LABORATORIO DE SISTEMAS OPERATIVOS

PERÍODO ACADÉMICO: 2024 – A **EQUIPO:**

PROFESOR: Marco Sánchez PhD.

TIPO DE INSTRUMENTO: Guía de Laboratorio

TEMA: CREACIÓN DE HILOS

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

1. OBJETIVOS

1.1. Familiarizar al estudiante con el uso de las funciones **pthread.**
1.2. Realizar varias actividades de creación de hilos.

2. MARCO TEÓRICO

Manejados por el S.O.	Manejados por los procesos
Independientes de otros procesos	Relacionados con otros hilos del mismo proceso
Memoria privada, se necesitan mecanismos de comunicación para compartir información	Memoria compartida con el resto de hilos que forman el proceso

Tabla 1: Procesos e hilos

proceso

hilos

Procesos

Manejados por el Sistema Operativo (S.O.):

- Los procesos son unidades de ejecución independientes que son controladas por el sistema operativo.
- **Ejemplo**: Imagina que estás trabajando con dos programas diferentes en tu computadora, como un editor de texto y un navegador web. Cada uno de estos programas es un proceso separado y funciona de manera independiente uno del otro.

Independencia:

- Los procesos no pueden acceder directamente a la memoria de otros procesos. Cada uno tiene su propio espacio de memoria.
- **Ejemplo**: El navegador web no puede directamente leer o escribir en el editor de texto a menos que utilicen un mecanismo especial como copiar y pegar.

Memoria Privada:

- Cada proceso tiene su propia memoria privada y necesita mecanismos especiales, como archivos o redes, para compartir información.
- **Ejemplo**: Si quieres pasar información del navegador web al editor de texto, tendrías que guardar un archivo en el navegador y luego abrirlo con el editor de texto.
-

Hilos

Manejados por los Procesos:

- Los hilos son ***unidades de ejecución más pequeñas dentro de un proceso*** y son manejados por el propio proceso.
- **Ejemplo**: Piensa en un programa de edición de video. Dentro de este programa, podrías tener un hilo para cargar el video, otro para aplicar efectos y otro para guardar el archivo final. Todos estos hilos trabajan juntos dentro del mismo programa (proceso).

Interrelación:

- Los hilos dentro del mismo proceso pueden comunicarse y compartir datos directamente entre ellos.
- **Ejemplo**: En el programa de edición de video, el hilo que aplica efectos puede comunicar directamente con el hilo que guarda el archivo para asegurarse de que los cambios se guarden correctamente.

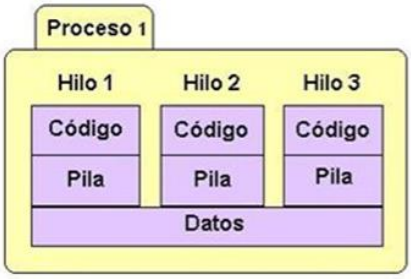
Memoria Compartida:

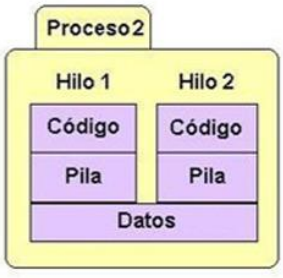
- Los hilos comparten el mismo espacio de memoria, lo que facilita la comunicación y el intercambio de datos.
- **Ejemplo**: Todos los hilos del programa de edición de video tienen acceso a la misma memoria, por lo que pueden trabajar juntos sin necesidad de mecanismos adicionales.

¿Qué Son los Hilos?

Para entender mejor qué son los hilos, piensa en un proceso como una fábrica (el programa). Dentro de esa fábrica, hay varias líneas de producción (hilos) que trabajan al mismo tiempo en diferentes tareas, pero todas contribuyen al mismo producto final.

- **Proceso (Fábrica):** La fábrica en sí es independiente de otras fábricas. Cada fábrica tiene su propio conjunto de recursos y trabaja en su propio producto.
- **Hilos (Líneas de Producción):** Dentro de la fábrica, las líneas de producción trabajan en paralelo y pueden compartir recursos fácilmente. Todas las líneas de producción están coordinadas para lograr un objetivo común.





