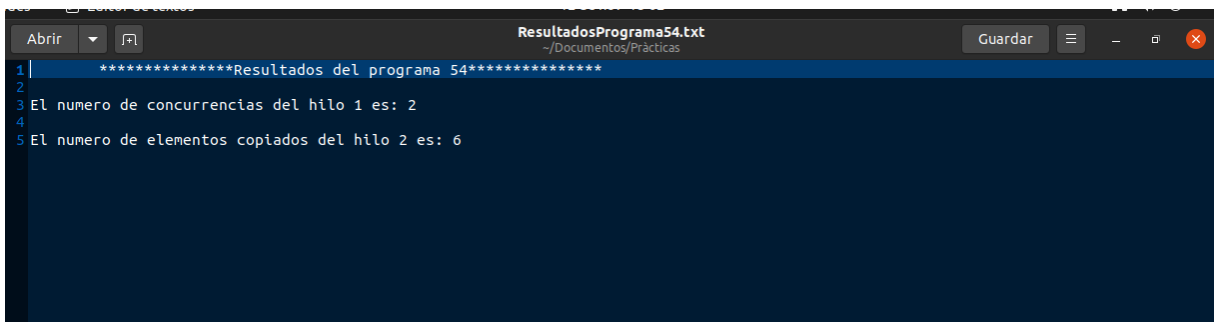


Ilustración 29 reporte generado



Programa55.c

Realizar un programa donde un hilo se encargará de decir si un número entero dado por el usuario es compuesto, si lo es, otro hilo se encargará de descomponerlo en sus números primos, si no lo es, otro hilo se encargará de decir que si es primo.

Código explicado por partes.

Comenzamos con los archivos de cabecera, estos son solo las bibliotecas de entrada y salida estándar y la biblioteca para el uso de hilo. También tenemos una variable y un apuntador global los cuales permitirán la comunicación entre los hilos y el proceso, lo descrito es fácilmente identificable en la ilustración 30 desplegada en la parte inferior.

```
1  #include<stdio.h>
2  #include<pthread.h>
3  #include<stdlib.h>
4  #include<unistd.h>
5
6  int *p;
7  int ref;
```

Ilustración 30 archivos de cabecera

En el hilo número uno nos encargamos de decir si un número ingresado por el usuario es un número compuesto, para esto nos apoyamos del operador modulo, si el módulo del número con 2 es cero, entonces podemos decir que el número es compuesto, con esta premisa devolveremos un valor al proceso padre quien decidirá a que otro hilo llamar, si al hilo 2 que descompondrá el número en sus factores primos o el otro hilo encargado de decir que es un número primo, lo cual es visible en la ilustración número 31 de la zona de abajo.



```
1 void *Hilo1(void *argumentos)//Hilo que verifica que el numero ingresado es compuesto
2 {
3
4     int factor = 2;
5     p = &ref;
6
7     int *parametros=(int*)argumentos;
8     printf("\nEstamos en el hilo 1...\n");
9     printf("Valor ingresado: %i",*parametros);
10
11     if(*parametros%factor==0){
12         printf("\nEl numero se puede descomponer");
13         ref = 0;
14     }
15     else{
16         printf("\nEl numero no se puede descomponer");
17         ref = 1;
18     }
19
20     return (p);
21     pthread_exit(NULL);
22 }
```

Ilustración 31 hilo 1 programa 55

En el hilo 2 nos encargamos de descomponer el número ingresado por el usuario en sus factores primos, para esto el hilo debe de recibir en número dado, esto lo hará el proceso padre. Para descomponer el número realizamos un ciclo que se realizara mientras el factor sea menor o igual al parámetro que recibió, además mientras el ciclo avanza realizaremos la división del parámetro entre el factor siempre y cuando el módulo de dicha operación sea cero, consideremos que esto se encuentra descrito en la imagen (ilustración 31) que se encuentra al pie de este párrafo, todo en forma de código.

```
1 void *Hilo2(void *arr){//Hilo que calcula los factores del numero
2
3     int *parametros=(int*)arr;
4     int factor;
5
6     printf("\n-----");
7     printf("\nHola!! soy el hilo 2 y tengo el dato [%d] y sus factores son\n",*
8     parametros);
9
10    for(factor =2;factor <=*parametros;factor++){
11        while(*parametros%factor==0){
12            printf("%d\t/%d\n",*parametros,factor);
13            *parametros/=factor;
14        }
15    }
16    pthread_exit(NULL);
17 }
```

