INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

SISTEMAS OPERATIVOS

PRÁCTICA 5:

Hilos

ALUMNOS:

Martínez Ramírez Sergi Alberto – 2020630260 - 2CV15
Meza Vargas Brandon David – 2020630288 - 2CM15
Peña Atanasio Alberto – 2020630367 - 2CV15
Sarmiento Gutiérrez Juan Carlos – 2020630386 - 2CV15

PROFESOR:

José Alfredo Jiménez Benítez

Índice de contenido

Índice de figuras

Glosario de términos

Hilo: un hilo es un proceso ligero o subproceso es una secuencia de tareas encadenadas muy pequeña que puede ser ejecutada por un sistema operativo.

Variable global: Es aquella variable a la que se puede hacer referencia a su dirección de memoria en cualquier parte del programa.

Contenido

Creación y gestión de hilos

Un hilo es una unidad básica de utilización de la CPU, la cual contiene un id de hilo, su propio program counter, un conjunto de registros y una pila; que se representa a nivel del sistema operativo con una estructura llamada TCB (thread control block).

Los hilos comparten con otros hilos que pertenecen al mismo proceso la sección de código, la sección de datos, entre otras cosas. Si un proceso tiene múltiples hijos, puede realizar más de una tarea a la vez (esto solo es real cuando se tiene más de una CPU).

Un ejemplo de uso es en un servidor web que tiene varios clientes, entonces en lugar de crear varios procesos para atender a cada proceso, se crean hilos. Generalmente es mejor el uso de hilos, ya que es más eficiente.

Aquí algunas ventajas que tiene el hacer uso de los hilos:

- El tiempo de respuesta mejora, ya que el programa puede estar ejecutándose, aunque parte del programa este bloqueado
- Los hilos comparten memoria y los recursos al proceso que pertenecen, por lo que se pueden tener varios hilos en el mismo espacio de direcciones
- Es más fácil la creación, cambio de contexto y gestión de hilos que de procesos
- Permite que hilos de un mismo proceso se ejecuten en diferentes CPUs

Los hilos comparten ciertos recursos con el resto de los hilos: señales, mapa de memoria, ficheros abiertos, semáforos, temporizadores, variables.

Las funciones necesarias para el manejo de hilos en c se encuentran en la biblioteca pthread.h. Primeramente tenemos la función que crea el hilo, esta es:

*int pthread_create(pthread_T *thread, const pthread_attr_t *attr, void*(*rutina)(void*), void *arg)*

El primer parámetro guarda el identificador del hilo que creamos. El segundo es un puntero a una struct con los atributos del hilo (podemos pasar NULL). El tercer parámetro es como un puntero especial, este recibe una función y será lo que ejecute el hilo, aquí se pasa la dirección de la función. El ultimo parámetro es un puntero y solo se pasa un argumento, aquí se recomienda que sea una struct.

La función que devuelve le identidicador de hilo del hilo que la ejecuta es:

pthread_tpthread_self(void)

Por otra parte, tenemos la función que sirve para esperar a otro hilo:

*int pthread_join(pthread_t thread, void **value)*

La función anterior recibe un identificador de hilo (parámetro thread) al que debemos esperar. El segundo parámetro es el valor de terminación del hilo.

La última función sirve para finalizar al hilo:

*int pthread_exit(void *value)*

Recibe un parámetro que será el valor de terminación.

Tres de las funciones regresan un entero, por lo que, si ocurre un error, se regresa un -1.

Desarrollo de la práctica

Programa 51

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. El código completo se muestra en los anexos.

Ahora se presenta el programa 51. Véase la imagen 1.

```
void *HiloP(void *args){ //Inicio de la función del hilo principal
    printf("Hilo creado\n"); //Impresión en pantalla
    int i;
    for(i = 0; i < 2; i++){ //for que itera desde 0 hasta 1
        pthread_t hilosH; //Declaración de la variable de los hilos secundarios

        printf("\tCreando otro hilo dentro del hilo principal...\n"); //Impresión en pantalla
        pthread_create(&hilosH, NULL, HilosH, NULL); //Creación de un hilo hijo

        pthread_join(hilosH, NULL); //El hilo principal espera la ejecución del hilo hijo

        printf("\tHilo terminado\n"); //Impresión en pantalla
    }

    printf("Terminando hilo principal...\n"); //Impresión en pantalla
    pthread_exit(NULL); //Termina el hilo principal
}

void *HilosH(void *args){ //Inicio de la función del hilo hijo
    printf("\tHilo creado\n"); //Impresión en pantalla
    printf("\tTerminando hilo...\n"); //Impresión en pantalla

    pthread_exit(NULL); //Termina el hilo hijo
}
```

Imagen 1. Funciones de los hilos del programa 51.

En la imagen anterior se puede observar dos funciones de hilos, la primera función es del hilo principal; primero hace una impresión en pantalla y ejecuta un for, el cuál imprime en pantalla y crea un hilo, luego termina a que el hilo termina y vuelve a imprimir en pantalla, este código lo ejecuta dos veces, por lo tanto, se crean 2 hilos, al terminar el for el hilo principal imprime en pantalla y termina.

La función de los hilos hijos solo imprimen 2 mensajes en pantalla y después terminan.

En la imagen 2 se observa la función main.

```c
int main(int argc, char const *argv[])
{
    pthread_t hiloP; //Declaración de la variable del hilo principal

    printf("Creando hilo principal...\n"); //Impresión en pantalla
    pthread_create(&hiloP, NULL, HiloP, NULL); //Creación del hilo principal

    pthread_join(hiloP, NULL); //El proceso principal espera la ejecución
                               //del hilo principal

    printf("Hilo terminado\n"); //Impresión en pantalla
    printf("Adios");//Impresión en pantalla

    return 0; //Fin del programa
}
```

Imagen 2. Función main() del programa 51.

En la imagen 2 solo se observa la creación del hilo principal, el proceso padre espera a que el hilo terminé, después imprime dos mensajes en pantalla y termina.

Finalmente, en la imagen 3 podemos ver la correcta compilación y ejecución del programa 51, arrojando los resultados esperados.

