Clases y monitores

Monitor: estructura de datos que se puede usar en diferentes hilos de ejecución ya que sus métodos se ejecutan en exclusión mutua. Los monitores nos servirán como comunicación entre un proceso productor y uno consumidor.

//La semantica SC de la pagina 36 no vale para varios consumidores.

Lock guards

Al usar exclusión mutua (*mutex*) en los métodos de un monitor podemos encontrar el caso de que se produzca una excepción y no se libere el cerrojo, por lo que se queda en una espera indefinida.

Idea: hay un cerrojo para esa clase y cada método tiene un cerrojo local que pide permiso al cerrojo de la clase.

Un lock guard (guarda de cerrojo) es una variable local a cada método que se exporte de la clase. Se declara al inicio del método y en su constructor consigue la exclusión mutua del cerrojo. Como es una variable local, se destruye automáticamente al final del método, y durante su destrucción se llama a `unlock()`.

Se pueden declarar mediante `lock_guard<mutex> nombre(cerr_clase)` o `unique_lock<mutex> nombre(cerr_clase)`. La diferencia es que la primera no es compatible con las variables de condición y la segunda sí.

Monitores SC

Las `variable_condition` (variables de condición) se usan para controlar los lock guards. Contiene una lista de hebras que están bloqueadas y las siguientes operaciones:

- **wait(lock_guard)**: la hebra que lo llama libera el cerrojo y espera. La hebra debe tener el cerrojo al llamar.
- **notify_one()**: despierta a una hebra aleatoria (política equitativa no especificada) que está esperando el cerrojo. La llamada se hace desde otra hebra que no está bloqueada (esperando).
- **notify_all()**: despierta a todas las hebras que están esperando.

Cuando una hebra despierta, si el cerrojo está libre lo coge, sino, se pone en cola del cerrojo.

Barrera simple

Se utiliza para que las hebras se ejecuten de forma "síncrona" esperándose entre sí, se trata de que varias hebras que ejecutan un bucle se esperen al final de cada iteración. Hay una única cola condición.

En este caso, el monitor tendrá un método llamado `cita()`, al que llama una hebra al final de cada iteración. En un contador compartido y utilizando exclusión mutua, se almacena el número de hebras que han llamado al método, y que por tanto han terminado su iteración. Si el contador es menor que el número de hebras, se llama a `wait(guard)` y la hebra espera; si es igual, se llama a `notify_all()` (En el ejemplo se hace un bucle de `num_hebras-1` llamando a `notify_one()` pero en teoría es lo mismo), se pone el contador a 0 y las hebras comienzan la siguiente iteración, saliendo antes del método `cita()`.

Barrera parcial

La idea es la misma que la anterior, solo que agrupamos a las n hebras en grupos de m hebras. Sólo hay una cita activa a la vez, de manera que las hebras que se ejecuten primero llamarán antes a cita y esperarán, la hebra número m desbloqueará a las anteriores. Hasta que no empiecen a salir las hebras de una cita, no pueden llegar nuevas hebras a la cita. La salida de la cita k se puede mezclar con la entrada de la cita *k+1*, ya que en cuanto empiezan a salir el contador se pone a 0.

Solución Prod/Cons

Monitores Señalar y espera urgente

Hay una cola de espera urgente a parte de la normal. Cuando una hebra hace `signal()` sobre una cola, esa hebra pasa a la cola de espera urgente, mientras que se le da el cerrojo a la hebra a la que se hace signal. Cuando una hebra hace `wait()` y deja el cerrojo, se le da el cerrojo a una hebra de la cola de urgentes en política FIFO. Si la cola de urgentes está vacía, entonces se le da a una de cola normal.

Operaciones sobre variables de condición(CondVar)

- **wait()**: la hebra que llama se espera bloqueada hasta que se haga signal(y le toque a ella).
- **signal()**: se libera la hebra que más tiempo lleva en la cola, y la que llama pasa a la cola de urgentes.
- **get_nwt()**: devuelve el nº de hebras esperando

- **empty()**: devuelve `true` si no hay hebras esperando y `false` si hay una o más en espera.

Barrera