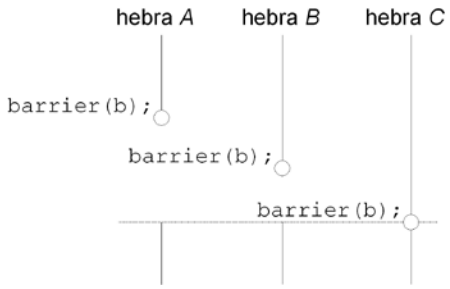
Sistemas en Tiempo Real – Práctica Posix 2

Ejercicio 1. Barrera

Vamos a diseñar un objeto de sincronización denominado barrera. Las hebras que utilizan una barrera dada, forman un conjunto preestablecido de N elementos, donde N es el parámetro del método de creación de la barrera. En la figura tenemos un ejemplo con N = 3. Todas las hebras van bloqueándose en la barrera, buscando un instante de reanudación común. Este llega cuando la hebra N-ésima del conjunto invoca la primitiva.



Para la realización de la práctica se deberá implementar una clase Barrera que constará de los siguientes elementos:

- Elementos privados:
  - Número de hilos que se bloquearán en la barrera (entero).
  - Número de hilos actualmente bloqueados en la barrera (entero).
  - Mutex → Permitirá a los hilos acceder en exclusión mutua al contenido del objeto (usar la clase *mutex_t* definida en la Práctica 0).
  - Variable de condición → Permitirá a los hilos bloquearse en la barrera cuando no hayan llegado todos (usar la clase *variable_condicion_t* definida en la Práctica 0).
- Elementos públicos:
  - Método constructor sin parámetros → Este método inicializará el mutex y la variable de condición del objeto y establecerá en 10 el número de hilos que se bloquearán en la barrera.
  - Método constructor con un parámetro → Este método inicializará el mutex y la variable de condición del objeto y establecerá el número de hilos que se bloquearán en la barrera al valor indicado por el parámetro de entrada.
  - Método Sincronizar → Si la hebra invocante no es la N-ésima, se bloquea, si es la N-ésima, libera al resto, resetea la barrera (esto permite poder volver a utilizarla posteriormente) y continúa la ejecución. En este caso, como la condición de bloqueo solo se debe comprobar una vez, no es necesario usar un bucle *while*, basta con una sentencia *if*.
  - Método Resetear sin parámetros → Si han llegado todos los hilos a la barrera, reinicia a 0 el contador de hilos en la barrera.
  - Método Resetear con un parámetro → Si han llegado todos los hilos a la barrera, reinicia a 0 el contador de hilos en la barrera y establecerá el número de hilos a los que se esperará en la barrera al del parámetro recibido.

Se deberá crear un fichero .h con la definición de la clase Barrera, un .cpp con el código de dicha clase y otro .cpp con el programa principal (descargar esqueletos del Campus Virtual). El programa principal deberá crear tantos hilos como se indiquen por parámetro de la barrera, los cuales se bloquearán en la barrera hasta que lleguen todos a la misma. Se deberá usar una estructura que contendrá dos campos: un puntero a *hilo_t* (para

almacenar la instancia manejadora de cada hilo) y un puntero a Barrera que permitirá compartir los datos de la barrera entre todos los hilos. Esta estructura se usará como parámetro de entrada de los hilos.

Deberá garantizarse en todo momento la exclusión mutua.

Ejercicio 2. Señales

Implementar un programa que cree un hilo que espere la señal SIGINT. Dicho hilo deberá esperar hasta que el número de ocurrencias de la señal sea el indicado como parámetro de ejecución, momento en el que terminará el hilo y, con él, el programa.

Los ejercicios de la práctica se comprimirán en un único archivo 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 – Posix

- 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 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
}
```

- Señales:
  - Todos los hilos (incluido el main) deben bloquear las señales que se van a procesar.
  - Poner un conjunto de señales a vacío: *int sigemptyset (sigset_t *set)*.
  - Añadir una señal a un conjunto de señales: *int sigaddset (sigset_t *set, int sig)*.
  - Establecer el conjunto de señales bloqueadas de un hilo: *int pthread_sigmask(int how, const sigset_t *set, sigset_t *oset)*, donde *how* puede ser *SIG_BLOCK* (añade las señales en *set* a la máscara), *SIG_UNBLOCK* (retira las señales en *set* de la máscara) y *SIG_SETMASK* (establece *set* como la nueva máscara). *oset* devolverá la máscara de señales previa a la modificación (puede ser *NULL* si no queremos guardar este dato).
  - Esperar por la recepción de una señal: *int sigwait(const sigset_t *set, int *sig)*, donde *set* es el conjunto de señales por el que se va a esperar y *sig* devolverá la señal que ha hecho salir de la función *sigwait* (no puede ser *NULL*, por lo que hay que definir una variable para almacenar este dato).