Sistemas en Tiempo Real – Práctica Posix 1

Programación con hebras POSIX

Pila acotada

Se trata de terminar la implementación del paquete de pila acotada de números enteros presentado en el tema 2. Para la realización de la práctica se deberá implementar una clase *PilaAcotada* que constará de los siguientes elementos:

- Elementos privados:
  - Pila de enteros → Se deberá usar la clase estándar de C++ *vector* (ver el anexo 3).
  - Tamaño de la pila → Valor entero que indicará cuántos elementos puede haber como máximo en la pila en cada momento.
  - Mutex → Permitirá a los hilos acceder en exclusión mutua al contenido del objeto. Deberá usarse la clase *mutex_t* definida en la práctica 0.
  - Variable de condición para pila vacía → Permitirá a los hilos consumidores bloquearse cuando la pila esté vacía. Deberá usarse la clase *variable_condicion_t* definida en la práctica 0.
  - Variable de condición para pila llena → Permitirá a los hilos productores bloquearse cuando la pila esté llena. Deberá usarse la clase *variable_condicion_t* definida en la práctica 0.
- Elementos públicos:
  - Método constructor sin parámetros → Este método inicializará el mutex y las variables de condición del objeto y establecerá en 10 el tamaño máximo de la pila.
  - Método constructor con un parámetro → Este método inicializará el mutex y las variables de condición del objeto y establecerá el tamaño máximo de la pila al valor indicado por el parámetro de entrada.
  - Método PilaAniadir → Tendrá un parámetro de tipo entero y se encargará de añadir dicho parámetro a la cima de la pila. Deberá bloquearse en caso de que la pila esté llena.
  - Método PilaSacar → Sacará de la pila al elemento que se encuentre en la cima y lo devolverá a la función invocante.

Se deberá crear un fichero .h con la definición de la clase PilaAcotada, un .cpp con el código de dicha clase y otro .cpp con el programa principal (descargar los esqueletos de la práctica del Campus Virtual). El programa principal deberá crear 5 productores y 5 consumidores, cada uno de los cuales añadirá/extraerá 50 valores enteros a la/de la pila, mostrando dichos valores, así como el identificador del productor/consumidor que realice la tarea. Se deberá usar una estructura que contendrá dos campos: un puntero a *hilo_t* para guardar el manejador de los productores/consumidores y un puntero a *PilaAcotada* que permitirá compartir los datos de la pila entre todos los hilos. Esta estructura se usará como parámetro de entrada de los hilos productores/consumidores.

Deberá garantizarse en todo momento la exclusión mutua.

Los archivos de la práctica se comprimirán en un único archivo comprimido que será entregado a través del Campus Virtual usando la tarea creada para tal efecto.

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 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 – Definición de funciones para crear hilos

- Definición 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 puntero a entero
    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
}
```

Anexo 3 – Clase *vector*

- Definición de un vector de tipo *TipoVector*: *vector<TipoVector> NombreVector*. El vector se creará vacío, es decir, que no contendrá ningún elemento.
- Acceso al elemento *i* del vector: *NombreVector[i]*.
- Borrado de todos los elementos del vector: *NombreVector.clear()*.
- Obtención del número de elementos existentes en el vector: *NombreVector.size()*.
- Adicción de un elemento al final del vector: *NombreVector.push_back(elemento)*.
- Obtención del último elemento del vector (no se elimina, solo se accede a su contenido): *NombreVector.back()*. De forma alternativa, también se puede hacer de la forma *NombreVector[NombreVector.sice()-1)]*.
- Borrado del último elemento del vector (no se devuelve el elemento, solo lo elimina): *NombreVector.pop_back()*.