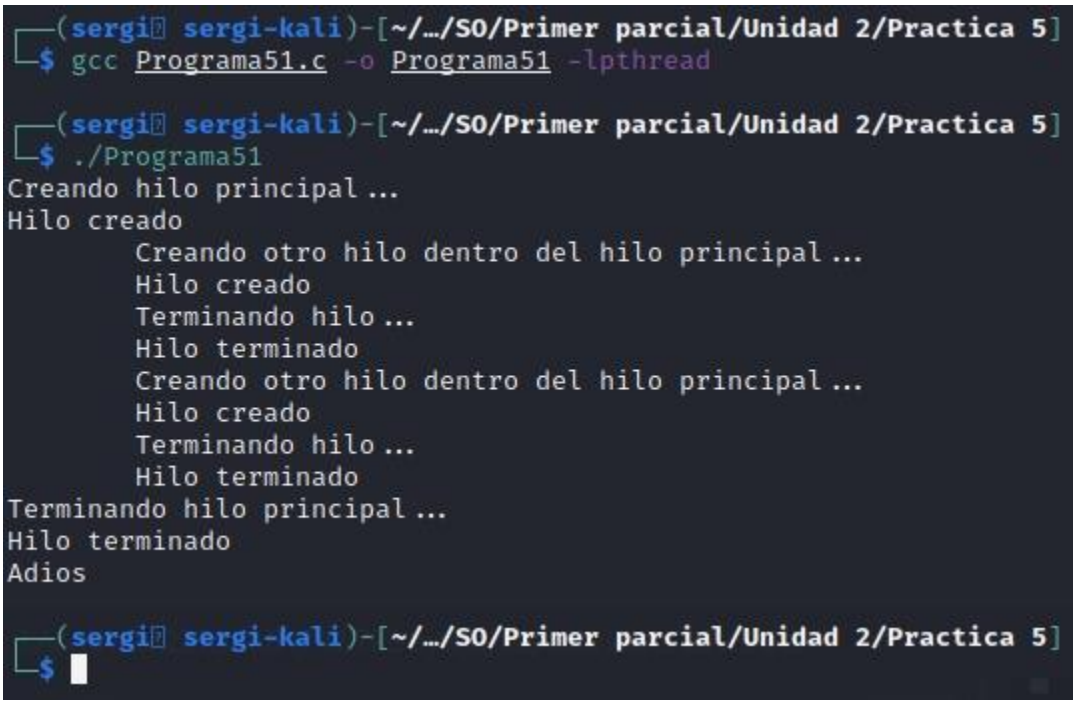


Imagen 3. Compilación y ejecución del programa 51.

Programa 52

En este programa se solicita lo siguiente:

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 imprimará el valor de la variable global. El código completo del programa se encuentra en los anexos.

Para este programa se crearon dos hilos, de los cuales se puede ver su código en la imagen 4.

```
void *hilo1(){ //código del primer hilo
    for (int i = 0; i < 5; i++) //for que hace 5 iteraciones en las que se incrementará glo
    {
        printf("Incrementando la variable global en %d unidades\n", A);
        glo+=A; //incrementa glo en 10 unidades
    }


    pthread_exit(NULL); //se acaba el hilo
}

void *hilo2(){ //código del segundo hilo

    for (int i = 0; i < 5; i++)//for que hace 5 iteraciones en las que se decrementará glo
    {
        printf("Decrementando la variable global en %d unidades\n", B);
        glo-=B; //decremetna glo en 5 unidades
    }


    pthread_exit(NULL);//se acaba el hilo
}
```

Imagen 4. Código del hilo 1 y del hilo 2.

Como vemos, los códigos son similares, el hilo uno se encarga de aumentar una variable global en A unidades, esto se hará las 5 veces que indica el ciclo for. En el hilo 2, se decrementa la variable global en B unidades.

En el main(), hacemos la creación de cada hilo, además de esperar por la ejecución de cada hilo y mostramos el valor que tiene al final la variable global, esto lo vemos en la imagen 5.

```
pthread_create(&IDhilo1, NULL, hilo1, NULL); //se crea el hilo 1
printf("Hilo 1 creado\n");
pthread_join(IDhilo1, NULL); //se espera a que termine de ejecutarse el hilo 1
printf("Hilo 1 finalizado:\n\n");

pthread_create(&IDhilo2, NULL, hilo2, NULL); //Se crea el hilo 2
printf("Hilo 2 creado\n");
pthread_join(IDhilo1, NULL);//se espera a que termine de ejecutarse el hilo 2
printf("Hilo 2 finalizado:\n\n");

printf("El valor final de la variable global es: %d\n", glo);// se imprime el valor final de glo
```

Imagen 5. Código dentro del main().

Como se ve, se hace uso de las funciones revisadas en la teoría.

La imagen 6 muestra la compilación y ejecución de este programa.

```
brandon@BDMV:~/programas so/practica 5$ gcc programa52.c -o p52 -lpthread
brandon@BDMV:~/programas so/practica 5$ ./p52
Hilo 1 creado
Incrementando la variable global en 10 unidades
Incrementando la variable global en 10 unidades
Incrementando la variable global en 10 unidades
Incrementando la variable global en 10 unidades
Incrementando la variable global en 10 unidades
Hilo 1 finalizado:

Hilo 2 creado
Decrementando la variable global en 5 unidades
Decrementando la variable global en 5 unidades
Decrementando la variable global en 5 unidades
Decrementando la variable global en 5 unidades
Decrementando la variable global en 5 unidades
Hilo 2 finalizado:

El valor final de la variable global es: 25
```

Imagen 6. Compilación y ejecución del programa.

Vemos que es necesario incluir la librería -lpthread al momento de compilar el programa, de otro modo surgirán errores. Podemos ver como el hilo uno incrementa el valor de la variable en 10 unidades y el hilo 2 lo decrementa en 5 unidades. El resultado final de la variable global es de 25.

Programa 53

En este programa se solicita lo siguiente:

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. El código completo se encuentra en los anexos.

