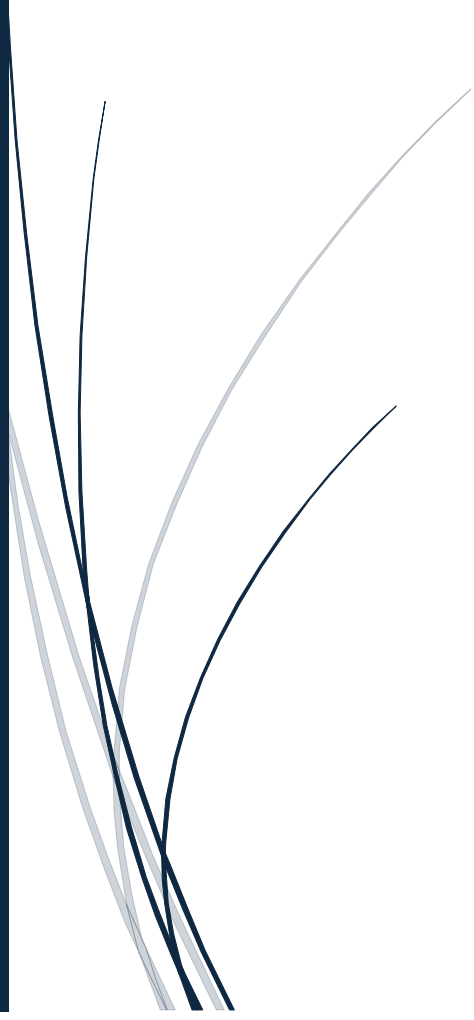
1-5-2024

Hilos POSIX

Práctica 6.2



Julián Blanco González
UNIVERSIDAD DE EXTREMADURA

Índice

1. Introducción

La práctica consiste en conocer como funcionan los hilos POSIX en el lenguaje C. Se hará una definición de que son los hilos, algunas ventajas e inconvenientes de usarlos y las funciones más importantes para poder usar y gestionar estos hilos. Por último, se hacen 4 ejercicios diferentes relacionados con el tratamiento de los hilos.

2. Hilos

Un hilo de ejecución es una forma de dividir el programa principal en dos o más tareas que pueden ejecutarse simultáneamente. Cada hilo en un programa realiza una tarea específica. Todos los hilos de un proceso comparten los recursos del proceso. Residen en el mismo espacio de direcciones y tienen acceso a los mismos datos. Los hilos son subprocesos dentro de un proceso principal (hilo main) y comparten memoria y recursos con otros hilos en el mismo proceso.

Ventajas de usar hilos:

- Eficiencia en el uso de recursos
- Mejora de rendimiento
- Respuesta más rápida

Inconvenientes de usar hilos:

- Dificultades en la sincronización
- Dependencia de plataforma

Teniendo toda esta información, aparecen los hilos POSIX, que son un modelo de "hilos ligeros", lo que significa que permiten la ejecución concurrente dentro de un mismo proceso sin necesidad de asignar recursos adicionales de proceso, como espacio de memoria independiente. Los hilos comparten el espacio de memoria del proceso, archivos abiertos, y otros recursos.

3. Funciones

Crear un hilo

Pthread_create (pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void *), void *arg)

- **pthread_t *thread:** Identificador del hilo creado.
- **const pthread_attr_t *attr:** Pasarle NULL para que coja los valores predeterminados.
- **void *(*start_routine)(void *):** llamar a la función que quieres que ejecute el hilo.
- **void *arg:** parámetro de la función anterior. Si no hay parámetros, NULL.

