Sistemas en Tiempo Real – P0 Posix (parte 1)

Creación de clases para manejar hilos, mutex y variables de condición

En esta práctica, que se va a dividir en dos partes, vamos a definir tres clases que van a permitirnos utilizar los hilos (parte 1) y los mutex y las variables de condición (parte 2) de forma más sencilla usando C++. Es importante realizar correctamente esta práctica ya que estas clases las usaremos (y ampliaremos) en las sucesivas prácticas de Posix. En muchos casos, los métodos de las clases solamente llamarán a una función, pero con el paradigma de programación orientada a objetos, facilitarán la gestión de todos los elementos (hilos, mutex y variables de condición), guardando los datos de cada elemento dentro de la clase, manejando punteros de forma oculta al usuario de las clases y reduciendo los parámetros que se utilizarán en las llamadas a los métodos.

A continuación, se definen las clases que vamos a implementar en la parte 1. Es importante mantener los nombres de los elementos públicos que se indican, así como la estructura de los diferentes métodos (tipo de retorno, número y tipo de parámetros, …), ya que los elementos públicos de una clase son los que se usan en otros códigos (los elementos privados o los nombres de los parámetros de los métodos, al no ser accesibles fuera de la clase, tienen menos restricciones, por ejemplo, a la hora de dar nombres; en cualquier caso, se recomienda darles nombres descriptivos).

- Clase *hilo_t*
  - Atributos privados:
    - Identificador del hilo (entero) → Almacenará un identificador único para cada uno de los hilos.
    - Manejador del hilo (puntero a *pthread_t*) → Almacenará el manejador del hilo que está asociado a la clase.
    - Atributos de creación del hilo (puntero a *pthread_attr_t*) → Almacenará los atributos de creación del hilo asociado a la clase. En las primeras prácticas, se usarán los atributos de creación por defecto, pero en posteriores prácticas le añadiremos funcionalidades a la clase para utilizarlos.
    - Función asociada al hilo (debe seguir el formato *void *(*NombreDelAtributo)(void *)*) → Almacenará la dirección de memoria de la función que se usará para lanzar el hilo.
    - Dato se usará con la función asociada al hilo (puntero a *void*) → Almacenará la dirección de memoria que se usará como parámetro de la función asociada al hilo.
  - Métodos públicos:
    - *hilo_t* (constructor sin parámetros) → Este método reservará memoria para el manejador del hilo y reservará memoria e inicializará los atributos de creación del hilo. Además, obtendrá un identificador mediante el método *ObtenerId()* de la clase *Identificador* (ya implementados) e inicializará a *NULL* tanto la función como el dato asociados al hilo.
    - *~hilo_t* (destructor sin parámetros) → Este método destruirá los atributos de creación del hilo y liberará la memoria de los dos atributos de la clase para los que se reservó memoria.
    - *Lanzar* (retorna un entero y no tiene parámetros) → Este método se encargará de lanzar un hilo usando los atributos almacenados en la clase: el manejador de hilo, los atributos de creación del hilo, la función asociada y el dato que se usará con la función. Deberá comprobar si se ha asociado previamente una función (debe tener valor distinto a *NULL*), en cuyo caso lanzará el hilo y devolverá el valor que retorne la función Posix de creación de hilos. En caso de no existir función asociada, devolverá -1.

- *AsignarFuncion* (no retorna nada y tiene un único parámetro que tiene que seguir el formato *(*Parametro)(void *), donde Parametro es el nombre que se le dará al parámetro de la función) → Esta función se encargará de asignar el parámetro recibido al atributo de clase que almacena la función asociada al hilo.
- *AsignarDato* (no retorna nada y tiene un único parámetro de tipo puntero a *void*) → Esta función se encarga de almacenar el parámetro recibido en el atributo de la clase que almacena el dato que se usará con la función asociada al hilo.
- *AsignarFuncionYDato* → Esta función es una combinación de las dos anteriores (*AsignarFuncion* y *AsignarDato*), por lo que tendrá los parámetros de ambas (primero la función y luego el dato). Simplemente llamará a estas dos funciones con sus respectivos parámetros.
- *Join* (retorna un puntero a *void* y no tiene parámetros) → Este método esperará por el hilo indicado por el manejador de la clase. Además, devolverá un valor de tipo indefinido (por eso retorna un puntero a *void*), que será el obtenido por la función Posix de espera por un hilo.
- *ObtenerManejador* (retorna un *pthread_t* y no tiene parámetros) → Este método devolverá el valor del manejador del hilo asociado a la clase. Aunque el manejador del hilo se almacena como un puntero, este método no devuelve un puntero, ya que no será necesario usarlo de esa forma fuera de la clase. Por tanto, para acceder al contenido del puntero, debemos poner el carácter '*' delante del nombre del manejador, accediendo así al contenido de la dirección de memoria del mismo.
- *ObtenerIdentificador* (retorna un entero y no tiene parámetros) → Este método devolverá el identificador del hilo.