Primeramente, se creó un proceso en el main, dicho proceso crea 3 hilos con ayuda de la función *pthread_create()*. Para tratar de que los hilos se vayan creando en orden se emplearon sleeps, una vez creados los hilos se espera a que terminen con la función *pthread_join()*. Esto lo vemos en la imagen 7.

```c
if(pid==0){

    printf("Proceso hijo con ID: %d, creare 3 hilos\n\n", getpid());

    pthread_create(&h1, NULL, hilo1, NULL); //se crea el hilo 1
    sleep(2);

    pthread_create(&h2, NULL, hilo2, NULL); //se crea el hilo 2
    sleep(2);

    pthread_create(&h3, NULL, hilo3, NULL); //se crea el hilo 3
    sleep(2);

    pthread_join(h1, NULL); //se espera a que termine de ejecutarse el hilo 1
    sleep(1);
    pthread_join(h2, NULL);//se espera a que termine de ejecutarse el hilo 2
    sleep(1);
    pthread_join(h3, NULL);//se espera a que termine de ejecutarse el hilo 3
```

Imagen 7. Creación de los hilos a partir de un proceso hijo.

Para la creación de cada hilo se usaron 3 funciones que corresponden a los 3 hilos, estas funciones son idénticas, por lo que en la imagen 8 solo se muestra la función del hilo 1.

```c
void *hilo1(){
    pthread_t hs1, hs2; //variables para los hilos hijos

    printf("\tSoy el hilo 1 con ID: %d y mi proceso padre es: %d\n", pthread_self(), getpid());
    printf("\tCreare dos hilos\n");
    sleep(1);

    pthread_create(&hs1, NULL, hiloH1, NULL); //se crea el hilo 1
    pthread_create(&hs2, NULL, hiloH2, NULL); //se crea el hilo 2

    pthread_join(hs1, NULL); //se espera a que termine de ejecutarse el hilo hijo 1
    pthread_join(hs2, NULL);//se espera a que termine de ejecutarse el hilo hijo 2

    pthread_exit(NULL);//se acaba el hilo

}
```

Imagen 8. Función del hilo 1.

Como se ve en la imagen anterior, el hilo imprime su id y el id del proceso que lo creó, además de crear dos hilos más y esperar por su correspondiente finalización.

En la imagen 9 podemos ver las funciones que corresponden a los dos hilos creados por cada uno de los 3 hilos creados al inicio.

```c
void *hiloH1(){ //función del hilo hijo número 1
    printf("\t\tSoy el hilo hijo 1 con ID: %d\n", pthread_self());

    pthread_exit(NULL);//se acaba el hilo
}

void *hiloH2(){ //función del hilo hijo número 2
    printf("\t\tSoy el hilo hijo 2 con ID: %d\n", pthread_self());

    pthread_exit(NULL);//se acaba el hilo
}
```

Imagen 9. Funciones de los hilos hijos creados por los 3 hilos iniciales.

Como vemos, las dos funciones de estos hilos son iguales, se encargan de imprimir su identificador y posteriormente finalizar.

En la imagen 10 podemos observas la correcta compilación y ejecución de este programa.

```
brandon@BDMV:~/programas so/practica 5$ gcc programa53.c -o p53 -lpthread
brandon@BDMV:~/programas so/practica 5$ ./p53
Proceso hijo con ID: 2444, creare 3 hilos

        Soy el hilo 1 con ID: -1556621568 y mi proceso padre es: 2444
        Creare dos hilos
                Soy el hilo hijo 1 con ID: -1565014272
                Soy el hilo hijo 2 con ID: -1573406976
        Soy el hilo 2 con ID: -1573406976 y mi proceso padre es: 2444
        Creare dos hilos
                Soy el hilo hijo 1 con ID: -1556621568
        Soy el hilo 3 con ID: -1565014272 y mi proceso padre es: 2444
        Creare dos hilos
                Soy el hilo hijo 1 con ID: -1590298880
                Soy el hilo hijo 2 con ID: -1598691584
                Soy el hilo hijo 2 con ID: -1581906176

Proceso padre con ID: 2443
```

Imagen 10. Compilación y ejecución del programa 53.

Como se ve, al final se despliega un hijo dos que corresponde al hilo 2, sin embargo, por como el sistema operativo va creando los hilos, estos no salen en orden.

Programa 54

“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 reportan los resultados devueltos por los otros dos hilos.”

El código completo de este programa lo podemos ver en los anexos.

Para este programa se creó el archivo Practica54.c en el cual se crean tres hilos para las tres tareas que menciona el problema de la práctica. Esto lo vemos en la imagen 11.

```
pthread_t hilo1;
pthread_t hilo2;
pthread_t hilo3;
pthread_create(&hilo1,NULL,Codigo_Hilo1,NULL);
pthread_join(hilo1,NULL);
printf("Hilo numero 1 terminado\n");
pthread_create(&hilo2,NULL,Codigo_Hilo2,NULL);
pthread_join(hilo2,NULL);
printf("Hilo numero 2 terminado\n");
pthread_create(&hilo3,NULL,Codigo_Hilo3,NULL);
pthread_join(hilo3,NULL);
printf("Hilo numero 3 terminado\n");
pthread_exit(NULL);
return 0;
```

Imagen 11. Código de creación de los hilos

Después para la instrucción para el hilo1, este cuenta las ocurrencias con una cadena de texto que está en otro archivo, en este caso el Archivo1.txt, lo podemos ver en la imagen 12.

```
juan@juanpc:~/Escritorio/PracticasSO/P5/P54$ cat Archivo1.txt
Esto es un archivo de prueba probando pero probalemente obtenga las cadenas igua
les a pr.
juan@juanpc:~/Escritorio/PracticasSO/P5/P54$
```