Funciones para establecer atributos de los hilos

	Nombre Función	Que hace
1	pthread_attr_init(pthread_attr_t *attr)	Inicializa la estructura de atributos de hilo con valores predeterminados
2	pthread_attr_destroy(pthread_attr_t *attr)	Libera todos los recursos asociados con la estructura de atributos de hilo
3	pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate)	Establece el estado de desvinculación del hilo: detachable (no join) o joinable (join si)
4	pthread_attr_getdetachstate(const pthread_attr_t *attr, int *detachstate)	Recupera el estado de desvinculación actual del hilo
5	pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy)	Define la política de planificación para el hilo
6	pthread_attr_getschedpolicy(const pthread_attr_t *attr, int *policy)	Obtiene la política de planificación actual
7	pthread_attr_setschedparam(pthread_attr_t *attr, const struct sched_param *param)	Establece los parámetros de planificación específicos
8	pthread_attr_getschedparam(const pthread_attr_t *attr, struct sched_param *param)	Recupera los parámetros de planificación actuales
9	pthread_attr_setinheritsched(pthread_attr_t *attr, int inherit)	Hilo hereda la política y los parámetros del padre o utiliza unos nuevos.
10	pthread_attr_getinheritsched(const pthread_attr_t *attr, int *inherit)	Obtiene el valor actual del atributo que controla si un hilo heredará la configuración de planificación del hilo padre
11	pthread_attr_setscope(pthread_attr_t *attr, int scope)	Define el ámbito de la competencia del hilo: sistema o proceso.
12	pthread_attr_getscope(const pthread_attr_t *attr, int *scope)	Obtiene el ámbito de competencia actual de los atributos de hilo

13	pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize)	Establece el tamaño de la pila del hilo
14	pthread_attr_getstacksize(const pthread_attr_t *attr, size_t *stacksize)	Obtiene el ámbito de competencia actual de los atributos de hilo

Funciones para el manejo y el control de hilos

	Nombre Función	Que hace
1	void pthread_exit(void *retval)	Termina el hilo llamante y devuelve un valor a través de retval
2	int pthread_join(pthread_t thread, void **retval)	Espera a que termine un hilo específico y recupera el valor de salida del hilo
3	int pthread_detach(pthread_t thread)	Separa el hilo especificado para que sus recursos se liberen automáticamente cuando termine
4	int pthread_cancel(pthread_t thread)	Envía una solicitud de cancelación a un hilo
5	pthread_t pthread_self(void)	Devuelve el identificador del hilo actual.
6	int pthread_equal(pthread_t t1, pthread_t t2)	Compara dos identificadores de hilo
7	pthread_setcancelstate(int state, int *oldstate)	Define el nuevo estado de cancelación para el hilo actual (ejercicio 4)
8	pthread_setcanceltype(int type, int *oldtype)	Define el tipo de cancelación para el hilo actual (ejercicio 4)

* La mayoría de estas funciones devuelven 0 en caso de éxito, y si no, un código de error.

4. Ejemplo

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
void *funcionHilo(void *arg)
{
    printf("Hola desde el hilo %lu!\n", pthread_self());
    return NULL;
}
int main()
{
    pthread_t miHilo;
    pthread_create(&miHilo, NULL, funcionHilo, NULL);
    pthread_join(miHilo, NULL);
    printf("Hilo terminado, fin del programa %d.\n", getpid());
    return 0;
}
```

El código implementa la librería “pthread.h”, que es la que permite el uso de todas las funciones de hilos POSIX. En el main se crea un hilo con pthread_create con los siguientes parámetros:

- **miHilo:** hilo creado con el tipo de variable pthread
- **NULL:** parámetros por defecto del hilo.
- **funcionHilo:** llamada a la función para mostrar un mensaje por pantalla.
- **NULL:** la funciónHilo no tiene argumentos.

Para compilar y ejecutar hay que usar los siguientes comandos:

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S7$ gcc -o pruebaHilos hilos_1.c
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S7$ ./pruebaHilos
```

Compilador gcc para lenguaje c, -o para darle nombre al ejecutable.