Ilustración 32 hilo 2 programa 55



En la imagen (ilustración 33) , vemos que en el hilo 3 solo nos encargamos de decir si el número ingresado es primo, para esto el hilo número dos se encargó de hacer la operación modulo, sabemos que un número primo únicamente se puede dividir entre 1 y entre sí mismo, si otro número lo divide, entonces el número será par,

```
1
2 void *Hilo3(void *arr){//Hilo que muestra el mensaje de ser primo
3
4     int *parametros=(int*)arr;
5     printf("\n-----");
6     printf("\nHola!! soy el hilo 3, el numero [%d] es primo\n",*parametros);
7     pthread_exit(NULL);
8 }
```

Ilustración 33 hilo 3 programa 55

El proceso padre se encargará de tener la definición de los hilos hijos, además de un arreglo en el cual almacenará la respuesta del hilo 1, con esta respuesta decidirá que otro hilo crear, si el hilo 2 o el hilo 3, recordando que cada uno tiene funciones distintas. Esta decisión la hará con el valor retornado del hilo 1, si es 0 entonces creara al hilo 2, de lo contrario creara al hilo 3. Este proceso tiene tres funciones para esperar a quien los hilos hijos terminen su ejecución, la imagen que concuerda con lo mencionado es la número 34, en donde se puede visualizar con mejor claridad lo ya dicho.

```
1 int main ()
2 {
3     pthread_t hilo1;
4     pthread_t hilo2;
5     pthread_t hilo3;
6
7     int argumentos[1],arr[1];
8
9     printf("Ingrese el numero a descomponer: ");
10    scanf("%i",&argumentos[0]);
11
12    arr[0] = argumentos[0];
13
14    pthread_create(&hilo1,NULL,Hilo1,(void*)argumentos);
15    pthread_join(hilo1,NULL);
16    printf("\nHilo 1 terminado...");
17
18    printf("\nEl valor de referencia es: %d\n",*p);
19
20    if(*p==0){
21        pthread_create(&hilo2,NULL,Hilo2,(void*)arr);
22        pthread_join(hilo2,NULL);
23    }
24    else {
25        pthread_create(&hilo3,NULL,Hilo3,(void*)arr);
26        pthread_join(hilo3,NULL);
27    }
28
29 }
```

Ilustración 34 main programa 55



Código completo.

A continuación, podemos ver el código completo que ha sido implementado, la explicación general del código se encuentra ubicado en la parte inferior del código, junto con la parte de la ejecución.

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

int *p;
int ref;

void *Hilo1(void *argumentos)//Hilo que verifica que el número ingresado es
compuesto
{
    int factor = 2;
    p = &ref;

    int *parametros=(int*)argumentos;
    printf("\nEstamos en el hilo 1...\n");
    printf("Valor ingresado: %i",*parametros);

    if(*parametros%factor==0){
        printf("\nEl número se puede descomponer");
        ref = 0;
    }
    else{
        printf("\nEl número no se puede descomponer");
        ref = 1;
    }

    return (p);
    pthread_exit(NULL);
}

void *Hilo2(void *arr){//Hilo que calcula los factores del número

    int *parametros=(int*)arr;
    int factor;

    printf("\n-----");
    printf("\nHola!! soy el hilo 2 y tengo el dato [%d] y sus factores son\
n",*parametros);

    for(factor =2;factor <=*parametros;factor++){
        while(*parametros%factor==0){
            printf("%d\t/%d\n",*parametros,factor);
            *parametros/=factor;
        }
    }
}
```