Imagen 12. Contenido de Archivo1.txt

Después creamos el código para la lectura de archivos, para este caso ocuparemos la función del flujo de datos “FILE”, para ello primero definimos nuestro flujo coma la dirección en memoria del archivo que deseamos abrir, indicando con qué permiso lo leeremos, en este caso un “rb” de read binary, después mediante funciones del mismo flujo de datos obtenemos la longitud de los caracteres contenidos en el archivo, esto lo hacemos con la intención de preservar la memoria suficiente para poder manejar la cadena obtenida, en seguida hay que contar las ocurrencias con una cadena de texto para este caso seleccionamos la cadena “pr”, después mediante una función que se programó llamada cuentaOcurrencia, obtendremos en número entero el número de ocurrencias de la cadena seleccionada en el archivo de texto, esa sería la funcionalidad de nuestro hilo1. Esta parte la podemos ver en la imagen 13.

```
void *Codigo_Hilo1(void *arg){
        printf("Contando ocurrencias en el archivo, similares a 'pr'\n");
        FILE* flujo = fopen("Archivo1.txt","rb");
        fseek(flujo,0,SEEK_END);
        int num_elementos = ftell(flujo);
        rewind(flujo);
        char* cadena = (char*)calloc(sizeof(char),num_elementos);
        int num_elementos_leidos = fread(cadena,sizeof(char),num_elementos,flujo);
        char cadena2[2]="pr";
        NumeroOcurrencias = cuentaOcurrencia(cadena,cadena2);
        printf("El numero de Ocurrencias encontradas fue de: %d\n",NumeroOcurrencias);
        free(cadena);
        fclose(flujo);
        pthread_exit(NULL);
}
```

Imagen 13. Código del hilo1

En la imagen 14, podemos ver la parte del código de la función que se encarga de contar las ocurrencias de la cadena pr.

```c
int cuentaOcurrencia(char *cadena, char *subcadena){
        char *tmp = cadena;
        char *pdest;
        int ocur = 0, pos;
        int len = strlen(subcadena);
        while(1){
                pdest = strstr(tmp, subcadena);
                if( !pdest ) break;
                pos = pdest - tmp;
                tmp += pos + len;
                ocur++;
        }
        return ocur;
}
```

Imagen 14. Código de la función cuentaOcurrencia

Para nuestro hilo2 hay que enviar una instrucción por sistema de que copie los archivos en el directorio actual y los mueva a otro directorio para nuestro caso “subdirectorio”, para ello recurrimos a la función system de C en la cual escribimos comandos de consola, en este caso usaremos un cp para la copia de archivos .c y .txt posteriormente indicamos el directorio, después mediante el comando “ls -A1 subdirectorio/ | wc -l” contamos el número de elementos copiados a este directorio, después lo asignamos a una variable global y listo. Lo podemos ver en la imagen 15.

```c
void *Codigo_Hilo2(void *arg){
        system("cp *.txt *.c subdirectorio/");
        system("ls -A1 subdirectorio/ | wc -l");
        NumeroElementosCopiados = 3;
        pthread_exit(NULL);
}
```

Imagen 15. Código del hilo2

Para el tercer y último hilo hay que generar un reporte de los datos obtenidos por los hilos anteriores en un archivo externo para nuestro caso Archivo2.txt, para esto nuevamente usamos el flujo de datos solo que ahora lo abrimos con el permiso “w” de write, escritura, en él mediante la función fputs() escribiremos una cadena que en este caso el número de ocurrencias obtenidas del hilo1 y el número de archivos copiados del hilo2 concatenamos texto con la variable entera y así obtenemos una cadena misma que mandamos a la función para que se genere nuestro reporte. Esta parte la vemos en la imagen 16.

```
void *Codigo_Hilo3(void *arg){
        printf("Generando Reporte de datos\n");
        FILE* flujo = fopen("Archivo2.txt","w");

        char cadena1[200] = "El numero de ocurrencias encontradas en el Archivo1.txt fue de: ";
        char ocurr[10];
        sprintf(ocurr,"%d",NumeroOcurrencias);
        strcat(cadena1,ocurr);
        strcat(cadena1,"\n");

        char cadena2[200] = "El numero de elementos copiados al subdirectorio fue de: ";
        char copi[10];
        sprintf(copi,"%d",NumeroElementosCopiados);
        strcat(cadena2,copi);
        strcat(cadena2,"\n");

        fputs(cadena1,flujo);
        fputs(cadena2,flujo);
        fflush(flujo);
        fclose(flujo);
        pthread_exit(NULL);
}
```

Imagen 16. Código del hilo3

Nuestro reporte en el Archivo2.txt quedaría como se ve en la imagen 17.

```
juan@juanpc:~/Escritorio/PracticasSO/P5/P54$ cat Archivo2.txt
El numero de ocurrencias encontradas en el Archivo1.txt fue de: 4
El numero de elementos copiados al subdirectorio fue de: 3
juan@juanpc:~/Escritorio/PracticasSO/P5/P54$
```

Imagen 17. Contenido del Archivo2.txt

A continuación, la compilación de nuestro archivo Practica54.c que se ve en la imagen 18.

```
juan@juanpc:~/Escritorio/PracticasSO/P5/P54$ gcc Practica54.c -o P54 -lpthread
juan@juanpc:~/Escritorio/PracticasSO/P5/P54$
```

Imagen 18. Compilación exitosa archivo Practica54.c

Finalmente, su ejecución en consola, donde podemos ver el resultado del programa en la imagen 19.

```
juan@juanpc:~/Escritorio/PracticasSO/P5/P54$ ./P54
Contando ocurrencias en el archivo, similares a 'pr'
El numero de Ocurrencias encontradas fue de: 4
Hilo numero 1 terminado
3
Hilo numero 2 terminado
Generando Reporte de datos
Hilo numero 3 terminado
juan@juanpc:~/Escritorio/PracticasSO/P5/P54$
```

Imagen 19. Resultado del programa P54 en consola.

Programa 55

“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.” El código completo de este programa lo podemos ver en los anexos.

Para realizar esta práctica, crearemos el archivo Practica55.c en donde nuevamente crearemos hilos para realizar las problemáticas planteadas.

Primero creamos 3 hilos que serán los necesarios para cubrir las acciones, decidir si un número proporcionado por el usuario es compuesto para el hilo1, si el número es compuesto descomponerlo en sus números primos para el hilo2, y en caso de que el número no sea compuesto, decir que es primo en nuestro hilo3.