Proceso 1

- Hilo 1, Hilo 2, Hilo 3: Dentro de "Proceso 1", hay tres hilos (Hilo 1, Hilo 2 y Hilo 3).
- Código: Cada hilo tiene su propia sección de código.
- **Pila**: Cada hilo también tiene su propia pila (stack), que es una estructura de datos utilizada para gestionar la ejecución de subrutinas y almacenar variables locales.
- **Datos Compartidos:** Todos los hilos dentro de "Proceso 1" comparten la misma sección de datos. Esto significa que pueden comunicarse fácilmente y acceder a los mismos datos, pero también significa que deben manejar cuidadosamente la sincronización para evitar conflictos.

Proceso 2

- Hilo 1, Hilo 2: Dentro de "Proceso 2", hay dos hilos (Hilo 1 y Hilo 2).
- Código: Cada hilo tiene su propia sección de código.
- Pila: Cada hilo también tiene su propia pila.
- Datos Compartidos: Similar a "Proceso 1", todos los hilos dentro de "Proceso 2" comparten la misma sección de datos.

Actividad Práctica

Identifica Procesos e Hilos en Windows

Dentro de un proceso, puede haber varios hilos de ejecución. Los hilos comparten la misma memoria del proceso. Para escribir programas multihilo en C, se utiliza la librería ***pthread***, que sigue el estándar POSIX (Portable Operating System Interface).

POSIX es un conjunto de normas que asegura la compatibilidad entre diferentes sistemas operativos, permitiendo que programas escritos para un sistema compatible funcionen en otros sistemas compatibles sin grandes cambios.

La función ***pthread_create()*** permite crear un nuevo hilo de ejecución. Esta función toma cuatro parámetros:

1. **pthread_t *****: Un puntero a una variable que identificará el hilo. La función pthread_create() asignará un valor a esta variable para que luego puedas referenciar el hilo.
2. **pthread_attr_t *****: Atributos de creación del hilo, como su prioridad. Un hilo con mayor prioridad se ejecutará antes que uno con menor prioridad. Puedes pasar NULL para usar los atributos por defecto.
3. **void *(*start_routine)(void *)**: Una función que el hilo ejecutará. Esta función toma un puntero void * como argumento y devuelve un puntero void *. El hilo terminará cuando esta función termine o llame a pthread_exit().
4. **void ***: El argumento que se le pasará a la función mencionada en el parámetro anterior.

La función pthread_create() devuelve 0 si el hilo se crea correctamente, o un número diferente de 0 si ocurre un error.

El código de creación del **hilo** quedaría.

```
void* funcionDelThread (void *);
...
pthread_t idHilo;
...
pthread_create (&idHilo, NULL, funcionDelThread, NULL);
```

Declaración de la función del hilo:

```
void* funcionDelThread(void*);
```

Aquí se declara una función llamada **funcionDelThread** que será ejecutada por el hilo. Esta función debe devolver un puntero void * y aceptar un puntero void * como argumento.

Declaración de la variable identificadora del hilo:

```
pthread_t idHilo;
```

Se declara una variable de tipo **pthread_t** llamada **idHilo**. Esta variable se utilizará para almacenar el identificador del hilo que se va a crear.

Creación del hilo:

```
pthread_create(&idHilo, NULL, funcionDelThread, NULL);
```

La función pthread_create() se utiliza para crear un nuevo hilo. Los parámetros que se le pasan son:

- &**idHilo**: Un puntero a la variable idHilo donde se almacenará el identificador del hilo creado.
- **NULL**: Atributos de creación del hilo. Aquí se pasa NULL para usar los atributos por defecto.
- **funcionDelThread**: La función que el hilo ejecutará. En este caso, funcionDelThread.
- **NULL**: El argumento que se le pasará a funcionDelThread cuando se ejecute en el hilo. Aquí se pasa **NULL** porque no se necesita pasar ningún argumento.

Explicación del código y relación con los parámetros:

1. **pthread_t ***:

```
pthread_t idHilo;
```

```
pthread_create(&idHilo, NULL, funcionDelThread, NULL);
```

pthread_t idHilo: Aquí se declara una variable idHilo de **tipo pthread_t.** Esta variable almacenará el identificador del hilo.