Ejecución:

```
Hola desde el hilo 140124379317824!
Hilo terminado, fin del programa 6921.
```

5. Ejercicios

5.1. Ejercicio 1

Pasar un valor de la función del hilo al hilo principal (main).

```c
#include <stdio.h>
#include <time.h>
#include <pthread.h>
#include <unistd.h>
void *funcion_tarea(void *argp)
{
    // TODO: Mostrar el mensaje:
    // “Tarea iniciada. Hilo $ID está corriendo”
    printf("Tarea iniciada. Hilo está corriendo...\n");
    // TODO: Esperar 2 segundos
    sleep(2);
    // TODO: Mostrar el mensaje:
    // “Tarea del hilo $ID va a finalizar ahora”
    printf("Tarea va a finalizar ahora\n");
    // TODO: Usar la función exit de los hilos, devolviendo el ID
    // del hilo (se pasa por parametro de entrada (void*) HiloID)
    pthread_exit((void*)10);

}
int main()
{
    unsigned long i;
    pthread_t tarea;
    // TODO Crear la tarea con pthread_create
    pthread_create(&tarea, NULL, funcion_tarea, NULL);
    // TODO Esperar que la tarea complete y capturar el valor devuelt
    pthread_join(tarea, (void **)&i);
    // Mostrar el valor devuelto por el hilo
    printf("El hilo devolvió: %lu\n", i);
    return 0;
}
```

Para pasar un valor, se usa la función pthread_exit y se le pasa el argumento, el valor. Luego, se recibe en el main, usando la función de espera pthread_join.

Compilar, ejecución y visualización por pantalla

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S7$ gcc -o ej_hilos1 ej_hilos1.c
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S7$ ./ej_hilos1
Tarea iniciada. Hilo está corriendo...
Tarea va a finalizar ahora
El hilo devolvió: 10
```

5.2. Ejercicio 2

Dividir un vector de tamaño 10 en x partes y hacer las sumas parciales. Una vez recibido los resultados, sumarlos todos en el hilo principal, para obtener la suma total.

```c
#include <stdio.h>
#include <time.h>
#include <pthread.h>
#include <unistd.h>
#define tamano 10
#define numHilos 5
int vector [] = {1, 2, 3, 4 , 5, 6, 7 , 8, 9, 10};
int sumaPartes [numHilos];

void *sumaArray(void *hilo_ID)
{
    long tid =(long)hilo_ID;
    int inicio = (int)(tid * tamano / numHilos);
    int final = (int)((tid+1) * tamano / numHilos);
    int suma = 0;

    for(int i=inicio;i<final;i++)
    {
        suma = suma + vector[i];
    }
    sumaPartes[tid] = suma;
    pthread_exit(NULL);
}
int main()
{
    printf("Tarea iniciada. Va a sumarse los valores del array...\n")
    int totalSuma = 0;
    pthread_t hilosVECTOR[numHilos];
    for(long i=0;i<numHilos;i++){
        pthread_create(&hilosVECTOR[i], NULL, sumaArray, (void *)i);
    }

    for(int i=0;i<numHilos;i++){
        pthread_join(hilosVECTOR[i], NULL);
        totalSuma = totalSuma + sumaPartes[i];
        printf("Suma del hilo %d: %d\n", i, sumaPartes[i]);
    }

    printf("Suma total: %d\n", totalSuma);
    return 0;
}
```

En mi caso, he dividido el vector en 5 partes, es decir, cada parte son 2 números. Lo primero es crear un vector de hilos con tamaño 5. Luego, con un for, creo cada hilo para recorrer el vector por partes, pasándole el índice. Dependiendo del índice, en la función cojo un par de valores. Luego, hago la suma parcial, y guardo el rresultado en un array auxiliar. Una vez

que termina cada hilo de hacer la suma parcial, el hilo main, con la función join (espera), va sumando el valor de las sumas parciales y por último, muestro el resultado total.

Compilar, ejecución y visualización por pantalla

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S7$ gcc -o ej_hilo2 ej_hilo2.c
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S7$ ./ej_hilo2
Tarea iniciada. Va a sumarse los valores del array...
Suma del hilo 0: 3
Suma del hilo 1: 7
Suma del hilo 2: 11
Suma del hilo 3: 15
Suma del hilo 4: 19
Suma total: 55
```

5.3. Ejercicio 3

Crear un número x de hilos, cada uno de ellos asociarlo a una operación matemática de la librería “math.h”, darle un rango de valores y mostrar los resultados por pantalla. Esperar en el programa principal a que terminen todos los hilos.