Esta práctica no será puntuable, por lo que no es necesario entregar nada. Como guía de implementación, se dispone, en el Campus Virtual, de una serie de ficheros esqueleto (*ClasesPosix.cpp* y *ClasesPosix.h*) junto con la clase *Identificador* y un programa de prueba para comprobar que se ha implementado correctamente.

Para compilar la práctica se usará el comando *g++ FicherosCpp -o Ejecutable -lpthread*, donde *FicherosCpp* son todos los ficheros de código (los *.cpp* no los *.h*) y *Ejecutable* es el nombre que se dará al fichero ejecutable generado.

Anexo 1 – Clases en C++

- La definición de una clase en C++ (se recomienda hacerla en un fichero .h) se realiza de la siguiente forma:
  class NombreDeClase {
  private:
    //Definición de elementos privados de la clase (solo puede accederse a ellos desde la propia clase)
  Public:
    //Definición de elementos públicos de la clase (puede accederse a ellos desde fuera de la clase)
  }; //La definición de una clase debe acabar con ';'

- Los constructores de la clase siempre deben ser públicos y tendrán el mismo nombre de la clase. Podrán tener o no parámetros, pero no devolverán ningún valor. Cuando se crea una variable de la clase, se llamará de forma automática al constructor correspondiente en función de los parámetros que se pongan.
- El destructor de la clase de siempre debe ser público y tendrá el mismo nombre de la clase precedido del carácter '~'. El destructor no lleva parámetros y se llama de forma automática al destruirse la variable correspondiente.
- Implementación de los métodos de una clase (se recomienda hacerlo en un fichero .cpp que haga un *include* al fichero .h de la definición de la clase): Todos los métodos de la clase, sean públicos o privados, deben ser implementados y, para indicar que son miembros de dicha clase, irán precedidos siempre por el nombre de la clase seguido del carácter ':' dos veces (por ejemplo *NombreDeClase::Metodo1*).
- Para definir una instancia de una clase, se hace como con cualquier otra variable pero pudiendo añadir parámetros para el constructor (es opcional, por eso está puesto entre corchetes): *NombreDeClase InstanciaDeClase[(ParámetrosConstructor)];*. Tras la creación de la instancia, se invocará automáticamente al constructor de la clase, que será aquel cuyos parámetros coincidan con los indicados. Una vez una instancia de clase está definida, puede accederse a cualquier elemento público de la clase de la forma *InstanciaDeClase.Elemento* (o *InstanciaDeClase->Elemento* si la instancia se declaró en forma de puntero).

Anexo 2 – Posix

- Hilos:
  - Lanzamiento de un hilo: *int pthread_create(pthread_t *thread, pthread_attr_t *attr, void *(*start_routine)(void *), void *arg)*, donde *attr* puede ser *NULL* para crear el hilo con los atributos por defecto.
  - Inicializar los parámetros de creación del hilo: *int pthread_attr_init(pthread_attr_t *attr)*, donde *attr* son los atributos de creación del hilo que se desean inicializar.
  - Destruir los parámetros de creación del hilo: *int pthread_attr_destroy(pthread_attr_t *attr)*, donde *attr* son los atributos de creación del hilo que se desean destruir.
  - Formato de la función que se usará para lanzar un hilo: *void *NombreFuncion(void *parámetro)*.
  - Paso de parámetros a un hilo:
    - Parámetro de tipo puntero:

```
void *funcion(void *aux) { //Por ejemplo, el parámetro es un vector de enteros
        int *n=(int *)aux;
 //Resto del código usando n de forma normal
}
```

    - Parámetro de tipo no puntero:

```
void *funcion(void *aux) { //El parámetro es un dato de tipo TDato
        Tdato d=*((TDato *)aux);
 //Resto del código usando d de forma normal
}
```

  - Espera por un hilo: *int pthread_join(pthread_t thread, void **valor_ptr)*, donde *valor_ptr* puede ser *NULL* si no esperamos que el hilo retorne ningún valor.