Creamos el Código para definir 3 hilos e inicializarlos, este código lo vemos en la imagen 20.

```
pthread_t hilo1;
pthread_t hilo2;
pthread_t hilo3;

pthread_create(&hilo1,NULL,Codigo_Hilo1,NULL);
pthread_join(hilo1,NULL);
printf("\nHilo numero 1 terminado\n");

pthread_create(&hilo2,NULL,Codigo_Hilo2,NULL);
pthread_join(hilo2,NULL);
printf("\nHilo numero 2 terminado\n");

pthread_create(&hilo3,NULL,Codigo_Hilo3,NULL);
pthread_join(hilo3,NULL);
printf("\nHilo numero 3 terminado\n");

pthread_exit(NULL);
```

Imagen 20. Creación e inicialización de los hilos 1, 2 y 3.

Para el hilo 1, el cual se muestra en la imagen 21, hay que saber que es un número compuesto es por ello que hay que saber su definición, *“Un número compuesto es cualquier número natural no primo que tiene uno o más divisores distintos a 1 y a sí mismo.”* una vez que sabemos esto continuemos, primero solicitamos un número, este será pasado a una función llamada EsCompuesto(), la cual identificará si nuestro número es o no compuesto, es decir tiene un divisor distinto de uno o si mismo, si lo es este hilo1 se lo comunicará al hilo2 para que continúe con la siguiente funcionalidad. A Continuación el código de la función EsCompuesto() en la imagen 22.

```
void *Codigo_Hilo1(void *arg){
        printf("\nIntroduce un numero: \n");
        scanf("%d",&numeroCompuesto);
        if(EsCompuesto(numeroCompuesto)>=2)
                printf("El numero %d es compuesto",numeroCompuesto);
        pthread_exit(NULL);
}
```

Imagen 21. Código del hilo1, Programa55.c

```
int EsCompuesto(int numC){
        int i;
        for(int i=2;i<numC;i++){
                if(numC%i==0)
                        return i;
        }
        return 0;
}
```

Imagen 22. Código de la función EsCompuesto

Después en el hilo 2 mediante la función DeterminaPrimos() (ver imagen 25) y pasándole como parámetro nuestro número que ya sabemos que si es compuesto, lo descomponemos en estos números, la función realiza lo siguiente en un primer bucle for almacena todos los posibles divisores del número sean pares o primos estos son guardados en un arreglo llamado Números, después en un segundo bucle for comprueba cada elemento de este número para saber si es primo o no, esto mediante la función esPrimo() (ver figura 24) en caso de serlo este número primo es almacenado en otro arreglo llamado NumerosPrimos en caso de ser par es desechado, finalmente en el tercer bucle for se imprimen cada uno de los elementos del arreglo NumerosPrimos.

```
void *Codigo_Hilo2(void *arg){
        if(EsCompuesto(numeroCompuesto)>=2)
                DeterminaPrimos(numeroCompuesto);
        pthread_exit(NULL);
}
```

Imagen 23. Código del hilo 2, Programa55.c

```
int esPrimo(int n){
        int bandera = 1;
        for(int j=2;j<n;j++){
                if(n%j==0)
                        bandera=0;
        }
        return bandera;
}
```

Imagen 24. Código función esPrimo()

```
void DeterminaPrimos(int numC){
        int i;
        int primos=0;
        int posicionArray = 0;
        for(i=2;i<numC;i++){
                if(numC%i==0){
                        Numeros[posicionArray]=i;
                        posicionArray++;
                }
        }
        for(i=0;i<ARR_LENGTH;i++){
                if(esPrimo(Numeros[i])==1){
                        NumerosPrimos[i]=Numeros[i];
                        primos++;
                }
        }
        if(primos>0)
                printf("El numero %d esta compuesto por los siguientes numeros primos: ",numeroCompuesto);
        else
                printf("El numero %d no esta compuesto por numeros primos",numeroCompuesto);
        for(i=0;i<ARR_LENGTH;i++){
                if(NumerosPrimos[i]>0)
                        printf("%d ",NumerosPrimos[i]);
        }
        printf("\n");
}
```

Imagen 25. Código de la función DeterminaPrimo()

Finalmente, para el hilo3 caso de que el hilo1 determine que el número introducido no está compuesto este hilo simplemente dice que es primo. Esto lo vemos en la imagen 26.

```
void *Codigo_Hilo3(void *arg){
        if(EsCompuesto(numeroCompuesto)<2)
                printf("\nEl numero %d es primo\n",numeroCompuesto);
        pthread_exit(NULL);
}
```

Imagen 26. Código del hilo3, Programa55.c

A continuación, la compilación del Programa55.c en la imagen 27.

```
juan@juanpc:~/Escritorio/PracticasSO/P5/P55$ gcc Practica55.c -o P55 -lpthread
juan@juanpc:~/Escritorio/PracticasSO/P5/P55$
```

Imagen 27. Compilación exitosa del Programa55.c

A continuación, la ejecución del programa que se ve en la imagen 28, junto con los resultados esperados.

```
juan@juanpc:~/Escritorio/PracticasSO/P5/P55$ ./P55

Introduce un numero:
219
El numero 219 es compuesto
Hilo numero 1 terminado
El numero 219 esta compuesto por los siguientes numeros primos: 3 73

Hilo numero 2 terminado

Hilo numero 3 terminado
juan@juanpc:~/Escritorio/PracticasSO/P5/P55$
```

Imagen 28. Ejecución del programa P55 en consola

Conclusiones

Martínez Ramírez Sergi Alberto

En esta práctica pudimos apreciar el uso de hilos, su funcionamiento y posibles implementaciones, en esta práctica también presenté algunos problemas, principalmente a la hora de compilar el programa, me saltaban varios errores, investigué algunos de ellos, pero la mayoría era por errores de incompatibilidad de datos, más en específico el error salía en la llamada al sistema "pthread_create" ya que al pasarle la función "HiloP", me decía que había un error de parseo, al principio escribí el código idéntico al de los programas de ejemplo y salía ese error, después traté de escribir las funciones de tipo normal y parsearlas a apuntador en la llamada al sistema y no funcionaba, traté de pasar por referencia a la llamada al sistema y lo mismo. Dejé el programa por la paz y al siguiente día, decidí buscar un ejemplo de implementación de hilos en internet, la persona que escribía el código tenía el mismo código que yo, utilizaba el mismo compilador que yo y a él sí le compilaba sin errores, la diferencia era que el escribía sus funciones como tipo apuntador y las declaraba al principio del programa. No llegué a pensar que fuera por eso, ya que el error era de tipo de dato y no me decía nada de que las funciones no estaban declaradas o algo así, al final resulto ser por eso, al declarar las funciones el código compiló sin errores ni warnings. Actualmente sigo sin saber porque si no declaro las funciones tipo apuntador me sale ese error, pero sigo investigando.