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

#define tamano 10
#define numHilos 4

void *compute_sine(void *threadid)
{
    for (int i = 0; i < numHilos; i++)
    {
        printf("Seno %d: %f\n", i, sin(i));
    }
    pthread_exit(NULL);
}
void *compute_cosine(void *threadid)
{
    for (int i = 0; i < numHilos; i++)
    {
        printf("Coseno %d: %f\n", i, cos(i));
    }
    pthread_exit(NULL);
}
void *compute_log(void *threadid)
{
    for (int i = 1; i <= numHilos; i++)
    { // Log 0 error
        printf("Log %d: %f\n", i, log(i));
    }
    pthread_exit(NULL);
}
```

```c
void *cuadrado(void *threadid)
{
    for (int i = 0; i < numHilos; i++)
    {
        printf("Cuadrado de %d: %f\n", i, pow(i, 2));
    }
    pthread_exit(NULL);
}

int main()
{
    pthread_t hilosVector[numHilos];
    int rc;
    long t;
    int thread_args[numHilos];
    // Escribe tu código para crear los hilos y asignar las funciones

    for(int i = 0; i < numHilos; i++) {
        thread_args[i] = i; // Argumento para cada hilo es su índice
        if (i == 0) {
            rc = pthread_create(&hilosVector[i], NULL, compute_sine, (void *)&thread_args[i]);
        } else if (i == 1) {
            rc = pthread_create(&hilosVector[i], NULL, compute_cosine, (void *)&thread_args[i]);
        } else if (i == 2) {
            rc = pthread_create(&hilosVector[i], NULL, compute_log, (void *)&thread_args[i]);
        } else {
            rc = pthread_create(&hilosVector[i], NULL, cuadrado, (void *)&thread_args[i]);
        }
    }


    for(int i=0;i<numHilos;i++){
        pthread_join(hilosVector[i], NULL);
    }

    printf("Todos los cálculos han sido completados.\n");
    return 0;
}
```

He añadido un hilo nuevo para una operación matemática nueva, que es el cuadrado de cada número. Creo el vector de hilos, y dependiendo del argumento del hilo (i), voy creando los hilos del vector y asignándole una función diferente a cada uno de ellos, pero todos tienen el mismo argumento, para que las operaciones matemáticas, cojan todos los mismos argumentos. Por último, con join, se espera a que terminen todos para finalizar el programa.

Compilar, ejecución y visualización por pantalla

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S7$ gcc -o ej_hilo3 ej_hilo3.c -lm
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S7$ ./ej_hilo3
Coseno 0: 1.000000
Coseno 1: 0.540302
Coseno 2: -0.416147
Coseno 3: -0.989992
Log 1: 0.000000
Log 2: 0.693147
Log 3: 1.098612
Log 4: 1.386294
Cuadrado de 0: 0.000000
Cuadrado de 1: 1.000000
Cuadrado de 2: 4.000000
Cuadrado de 3: 9.000000
Seno 0: 0.000000
Seno 1: 0.841471
Seno 2: 0.909297
Seno 3: 0.141120
Todos los cálculos han sido completados.
```

* Como apunte, al usar la biblioteca “math.h”, hay que usar -lm a la hora de compilar para poder ejecutar el código.

5.4. Ejercicio 4

Cancelar un hilo en diferentes situaciones.

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
void *thread_function(void *arg)
{
    // Variables para establecer el tipo de cancelación y habilitacion
    int type, oldtype;
    int enabling, oldenabling;

    // TODO: Establecer calceltype del hilo
    // 1.1: para que sea asíncrono con PTHREAD_CANCEL_ASYNCHRONOUS
    // 1.2: para que sea deferred con PTHREAD_CANCEL_DEFERRED
    type = PTHREAD_CANCEL_ASYNCHRONOUS;
    pthread_setcanceltype(type, &oldtype);

    // TODO: habilitamos o no la cancelación del hilo
    // 2.1: la habilitamos con PTHREAD_CANCEL_ENABLE
    // 2.2: la deshabilitamos con PTHREAD_CANCEL_DISABLE
    enabling = PTHREAD_CANCEL_ENABLE;
    pthread_setcancelstate(enabling, &oldenabling);

    printf("Hilo configurado con tipo %d y habilitacion %d\n",
           type, enabling);
    // Bucle infinito para simular un trabajo
    while (1)
    {
        printf("Hilo trabajando...\n");
        sleep(1); // Simulamos un trabajo
    }
    return NULL;
}

int main()
{
    pthread_t thread;
    void *result;
    // TODO CREA UN HILO
    pthread_create(&thread, NULL, thread_function, NULL);