&idHilo: En la llamada a pthread_create(), se pasa la dirección de la variable idHilo. Esto permite que pthread_create() asigne un valor a idHilo para identificar el hilo.

2. **pthread_attr_t *:**

```
pthread_create(&idHilo, NULL, funcionDelThread, NULL);
```

NULL: Este parámetro representa los atributos de creación del hilo. Pasando NULL, se indica que se usarán los atributos por defecto.

3. **void *(*start_routine)(void *):**

```
void *funcionDelThread(void *);

pthread_create(&idHilo, NULL, funcionDelThread, NULL);
```

funcionDelThread: Es una función que será ejecutada por el hilo. La función funcionDelThread toma un puntero void* como argumento y devuelve un puntero void*.

4. **void ***:

```
pthread_create(&idHilo, NULL, funcionDelThread, NULL);
```

NULL: Este es el argumento que se le pasará a funcionDelThread cuando el hilo se ejecute. En este caso, se pasa NULL porque no se necesita pasar ningún argumento a la función.

En resumen, la llamada pthread_create(&idHilo, NULL, funcionDelThread, NULL); crea un nuevo hilo que ejecutará la función funcionDelThread sin ningún argumento adicional, utilizando los atributos por defecto para la creación del hilo, y almacenando el identificador del hilo en la variable idHilo.

En este ejemplo, el hilo se crea y ejecuta la función **funcionDelThread**. No se le pasa ningún argumento a la función del hilo, por lo que se utiliza **NULL** en el último parámetro de **pthread_create().**

Ahora el programa principal y la ***funciónDelThread()*** se estarán ejecutando "simultáneamente".

- ***pthread_join()*** permite que un hilo espere por otro**.**

- ***pthread_mutex_lock()*** permite que dos o más hilos accedan sincronizadamente a un recurso común.

Ejemplo:

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

//Primera funcion donde se imprime el valor de a
void *function1 (void *arg)
{
int a = 10;
int *p = &a;
printf ("El valor de p con function1 es %d \n",*p);
}

//Segunda funcion donde se imprime el valor de a
void *function2 (void *arg)
{
int a=20;
int *p = &a;
printf ("El valor de p con function2 es %d \n",*p);
}


int main()
{
/*Declaracion de hilos h1 y h2 de tipo pthread*/

/*Es necesario guardar el identificador ya que una vez que un hilo comienza a funcionar, la única forma de controlarlo es a través de su identificador*/
pthread_t h1 ;
pthread_t h2 ;
/*Se crean los hilos h1 y h2 y se inicia la ejecución de la función que se le pasa como tercer argumento function1 y function2*/
pthread_create (& h1 , NULL , function1, NULL);
/*La funcion pthread_join permite la ejecucion de los hilos*/
pthread_join(h1,NULL);
pthread_create (& h2 , NULL , function2, NULL);
pthread_join(h2,NULL);
}
```

Explicación del código:

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

Estas cabeceras incluyen funciones estándar de entrada/salida, gestión de memoria, manipulación de cadenas, llamadas al sistema y la librería `pthread` para manejo de hilos.

Definición de `function1` y `function2`:

```
void *function1(void *arg) {
    int a = 10;
    int *p = &a;
    printf("El valor de a con function1 es %d \n", *p);
    return NULL;
}
void *function2(void *arg) {
    int b = 20;
    int *p = &b;
    printf("El valor de a con function2 es %d \n", *p);
    return NULL;
}
```

Ambas funciones inicializan variables locales `a` y `b` y luego imprimen sus valores. Estas funciones no necesitan argumentos externos, por lo que el parámetro `arg` no se utiliza.

Función `main`:

```
int main() {

    pthread_t h1, h2;

    pthread_create(&h1, NULL, function1, NULL);

    pthread_create(&h2, NULL, function2, NULL);

    pthread_join(h1, NULL);

    pthread_join(h2, NULL);

    return 0;

}
```

Declaración de Hilos: Se declaran dos hilos `pthread_t` llamados `h1` y `h2`.

Creación de Hilos:

```
pthread_create(&h1, NULL, function1, NULL);

pthread_create(&h2, NULL, function2, NULL);
```

Se crean dos hilos que ejecutan `function1` y `function2`, respectivamente. No se pasan argumentos adicionales, por lo que se usa `NULL`.

Sincronización de Hilos:

```
pthread_join(h1, NULL);

pthread_join(h2, NULL);
```

`pthread_join` se utiliza para esperar a que los hilos `h1` y `h2` terminen su ejecución antes de que el programa principal continúe. Esto asegura que el programa principal no termine antes que los hilos.

Sincronización de Hilos con Mutex:

Cuando varios hilos necesitan acceder a un recurso compartido, es importante sincronizar el acceso para evitar condiciones de carrera. Esto se puede hacer utilizando un mutex (mutual exclusion).

Ejemplo de uso de `pthread_mutex_lock` y `pthread_mutex_unlock`:

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

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
int contador = 0;

void *incrementarContador(void *arg) {
    for (int i = 0; i < 100000; i++) {
        pthread_mutex_lock(&mutex); // Bloquear el mutex
        contador++;
        pthread_mutex_unlock(&mutex); // Desbloquear el mutex
    }
    pthread_exit(NULL);
}

int main() {
    pthread_t hilo1, hilo2;

    // Crear dos hilos
    pthread_create(&hilo1, NULL, incrementarContador, NULL);
    pthread_create(&hilo2, NULL, incrementarContador, NULL);

    // Esperar a que ambos hilos terminen
    pthread_join(hilo1, NULL);
    pthread_join(hilo2, NULL);

    printf("Valor final del contador: %d\n", contador);
    return 0;
}
```

Explicación del código:

1. **Inicialización del mutex**:

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

Se inicializa un mutex que se utilizará para proteger el acceso al contador.

Uso del mutex en los hilos:

```
pthread_mutex_lock(&mutex); // Bloquear el mutex
contador++;
pthread_mutex_unlock(&mutex); // Desbloquear el mutex
```

- `pthread_mutex_lock(&mutex)`: Bloquea el mutex antes de acceder al contador para asegurar que solo un hilo pueda modificarlo a la vez.
- `pthread_mutex_unlock(&mutex)`: Desbloquea el mutex después de modificar el contador, permitiendo que otros hilos accedan a él.

Creación y unión de hilos:

```
pthread_create(&hilo1, NULL, incrementarContador, NULL);
pthread_create(&hilo2, NULL, incrementarContador, NULL);
pthread_join(hilo1, NULL);
pthread_join(hilo2, NULL);
```

Se crean dos hilos que ejecutan la función `incrementarContador` y se espera a que ambos hilos terminen antes de continuar.

Resumen:

- **pthread_create**: Crea un nuevo hilo.
- **pthread_join**: Espera a que un hilo termine.
- **pthread_mutex_lock / pthread_mutex_unlock**: Sincronizan el acceso a un recurso compartido, asegurando que solo un hilo a la vez pueda modificarlo.

Ejecutar con:
#gcc -pthread código_hilo.c -o código_hilo

3. PROCEDIMIENTO

3.1 Creación de “Hello world” con hilos.

Programa hilo.c que escribe “Hello world” utilizando dos hilos distintos h1 y h2, uno para la palabra Hello y otro para la palabra world.

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


/*Función que imprime Hello*/
void *hello (void *arg)
{
char msg[5] = "Hello";
printf("%s",msg);
}

/*Función que imprime world*/
void *world(void *arg)
{
char msg[5]="world";
printf("%s",msg);
}