Meza Vargas Brandon David

Los hilos son una parte fundamental de cualquier sistema operativo al momento de realizar distintas acciones, al realizar esta práctica comprendí de una mejor manera como es que funcionan y su implementación en c, ya que anteriormente había implementado hilos, pero en Java. También con la realización me di cuenta de una diferencia que existe entre los procesos y los hilos, ya que en los procesos si un proceso padre termina, los procesos hijos continúan con su ejecución, pero con los hilos no pasa esto, si un hilo padre termina, entonces también lo harán sus hijos. Esto lo podemos ver en los programas si no colocamos la línea de código que corresponde a la espera que hace el hilo padre a que acaben sus hijos.

Sin duda fue una práctica de suma importancia para entender mejor el tema de los hilos, viendo claramente con los programas 54 y 55 como cada hilo puede hacer acciones independientes de otros hilos, y también como dos hilos pueden modificar variables globales como se vio en el programa 52.

Peña Atanasio Alberto

Hoy en día las aplicaciones de los hilos son muy amplias, debido a que la mayoria de dispositivos que utilizamos cómo el Sistema Operativo de nuestra computadora o nuestro smarthpone, las televisiones, los cajeros, entre otros, necesitan procesos y en consecuencia una serie de recursos. Los hilos son una unidad de ejecución que poseen registros, una pila y un contador. El objetivo de los hilos o procesos ligeros es lograr paralelismo dividiendo un proceso en multiples subprocesos.

Se considera importatne el conocimiento de los hilos debido a que de esta forma se podra tener una gran idea de cómo funcionan los procesos dentro de cualquier dispositivo o tecnología y cómo consecuencia una mejor solución si se llegaran a presentar problemas.

La práctica contribuyó a que se aclararan las ideas teóricas que se tenían acercan de los hilos y por ende se logro una mejor comprensión de estos, notadose que son una herramienta muy útil y práctica de aplicar.

Sarmiento Gutiérrez Juan Carlos

Durante el desarrollo de esta práctica entendí mejor el concepto de hilos así de cómo se implementan en lenguaje C, además esto me hace ver que cada vez podemos hacer tareas más eficiente implementando esta arquitectura ya que podemos poner tareas distintas a cada hilo y lo que es mejor que estos se comuniquen mediante el proceso que los creó esto además me hace reflexionar ya que si un proceso tuviese hilos y estos hilos a su vez hilos podríamos hacer tareas de manera muy rápida lo cual volvería mejor el cómputo, tal es el caso del programa 55 en el cual se determinaba si un número era compuesto y otros hilos se encargaran de descomponerlo sin duda una muy buena práctica para reforzar el conocimiento adquirido en clase.

Bibliografía

[1] J. Pérez, "Hilos y Procesos". [En línea], Disponible: http://ocw.uc3m.es/ingenieria-informatica/sistemas-operativos/material-de-clase-1/mt_t2_l5.pdf

[2] J. Camazón, "Creación y destrucción de hilos en Linux con c", 2018. [En línea]. Disponible: https://www.jesusninoc.com/06/20/creacion-y-destruccion-de-hilos-en-linux-con-c/

[3] L. Haylem, "Hilos en c", 2019. [En línea]. Disponible: https://gutl.jovenclub.cu/como-crear-hilos-en-c-de-forma-facil/

Anexos

Código programa 51

```c
#include <semaphore.h>
#include <unistd.h>

void *HiloP(void *args); //Declaración de la función del hilo principal
void *HilosH(void *args); //Declaración de la función de los hilos hijos

int main(int argc, char const *argv[])
{
    pthread_t hiloP; //Declaración de la variable del hilo principal

    printf("Creando hilo principal...\n"); //Impresión en pantalla
    pthread_create(&hiloP, NULL, HiloP, NULL); //Creación del hilo principal

    pthread_join(hiloP, NULL); //El proceso principal espera la ejecución
                               //del hilo principal

    printf("Hilo terminado\n"); //Impresión en pantalla
    printf("Adios");//Impresión en pantalla

    return 0; //Fin del programa
}

void *HiloP(void *args){ //Inicio de la función del hilo principal
    printf("Hilo creado\n"); //Impresión en pantalla
    int i;
    for(i = 0; i < 2; i++){ //for que itera desde 0 hasta 1
        pthread_t hilosH; //Declaración de la variable de los hilos secundarios

        printf("\tCreando otro hilo dentro del hilo principal...\n"); //Impresión en pantalla
        pthread_create(&hilosH, NULL, HilosH, NULL); //Creación de un hilo hijo

        pthread_join(hilosH, NULL); //El hilo principal espera la ejecución del hilo hijo

        printf("\tHilo terminado\n"); //Impresión en pantalla
    }

    printf("Terminando hilo principal...\n"); //Impresión en pantalla
    pthread_exit(NULL); //Termina el hilo principal
}

void *HilosH(void *args){ //Inicio de la función del hilo hijo
    printf("\tHilo creado\n"); //Impresión en pantalla
    printf("\tTerminando hilo...\n"); //Impresión en pantalla

    pthread_exit(NULL); //Termina el hilo hijo
}
```

Imagen 29. Código del programa 51.