```
        pthread_exit(NULL);
    }

    void *Hilo3(void *arr){//Hilo que muestra el mensaje de ser primo

        int *parametros=(int*)arr;
        printf("\n-----");
        printf("\nHola!! soy el hilo 3, el número [%d] es primo\n",*parametros)
    ;
        pthread_exit(NULL);
    }

    int main ()
    {
        pthread_t hilo1;
        pthread_t hilo2;
        pthread_t hilo3;

        int argumentos[1],arr[1];

        printf("Ingrese el número a descomponer: ");
        scanf("%i",&argumentos[0]);

        arr[0] = argumentos[0];

        pthread_create(&hilo1,NULL,Hilo1,(void*)argumentos);
        pthread_join(hilo1,NULL);
        printf("\nHilo 1 terminado...");

        printf("\nEl valor de referencia es: %d\n",*p);

        if(*p==0){
            pthread_create(&hilo2,NULL,Hilo2,(void*)arr);
            pthread_join(hilo2,NULL);
        }
        else {
            pthread_create(&hilo3,NULL,Hilo3,(void*)arr);
            pthread_join(hilo3,NULL);
        }
    }
}
```

Explicación de manera global del código.

En este programa se hace uso de hilos y de variables globales que nos permiten la comunicación entre el proceso padre y los hilos hijos. La idea general es recibir un número por el usuario y decir si este es un número compuesto, en el caso de serlo entonces se calcularán los factores de este, de lo contrario solo se indicará que es un número primo. Para la decisión si el número es primo o no, se usa un hilo que asigna el valor a una variable dependiendo del resultado de la operación modulo, y retorna un apuntador a esa variable, con este apuntador tendremos acceso al valor de a variable. Mediante un condicional el proceso padre decidirá que hilo hijo creará, uno de los hilos calcula los factores del número (si los tiene) y el otro indicará que es un número primo.



Ejecución:

Para la compilación del programa utilizamos el comando `gcc Programa55.c -o Programa55 -lpthread`, y la ejecución fue con `./Programa55`. El programa pide un número al usuario y si este número es compuesto lo descompone en sus factores primos, de lo contrario manda un mensaje al usuario diciendo que el número es primo, esto queda perfectamente ejemplificado en la ilustración 35.

```
victor@victor-PC: ~/Documentos/Prácticas
victor@victor-PC:~/Documentos/Prácticas$ gcc Programa55.c -o Programa55 -lpthread
victor@victor-PC:~/Documentos/Prácticas$ ./Programa55
```

Ilustración 35 Ejecución

Este es el resultado si el número no se puede descomponer en factores primos, lo cual es visible en la ilustración 36 de abajo.

```
victor@victor-PC: ~/Documentos/Prácticas
victor@victor-PC:~/Documentos/Prácticas$ gcc Programa55.c -o Programa55 -lpthread
victor@victor-PC:~/Documentos/Prácticas$ ./Programa55
Ingrese el numero a descomponer: 45

Estamos en el hilo 1...
Valor ingresado: 45
El numero no se puede descomponer
Hilo 1 terminado...
El valor de referencia es: 1

-----
Hola!! soy el hilo 3, el numero [45] es primo
victor@victor-PC:~/Documentos/Prácticas$
```

Ilustración 36 número primo

Resultado si el número es compuesto, acorde con lo mostrado en la ilustración 37.

```
victor@victor-PC: ~/Documentos/Prácticas
victor@victor-PC:~/Documentos/Prácticas$ gcc Programa55.c -o Programa55 -lpthread
victor@victor-PC:~/Documentos/Prácticas$ ./Programa55
Ingrese el numero a descomponer: 45

Estamos en el hilo 1...
Valor ingresado: 45
El numero no se puede descomponer
Hilo 1 terminado...
El valor de referencia es: 1

-----
Hola!! soy el hilo 3, el numero [45] es primo
victor@victor-PC:~/Documentos/Prácticas$ ./Programa55
Ingrese el numero a descomponer: 86

Estamos en el hilo 1...
Valor ingresado: 86
El numero se puede descomponer
Hilo 1 terminado...
El valor de referencia es: 0

