Sistemas en Tiempo Real – P0 Posix (parte 2)

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 2 (la clase *hilo_t* definida en la parte 1 habrá que añadirla al código de esta práctica). 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 *mutex_t*
  - Atributos privados:
    - Manejador del mutex (puntero a *pthread_mutex_t*) → Almacenará el manejador del mutex que está asociado a la clase.
    - Atributos de creación del mutex (puntero a *pthread_mutexattr_t*) → Almacenará los atributos de creación del mutex 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.
  - Métodos públicos:
    - *mutex_t* (constructor sin parámetros) → Este método reservará memoria para los dos atributos de la clase e inicializará los atributos de creación del mutex.
    - *~mutex_t* (destructor sin parámetros) → Este método destruirá los atributos de creación del mutex y liberará la memoria de los dos atributos de la clase.
    - *Inicializar* (retorna un entero y no tiene parámetros) → Este método será el encargado de crear el mutex de la clase usando los parámetros de creación del mutex. Devolverá el valor que retorne la función Posix de creación de mutex.
    - *Lock* (retorna un entero y no tiene parámetros) → Este método se encargará de acceder al mutex asociado a la clase usando el manejador del mutex de la clase, es decir, si el mutex está abierto, lo cierra y accede a la región crítica, y si está cerrado, se bloquea en el mutex. Devolverá el valor que retorne la función Posix de acceso a un mutex.
    - *Unlock* (retorna un entero y no tiene parámetros) → Este método se encargará de liberar el mutex asociado a la clase usando el manejador del mutex de la clase. Devolverá el valor que retorne la función Posix de desbloqueo de un mutex.
    - *ObtenerManejador* (retorna un puntero a *pthread_mutex_t* y no tiene parámetros) → Este método devolverá el manejador del mutex de la clase. A diferencia del método homónimo la

clase *hilo_t*, aquí se devuelve el puntero y no su contenido, ya que dicho puntero será necesario en las variables de condición.

- Clase *variable_condicion_t*
  - Atributos privados:
    - Manejador de la variable de condición (puntero a *pthread_cond_t*) → Almacenará el manejador de la variable de condición que está asociada a la clase.
    - Atributos de creación de la variable de condición (puntero a *pthread_condattr_t*) → Almacenará los atributos de creación de la variable de condición asociada 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.
    - Mutex asociado a la variable de condición (puntero a *pthread_mutex_t*) → Cuando se usa una variable de condición, esta debe tener un mutex asociado. Este atributo nos permitirá mantener la información de dicho mutex dentro de la clase. A diferencia de los atributos anteriores, no se debe reservar memoria para este atributo, ya que apuntará a un mutex definido fuera de la clase y que ya tiene hecha su reserva de memoria.
  - Métodos públicos:
    - *variable_condicion_t* (constructor sin parámetros) → Este método reservará memoria para los dos primeros atributos de la clase (no para el mutex asociado) e inicializará los atributos de creación de la variable de condición.
    - ~ *variable_condicion_t* (destructor sin parámetros) → Este método destruirá los atributos de creación de la variable de condición y liberará la memoria de los dos primeros atributos de la clase (no para el mutex asociado).
    - *Inicializar* (retorna un entero y no tiene parámetros) → Este método será el encargado de crear de la variable de condición de la clase usando los parámetros de creación de la variable de condición. Devolverá el valor que retorne la función Posix de creación de variables de condición.
    - *Inicializar* (retorna un entero y tiene un parámetro de tipo puntero a *pthread_mutex_t*) → Este método será el encargado de crear de la variable de condición de la clase usando los parámetros de creación de la variable de condición y guardará la dirección de memoria de mutex que llega como parámetro (esta será la dirección de memoria del mutex al que se asociará la variable de condición) al atributo de la clase que se almacenará el mutex asociado a la clase. De esta forma, mediante este método podremos inicializar la variable de condición y asociar un mutex en un único paso. Devolverá el valor que retorne la función Posix de creación de variables de condición.
    - *AsociarManejadorMutex* (tiene un parámetro de tipo puntero a *pthread_mutex_t*) → Este método simplemente guardará la dirección de memoria de mutex que llega como parámetro (esta será la dirección de memoria del mutex al que se asociará la variable de condición) al atributo de la clase que se almacenará el mutex asociado a la clase.
    - *Wait* (retorna un entero y no tiene parámetros) → Este método se encargará de bloquear el hilo invocante en la variable de condición de la clase. Devolverá el valor que retorne la función Posix de bloqueo en una variable de condición.
    - *TimedWait* (retorna un entero y tiene un parámetro de tipo *struct timespec*) → Este método se encargará de bloquear el hilo invocante en la variable de condición de la clase hasta, como máximo, el instante de tiempo indicado por el parámetro de entrada. Devolverá el valor que retorne la función Posix de bloqueo temporalizado en una variable de condición.