Código programa 52

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h> //libreria para usar hilos
#include <unistd.h>
#define A 10 //unidades en las que se incremente glo
#define B 5 //unidades en las que se decrementa glo
int glo=0; //Variable global

void *hilo1(){ //código del primer hilo
    for (int i = 0; i < 5; i++) //for que hace 5 iteraciones en las que se incrementará glo
    {
        printf("Incrementando la variable global en %d unidades\n", A);
        glo+=A; //incrementa glo en 10 unidades
    }
    pthread_exit(NULL); //se acaba el hilo
}
void *hilo2(){ //código del segundo hilo
    for (int i = 0; i < 5; i++)//for que hace 5 iteraciones en las que se decrementará glo
    {
        printf("Decrementando la variable global en %d unidades\n", B);
        glo-=B; //decremetna glo en 5 unidades
    }
    pthread_exit(NULL);//se acaba el hilo
}

int main(){
    pthread_t IDhilo1; //pthread_t es un entero largo, almacena el id del hilo
    pthread_t IDhilo2;
    pthread_create(&IDhilo1, NULL, hilo1, NULL); //se crea el hilo 1
    printf("Hilo 1 creado\n");
    pthread_join(IDhilo1, NULL); //se espera a que termine de ejecutarse el hilo 1
    printf("Hilo 1 finalizado:\n\n");

    pthread_create(&IDhilo2, NULL, hilo2, NULL); //Se crea el hilo 2
    printf("Hilo 2 creado\n");
    pthread_join(IDhilo1, NULL);//se espera a que termine de ejecutarse el hilo 2
    printf("Hilo 2 finalizado:\n\n");

    printf("El valor final de la variable global es: %d\n", glo);// se imprime el valor final de glo
    return 0;
}
```

Imagen 30. Código programa 52.

Código programa 53

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h> //libreria para usar hilos
#include <unistd.h>
#include <unistd.h> //permite utilizar la llamada al sistema fork()
#include <wait.h> //define las declaraciones de espera
#include <sys/types.h> //tipos de datos, permite usar pid_t

void *hiloH1(){ //función del hilo hijo número 1
    printf("\t\tSoy el hilo hijo 1 con ID: %d\n", pthread_self());

    pthread_exit(NULL);//se acaba el hilo
}

void *hiloH2(){ //función del hilo hijo número 2
    printf("\t\tSoy el hilo hijo 2 con ID: %d\n", pthread_self());

    pthread_exit(NULL);//se acaba el hilo
}

void *hilo1(){
    pthread_t hs1, hs2; //variables para los hilos hijos

    printf("\tSoy el hilo 1 con ID: %d y mi proceso padre es: %d\n", pthread_self(), getpid());
    printf("\tCreare dos hilos\n");


    pthread_create(&hs1, NULL, hiloH1, NULL); //se crea el hilo 1

    pthread_create(&hs2, NULL, hiloH2, NULL); //se crea el hilo 2

    pthread_join(hs1, NULL); //se espera a que termine de ejecutarse el hilo hijo 1
    pthread_join(hs2, NULL);//se espera a que termine de ejecutarse el hilo hijo 2

    pthread_exit(NULL);//se acaba el hilo

}

void *hilo2(){
    pthread_t hs1, hs2; //variables para los hilos hijos

    printf("\tSoy el hilo 2 con ID: %d y mi proceso padre es: %d\n", pthread_self(), getpid());
    printf("\tCreare dos hilos\n");


```

```c
    pthread_create(&hs1, NULL, hiloH1, NULL); //se crea el hilo 1
    pthread_create(&hs2, NULL, hiloH2, NULL); //se crea el hilo 2
    sleep(1);

    pthread_join(hs1, NULL); //se espera a que termine de ejecutarse el hilo hijo 1
    pthread_join(hs2, NULL);//se espera a que termine de ejecutarse el hilo hijo 2

    pthread_exit(NULL);//se acaba el hilo
}

void *hilo3(){
    pthread_t hs1, hs2; //variables para los hilos hijos

    printf("\tSoy el hilo 3 con ID: %d y mi proceso padre es: %d\n", pthread_self(), getpid());
    printf("\tCreare dos hilos\n");


    pthread_create(&hs1, NULL, hiloH1, NULL); //se crea el hilo 1
    pthread_create(&hs2, NULL, hiloH2, NULL); //se crea el hilo 2
    sleep(2);
    pthread_join(hs1, NULL); //se espera a que termine de ejecutarse el hilo hijo 1
    pthread_join(hs2, NULL);//se espera a que termine de ejecutarse el hilo hijo 2

    pthread_exit(NULL);//se acaba el hilo

}

int main(){

    pid_t pid; //guarda el id del procesp
    pthread_t h1, h2, h3;

    pid = fork();

    if(pid==0){

        printf("Proceso hijo con ID: %d, creare 3 hilos\n\n", getpid());

        pthread_create(&h1, NULL, hilo1, NULL); //se crea el hilo 1
        sleep(1);
        pthread_create(&h2, NULL, hilo2, NULL); //se crea el hilo 2

        pthread_create(&h3, NULL, hilo3, NULL); //se crea el hilo 3


         pthread_join(h1, NULL); //se espera a que termine de ejecutarse el hilo 1
         sleep(1);
         pthread_join(h2, NULL);//se espera a que termine de ejecutarse el hilo 2
         pthread_join(h3, NULL);//se espera a que termine de ejecutarse el hilo 3


    }else{
        wait(NULL);
        printf("\nProceso padre con ID: %d\n", getpid());
    }

    return 0;
}

```

Imagen 31. Código del programa 53.