-----
Hola!! soy el hilo 2 y tengo el dato [86] y sus factores son
86      |2
43      |43
victor@victor-PC:~/Documentos/Prácticas$
```

Ilustración 37 número par



Conclusiones.

Chavarría Vázquez Luis Enrique.

Esta práctica fue de suma relevancia porque conseguí entender de forma clara la importancia que tienen los hilos en el desarrollo de software, no solamente en los sistemas operativos ya que tienen un montón de aplicaciones en la industria de las cuales muchas veces nosotros ni siquiera somos conscientes y que día a día nos permiten utilizar programas tan fundamentales en nuestra cotidianidad que a veces damos por hecho su uso y ni siquiera pensamos en todo lo que hay detrás de ellos.

Uno de sus usos que mí me sorprendió bastante es precisamente la parte de procesamiento asíncrono, que en software como lo son procesadores de texto u otro software de aplicación como las herramientas para desarrollo de las cuales podemos escribir código, tienen un algoritmo que precisamente se basa en el uso de hilos para generar archivos de guardado temporal de lo que hemos estado trabajando, es super importante esto porque de hecho permite muchas más facilidades al momento de no solamente desplegar la información en pantalla, sino que también tiene respaldos de dicha información en caso de que posibles fallos o posibles errores no sólo dentro del sistema sino también del usuario, por lo que podría haber cierta capa de seguridad y de ser capaces de garantizar al usuario de que su información se mantendrá salvo.

Eso sólo por arrancar también tenemos otros usos bastante interesantes como lo es la aceleración de la ejecución de los programas, debemos preguntarnos qué es lo que podría pasar en caso de que estamos desarrollando un software que realmente ponga el límite las capacidades de algún dispositivo y entonces al no poder modificar el dispositivo de los usuarios o el dispositivo mismo, contamos con técnicas para ejecutar dicho programa, entonces los hilos pueden ser de mucha ayuda para optimizar procesos y generar distintas operaciones de manera paralela, esto es bastante interesante porque de hecho permite hacer llegar a multitud de usuarios un software con funcionalidades super complejas que aproveche de forma eficiente las capacidades de los diversos dispositivos sin importar las diferencias que existan entre sus límites a nivel hardware.

Ya entrados en esta parte de la optimización, no solamente entran los procesos en segundo plano sino que también podemos de dicho tener una estructura mucho más modular de los programas que realicemos, a lo que me refiero con esto es que podemos tener un mecanismo sumamente eficiente que ejecute teniendo las operaciones de forma bien separada y en caso de que si algún proceso llegase a fallar nosotros podemos tener un plan de gestión en el cual podamos decidir qué hilo debe morir y que hilo debe vivir y en caso de que ocurra un fallo esto es sumamente importante porque puede ahorrarle al usuario y realmente al cliente en muchísimo sufrimiento o si es que algún proceso falla, para que este proceso que ha fallado se mantenga dentro de ese mismo hilo y no genere una reacción en cadena dentro del mismo código y todo se mantenga de forma mucho más modular.

Una vez que ya mencionado esto, creo que queda bastante claro la importancia que tienen los hilos en la industria, no solamente en el sistema operativo también en muchas otras ramas de la industria y de hecho si lo pensamos la optimización del software da la pauta para que muchos usuarios puedan disfrutar de funcionalidades que posiblemente no están a su alcance por limitaciones técnicas de los dispositivos a los que estos corren o inclusive limitaciones temporales en las que los dispositivos simplemente no están listos para poder ejecutar ciertos programas pero gracias a la optimización del software podemos conseguir eso con el uso de los hilos y desde luego muchas otras técnicas más, pero podemos partir de la base, de los hilos mismos.



Quiero resaltar que uno de los aprendizajes principales que me llevé en esta práctica, es precisamente manejar los hilos en el lenguaje de programación C porque en un principio debo admitir que parecía bastante desafiante la parte de implementación, pero al final resultó ser bastante simple es cierto punto intuitivo porque todo momento el código se trató de mantener siempre de forma sumamente estructurada para poder mantener siempre el recorrido de los datos dentro de nuestro código en las soluciones a los problemas que implementamos.