/*Funcion main*/
void main()
{
pthread_t h1 ;
pthread_t h2 ;
pthread_create (& h1 , NULL , hello, NULL);
pthread_join(h1,NULL);
printf("\t");
pthread_create (& h2 , NULL , world, NULL);
pthread_join(h2,NULL);
printf ( "\n ");
}
```

Figura 1. Script de creación de un programa que imprime "Hello World" con hilos

3.2 Pasando parámetros a los hilos

El argumento se pasa a la función a través del cuarto parámetro de pthread_create
Si se quiere pasar un parámetro, éste debe ser de tipo puntero a void.

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

void *mensaje(void *arg)
{
        char *msg;
        msg = ( char *) arg;
        printf ("%s",msg) ;
}

void main()
{
        pthread_t h1;
        pthread_t h2;
        char *hello = "Hello";
        char *world = "world";
        pthread_create (& h1 , NULL , mensaje , (void *) hello);
        pthread_join ( h1 , NULL);
        printf("\t");
        pthread_create (& h2 , NULL , mensaje , (void *) world);
        pthread_join ( h2 , NULL);
        printf("\n");
}
```

Figura 2. Script de ejemplo para pasar parámetros a los hilos

3.3 Pasando estructuras a los hilos

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

struct parametros
{
        int id ;
        float escalar ;
        float matriz [3][3] ;
};

void init ( float m [3][3])
{
        int i;
        int j;
        for ( i = 0 ; i < 3 ; i ++)
        {
        for ( j = 0 ; j < 3 ; j ++)
        {
        m[i ][ j ]= random () *100;
        }
        }
}

void * matrizporescalar ( void * arg )
{
        struct parametros * p;
        int i;
        int j;
        p = ( struct parametros *) arg ;
        for ( i = 0 ; i < 3 ; i ++)
        {
        printf (" Este es el hilo %d , multiplicación de la fila %d \n" , p -> id , i );
        for ( j = 0 ; j < 3 ; j ++)
        {
        p -> matriz [ i ][ j] = p -> matriz [ i ][ j] * p -> escalar ;
        usleep (1000) ;
        }
        }
}

int main ( int argc , char argv [])
{
        pthread_t h1 ;
        pthread_t h2 ;
        struct parametros p1 ;
        struct parametros p2 ;
        p1 . id = 1;
        p1 . escalar = 5.0;
        init ( p1 . matriz);
        p2 . id = 2;
        p2 . escalar = 10.0;
        init ( p2 . matriz );
        pthread_create (& h1 , NULL , matrizporescalar , (void *) & p1);
        pthread_join ( h1, NULL );
        pthread_create (& h2 , NULL , matrizporescalar , (void *) & p2);
        pthread_join ( h2 , NULL );
        printf ( " \n ");
}
```

Figura 3. Script de ejemplo para pasar estructuras en los hilos

Terminación del hilo

La terminación del hilo se produce cuando:

- La función que está ejecutando se termina.
- Ejecuta un return.
- Ejecuta la función pthread_exit.

Tiempo de ejecución del hilo

Se debe incluir la librería # include <time.h >
Se pueden usar las funciones gettimeofday() y settimeofday().
El argumento ***t*** es una struct timeval.

```
struct timeval {
        time_t      tv_sec;     /* seconds */
        suseconds_t tv_usec;    /* microseconds */
};
```

3.4 Cálculo de tiempo de ejecución

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

struct timeval tinicio , tfin ;
double promedio = 0.0;

void * hilo ( void * arg )
{
        gettimeofday (& tfin , NULL ) ;
        int tfin_seg= tfin.tv_sec;
        int tini_seg = tinicio.tv_sec;
        promedio += ( tfin_seg - tini_seg);
}
int main ()
{
        int i = 0;
        pthread_t h ;
        for ( i = 0 ; i < 100000 ; i ++ )
        {
        gettimeofday (& tinicio , NULL );
        pthread_create (& h, NULL , hilo , NULL) ;
        pthread_join (h, NULL);
        }
printf ("El tiempo de creación de los %d hilos es %f segundos\n", i , promedio);
}
```

Figura 4. Script de ejemplo para calcular el tiempo de ejecución de un programa

Informe

1. Crear 4 hilos y cada uno que imprima un mensaje diferente.

2. Crear 5 hilos y que cada uno pase un parámetro numérico a una función que devolverá la suma de este número con un valor entero declarado dentro de la función.

3. Utilizando estructuras muestre el resultado del producto de un escalar por valores dentro de una matriz.

4. Escribir código para determinar el resultado del tiempo de ejecución de 1 millón, 2 millones, y 3 millones de hilos. Tomar los tiempos en microsegundos.