Código programa 54

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

int NumeroOcurrencias;
int NumeroElementosCopiados;

int cuentaOcurrencia(char *cadena, char *subcadena){
    char *tmp = cadena;
    char *pdest;
    int ocur = 0, pos;
    int len = strlen(subcadena);
    while(1){
        pdest = strstr(tmp, subcadena);
        if( !pdest ) break;
            pos = pdest - tmp;
        tmp += pos + len;
        ocur++;
    }
    return ocur;
}
void *Codigo_Hilo1(void *arg){
    printf("Contando ocurrencias en el archivo, similares a 'pr'\n");
    FILE* flujo = fopen("Archivo1.txt","rb");
    fseek(flujo,0,SEEK_END);
    int num_elementos = ftell(flujo);
    rewind(flujo);
    char* cadena = (char*)calloc(sizeof(char),num_elementos);
    int num_elementos_leidos = fread(cadena,sizeof(char),num_elementos,flujo);
    char cadena2[2]="pr";
    NumeroOcurrencias = cuentaOcurrencia(cadena,cadena2);
    printf("El numero de Ocurrencias encontradas fue de: %d\n",NumeroOcurrencias);
    free(cadena);
    fclose(flujo);
    pthread_exit(NULL);
}
void *Codigo_Hilo2(void *arg){
    system("cp *.txt *.c subdirectorio/");
    system("ls -A1 subdirectorio/ | wc -l");
    NumeroElementosCopiados = 3;
    pthread_exit(NULL);
}
```

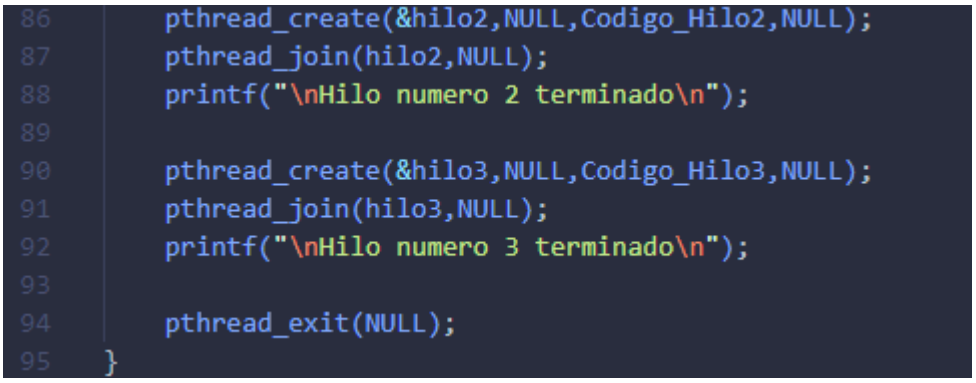
```c
void *Codigo_Hilo3(void *arg){
    printf("Generando Reporte de datos\n");
    FILE* flujo = fopen("Archivo2.txt","w");

    char cadena1[200] = "El numero de ocurrencias encontradas en el Archivo1.txt fue de: ";
    char ocurr[10];
    sprintf(ocurr,"%d",NumeroOcurrencias);
    strcat(cadena1,ocurr);
    strcat(cadena1,"\n");

    char cadena2[200] = "El numero de elementos copiados al subdirectorio fue de: ";
    char copi[10];
    sprintf(copi,"%d",NumeroElementosCopiados);
    strcat(cadena2,copi);
    strcat(cadena2,"\n");

    fputs(cadena1,flujo);
    fputs(cadena2,flujo);
    fflush(flujo);
    fclose(flujo);
    pthread_exit(NULL);
}

int main(){
    pthread_t hilo1;
    pthread_t hilo2;
    pthread_t hilo3;
    pthread_create(&hilo1,NULL,Codigo_Hilo1,NULL);
    pthread_join(hilo1,NULL);
    printf("Hilo numero 1 terminado\n");
    pthread_create(&hilo2,NULL,Codigo_Hilo2,NULL);
    pthread_join(hilo2,NULL);
    printf("Hilo numero 2 terminado\n");
    pthread_create(&hilo3,NULL,Codigo_Hilo3,NULL);
    pthread_join(hilo3,NULL);
    printf("Hilo numero 3 terminado\n");
    pthread_exit(NULL);
    return 0;
}
```

Imagen 32. Código programa 54

Código programa 55

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

#define ARR_LENGTH 10

int Numeros[ARR_LENGTH];
int NumerosPrimos[ARR_LENGTH];
int numeroCompuesto;

int esPrimo(int n){
    int bandera = 1;
    for(int j=2;j<n;j++){
        if(n%j==0)
            bandera=0;
    }
    return bandera;
}

void DeterminaPrimos(int numC){
    int i;
    int primos=0;
    int posicionArray = 0;
    for(i=2;i<numC;i++){
        if(numC%i==0){
            Numeros[posicionArray]=i;
            posicionArray++;
        }
    }
    for(i=0;i<ARR_LENGTH;i++){
        if(esPrimo(Numeros[i])==1){
            NumerosPrimos[i]=Numeros[i];
            primos++;
        }
    }
    if(primos>0)
        printf("El numero %d esta compuesto por los siguientes numeros primos: ",numeroCompuesto);
    else
        printf("El numero %d no esta compuesto por numeros primos",numeroCompuesto);
    for(i=0;i<ARR_LENGTH;i++){
        if(NumerosPrimos[i]>0)
            printf("%d ",NumerosPrimos[i]);
    }
```

```c
    printf("\n");
}

int EsCompuesto(int numC){
    int i;
    for(int i=2;i<numC;i++){
        if(numC%i==0)
            return i;
    }
    return 0;
}

void *Codigo_Hilo1(void *arg){
    printf("\nIntroduce un numero: \n");
    scanf("%d",&numeroCompuesto);
    if(EsCompuesto(numeroCompuesto)>=2)
        printf("El numero %d es compuesto",numeroCompuesto);
    pthread_exit(NULL);
}

void *Codigo_Hilo2(void *arg){
    if(EsCompuesto(numeroCompuesto)>=2)
        DeterminaPrimos(numeroCompuesto);
    pthread_exit(NULL);
}

void *Codigo_Hilo3(void *arg){
    if(EsCompuesto(numeroCompuesto)<2)
        printf("\nEl numero %d es primo\n",numeroCompuesto);
    pthread_exit(NULL);
}

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

    pthread_create(&hilo1,NULL,Codigo_Hilo1,NULL);
    pthread_join(hilo1,NULL);
    printf("\nHilo numero 1 terminado\n");

```



```
86      pthread_create(&hilo2,NULL,Codigo_Hilo2,NULL);
87      pthread_join(hilo2,NULL);
88      printf("\nHilo numero 2 terminado\n");
89
90      pthread_create(&hilo3,NULL,Codigo_Hilo3,NULL);
91      pthread_join(hilo3,NULL);
92      printf("\nHilo numero 3 terminado\n");
93
94      pthread_exit(NULL);
95  }
```

Imagen 33. Código programa 55.