Quedo bastante satisfecho con la práctica, porque como siempre lo mencionado no solamente la parte teórica ha quedado clara, sino que también he logrado expandir un poco más mi visión sobre cómo y para que podemos utilizar los hilos en la industria y creo que sería bastante interesante poder implementarlos en algún proyecto de escala mucho más grande que pudiera satisfacer necesidades de clientes en el mercado.



Juárez Espinoza Ulises.

En esta práctica, si bien logré aprender bastantes cosas sobre el uso de los hilos, aprendí en compañía de mis compañeros la importancia los hilos y de hecho en una de las discusiones que tuvimos durante la realización de las prácticas empezamos a platicar bastante sobre la importancia que esto puede tener para múltiples aplicaciones e inclusive problemas a los cuales nos hemos enfrentado de manera previa y que de hecho no supimos cómo resolver en su momento o ya que no conocíamos el concepto de los hilos y sobre todo los alcances que esto puede tener, debo confesar que en un comienzo parecía bastante complejo el tema y de hecho aunque ella había escuchado de forma teórica en qué consistía la verdad es que no me había animado aprender un poco más sobre ellos debido a que creí que había una curva de aprendizaje demasiado pronunciada en la aplicación de estos, pero menuda sorpresa me lleve al darme cuenta de que en realidad es bastante sencillo y de hecho hasta intuitivo poder trabajar con los hilos ya que el mismo lenguaje de programación nos permite hacer implementaciones de una manera muy natural muy flexible.

Durante proceso del investigación pareció bastante interesante todo lo que aprendimos, init específico aquella parte en la que vimos diversas aplicaciones para trabajo asíncrono, aunque las prácticas no muestran en esencia cómo usar estas herramientas la verdad es que me quedo con un muy buen sabor de boca ya que ahora veo con más claridad las posibilidades que los hilos ofrecen y sobre todo estamos ansiosos de poder aplicarlo a nuestro proyecto final de manera que nuestro programa puede ser bastante eficiente y sobre todo fresco una solución confiable nuestros clientes.

En un comienzo tuve algunas dificultades con la parte de la creación de hilos basados en hilos ya previamente creados, porque a decir verdad mucha de la documentación que había estado en inglés pero a pesar de eso la documentación encontrada fue bastante útil y ya una vez entrado en materia poder fácilmente analizar como implementar los hilos y se hizo más y más fácil conforme realiza un nuevo intento, por lo que a decir verdad me quedó bastante satisfecho con lo aprendido en esta práctica y desde luego estamos ansiosos de poder seguir trabajando con mucha más implementaciones y sobre todo emocionados por llevarlo a situaciones del mundo real en donde podamos satisfacer necesidades del mercado y desde luego aportar a la sociedad.

Algo similar, me pasó cuando descubrí el paradigma de programación funcional, ya que al darme cuenta de que era mucho más y siente para ciertas operaciones y cálculos tremendos quedé fascinado, pero ahora contó estas herramientas del sistema operativo que antes ignorábamos por completo en considero que se pueden hacer muchísimas cosas bastante interesantes y sobre todo tener software desarrollado ya un nivel que me atrevería mar industrial, capaz de dar solución a problemas de una escala gigantesca, es ahí donde radica mi fascinación por estos temas y esta práctica sin duda a satisfasido dicha fascinación.



Machorro Vences Ricardo Alberto.

La capacidad de poder hacer varios procesos en “paralelo” en un mismo programa es impresionante porque así se puede gestionar varias cosas y realizar múltiples acciones de manera sencilla, pero estos procesos usan mucha memoria, por lo que para esta práctica se nos dio una solución para poder hacer lo mismo, pero de manera más ligera. La solución que esta práctica nos da son los hilos, que por el hecho de usar el mismo espacio de memoria.

Esta compartición de memoria quiere decir que, si un hilo toca una variable, todos los demás hilos del mismo proceso verán el nuevo valor de la variable. Esto hace imprescindible el uso de semáforos o mutex (exclusión mutua) para evitar que dos hilos accedan a la vez a la misma estructura de datos. También hace que, si un hilo "se equivoca" y corrompe una zona de memoria, todos los demás hilos del mismo proceso vean la memoria corrompida. Un fallo en un hilo puede hacer fallar a todos los demás hilos del mismo proceso. Esto se aprendió de forma difícil en esta práctica por las dificultades que hubo para algunas acciones.