    // Damos tiempo al hilo para que empiece y muestre algunos mensajes
    sleep(3);
    // TODO: Solicitamos la cancelación del hilo
    printf("Cancelando el hilo...\n");
    pthread_cancel(thread);

    // TODO Esperamos a que el hilo responda a la cancelación con join
    pthread_join(thread, &result);
    if (result == PTHREAD_CANCELED)
        printf("El hilo fue cancelado %p\n", result);
    else
        printf("El hilo no fue cancelado correctamente %p\n",
               result);
    return 0;
}
```

Creo un hilo con la función que configurar el tipo de cancelación usando:

- type = PTHREAD_CANCEL_ASYNCHRONOUS;
- pthread_setcanceltype(type, &oldtype);

y luego habilito la cancelación:

- enabling = PTHREAD_CANCEL_ENABLE;
- pthread_setcancelstate(enabling, &oldenabling);

Si se cancela correctamente, el valor devuelto en el join, va a ser igual a PTHREAD_CANCELED, si no, no ha podido ser cancelado.

Compilar, ejecución y visualización por pantalla

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S7$ gcc -o ej_hilo4 ej_hilo4.c
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S7$ ./ej_hilo4
Hilo configurado con tipo 1 y habilitacion 0
Hilo trabajando...
Hilo trabajando...
Hilo trabajando...
Cancelando el hilo...
El hilo fue cancelado 0xffffffffffffffff
```

¿Qué está pasando en cada combinación? ¿En qué se diferencia el comportamiento del hilo?

Para cancelar un hilo depende de 2 configuraciones: el tipo de cancelación (asíncrona o deferred), y habilitar si se cancela o no (enable o disable).

Entonces, hay un total de 4 configuraciones:

- **Asíncrono y habilitado**: El hilo puede ser cancelado en cualquier momento.
- **Asíncrono y deshabilitado**: El hilo no puede ser cancelado, independientemente de si está en un punto de cancelación o no.

  ```
  julian@julian-VirtualBox:~/Escritorio/Distribuidos/S7$ gcc -o ej_hilo4 ej_hilo4.c
  julian@julian-VirtualBox:~/Escritorio/Distribuidos/S7$ ./ej_hilo4
  Hilo configurado con tipo 1 y habilitacion 1
  Hilo trabajando...
  Hilo trabajando...
  Hilo trabajando...
  Cancelando el hilo...
  Hilo trabajando...
  Hilo trabajando...
  Hilo trabajando...
  Hilo trabajando...
  Hilo trabajando...
  ```

- **Diferido y habilitado**: El hilo solo puede ser cancelado en los puntos de cancelación.

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S7$ gcc -o ej_hilo4 ej_hilo4.c
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S7$ ./ej_hilo4
Hilo configurado con tipo 0 y habilitacion 0
Hilo trabajando...
Hilo trabajando...
Hilo trabajando...
Cancelando el hilo...
El hilo fue cancelado 0xffffffffffffffff
```

- **Diferido y deshabilitado**: Al igual que con el asíncrono y deshabilitado, el hilo no puede ser cancelado.

```
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S7$ gcc -o ej_hilo4 ej_hilo4.c
julian@julian-VirtualBox:~/Escritorio/Distribuidos/S7$ ./ej_hilo4
Hilo configurado con tipo 0 y habilitacion 1
Hilo trabajando...
Hilo trabajando...
Hilo trabajando...
Cancelando el hilo...
Hilo trabajando...
Hilo trabajando...
Hilo trabajando...
```

6. Bibliografía

Realización de la práctica

Hilos POSIX.pdf: pdf de la práctica.

Funciones de la librería math.h

https://platzi.com/tutoriales/1968-funciones-c/9103-todas-las-funciones-de-la-biblioteca-mathh-en-c/