- *Signal* (retorna un entero y no tiene parámetros) → Este método se encargará de liberar (al menos) un hilo bloqueado en la variable de condición de la clase. Devolverá el valor que retorne la función Posix de liberación de (al menos) un hilo de una variable de condición.
- *Broadcast* (retorna un entero y no tiene parámetros) → Este método se encargará de liberar todos los hilos bloqueados en la variable de condición de la clase. Devolverá el valor que retorne la función Posix de liberación de todos los hilos de una variable de condición.

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

- Mutex:
  - Creación: *int pthread_mutex_init(pthread_mutex_t *mutex, pthread_mutexattr_t *attr)*, donde *attr* puede ser *NULL* para crear el mutex con los atributos por defecto.
  - Destrucción: *int pthread_mutex_destroy(pthread_mutex_t *mutex)*.
  - Creación de los atributos de creación de un mutex: *int pthread_mutexattr_init(pthread_mutexattr_t *attr)*, donde attr se inicializará a los valores por defecto de creación de un mutex.
  - Destrucción de los atributos de creación de un mutex: *int pthread_mutexattr_destroy (pthread_mutexattr_t *attr)*.
  - Bloqueo de un hilo en un mutex (si está cerrado) o cierre de un mutex (si está abierto): *int pthread_mutex_lock(pthread_mutex_t *mutex)*.
  - Apertura de un mutex cerrado: *int pthread_mutex_unlock(pthread_mutex_t *mutex)*. El hilo invocante debe ser poseedor de *mutex*.

- Variables de condición:
  - Creación: *int pthread_cond_init(pthread_cond_t *cond pthread_condattr_t *attr)*, donde *attr* puede ser *NULL* para crear la variable de condición con los atributos por defecto.
  - Destrucción: *int pthread_cond_destroy(pthread_cond_t *cond)*.
  - Creación de los atributos de creación de una variable de condición: *int pthread_condattr_init (pthread_mutexattr_t *attr)*, donde attr se inicializará a los valores por defecto de creación de una variable de condición.
  - Destrucción de los atributos de creación de una variable de condición: *int pthread_condattr_destroy (pthread_mutexattr_t *attr)*.
  - Bloqueo en una variable de condición: *int pthread_cond_wait(pthread_cond_t *cond pthread_mutex_t *mutex)*. El hilo invocante debe ser poseedor de *mutex*.
  - Bloqueo en una variable de condición hasta un instante máximo: *int pthread_cond_timedwait (pthread_cond_t *cond pthread_mutex_t *mutex, struct timespec abstime)*. El hilo invocante debe ser poseedor de *mutex*.
  - Desbloqueo de, al menos, un hilo bloqueado en una variable de condición: *int pthread_cond_signal(pthread_cond_t *cond)*.
  - Desbloqueo de todos hilos bloqueados en una variable de condición: *int pthread_cond_broadcast (pthread_cond_t *cond)*.