Por lo que en esta práctica yo aprendí tanto por la parte manual como por la investigación que se hizo de que siempre hay que pensar bien si usar un proceso o un hilo. Por lo que investigue esta elección de pende de muchos factores, para los procesos se suele elegir estos cuando una vez lanzado el hijo no se requiere demasiada comunicación con este o para gestionar entradas/salidas (atender simultáneamente a varias entradas de sockets, por ejemplo). Para los hilos según mi investigación se usa cuando se tienen que compartir y actualizarse datos y para hacer programas con muchos cálculos en paralelo.

En resumen, esta práctica me ayudo a ver de primera mano lo sencillo que es implementar hilos, que los hilos son mejores para cuando se requiere compartir memoria, pero hacen muchas acciones y hasta cierto grado como se gestiona la memoria en los sistemas Unix/Linux.



Pastrana Torres Victor Norberto.

La practica estuvo interesante, un poco laboriosa en cuanto al manejo de hilos porque para establecer la comunicación entre ellos necesita un cuidado extra ya que son apuntadores. Los apuntadores son lo que le dan el alma al lenguaje C, gracias a esta herramienta este lenguaje puede manejar la memoria a bajo nivel, esto es algo que pocos lenguajes logran hacer, además otras de las ventajas de los apuntadores es que mediante su uso los programas tienen una velocidad increíble si lo comparamos con otros lenguajes de programación.

Algunos de los obstáculos que enfrentamos fue que al tratar de pasar los datos a los hilos es que el tipo de punteros no coincidía, ya que el hilo únicamente recibe argumentos de tipo void y nosotros necesitamos enviar un entero, para esto recurrimos a una técnica sencilla pero muy efectiva, el cast, esto permite hacer la conversión entre tipos de datos, para enviar el argumento lo convertíamos en void, y una vez que el dato estuviera en el hilo lo pasábamos a entero con otro cast. El retorno de valores también fue otro detalle porque de igual forma solo se pueden retornar cierto tipo de datos, para resolver este detalle recurrimos a variables globales, este tipo de datos pueden ser usadas entre los hilos y el proceso sin restricciones. Otra de las opciones que teníamos era crear una región crítica a la que los hilos y el proceso tuvieran acceso, el detalle es que esta solución conlleva más atención por lo que preferimos variables globales.

Otro de los obstáculos que en lo personal tuve que enfrentar fue al uso de archivos, en Windows los trabajo sin problemas pero existen algunas diferencias en cuanto al uso de archivos en Windows y en Linux, una de las diferencias que note fue a las formas de abrir un archivo, en Linux existen más funciones a parte de las conocidas fopen() o fclose(), la funciones que son similares es read() y write() estas funciones realizan las mismas acciones, la diferencia radica en el como lo hacen, pues mientras las funciones f usan un puntero FILE, las otras funciones usan un valor entero, así que tuve confusiones al no decidir que tipo de funciones utilizaría.

Los problemas que tuvimos los solucionamos en parte gracias a la investigación previa y también a las que realizábamos mientras llevamos a cabo la realización de los programas, así que aprendimos más cosas acerca del sistema Linux y también del lenguaje C.



Bibliografía

- [1] TutorialsPoint, «Tutorialspoint,» 2020. [En línea]. Available: https://www.tutorialspoint.com/operating_system/os_multi_threading.htm. [Último acceso: Octubre 2020].
- [2] CEAC, «CEAC,» CEAC, 01 02 2018. [En línea]. Available: <https://www.ceac.es/blog/que-es-la-programacion-multihilo-y-que-ventajas-tiene>. [Último acceso: 08 2020].
- [3] fingEDU, fingEDU, 2020. [En línea]. Available: <https://www.fing.edu.uy/tecnoinf/mvd/cursos/so/material/teo/so05-hilos.pdf>. [Último acceso: 2020].
- [4] GURU99, «GURU99,» 2020. [En línea]. Available: <https://www.guru99.com/difference-between-process-and-thread.html#:~:text=Process%20means%20a%20program%20is,Lightweight%2C%20whereas%20Threads%20are%20Lightweight.&text=A%20Process%20is%20mostly%20isolated,share%20data%20with%20each%20other..> [Último acceso: Octubre 2020].
- [5] Universidad Carlos III de Madrid, «UC3M,» 2020. [En línea]. Available: http://ocw.uc3m.es/ingenieria-informatica/sistemas-operativos/material-de-clase-1/mt_t2_l5.pdf. [Último acceso: Octubre 2020].
- [6] sistemasoper2, «sistemasoper2,» sistemasoper2, 2020. [En línea]. Available: <https://sistemasoper2.wordpress.com/2014/10/21/usos-mas-comunes-de-los-hilos/>. [Último acceso: Octubre 2020].
- [7] jovenclub, «jovenclub,» jovenclub, 2020. [En línea]. Available: <https://gutl.jovenclub.cu/como-crear-hilos-en-c-de-forma-facil/>. [Último acceso: Octubre 2020].
- [8] slideshare, «slideshare,» slideshare, 8 Enero 2019. [En línea]. Available: <https://es.slideshare.net/EmmanuelGarcaFortuna/procesos-e-hilos-en-los-sistemas-operativos>. [Último acceso: 2020].