Синхронизаторы. Класс Semaphore

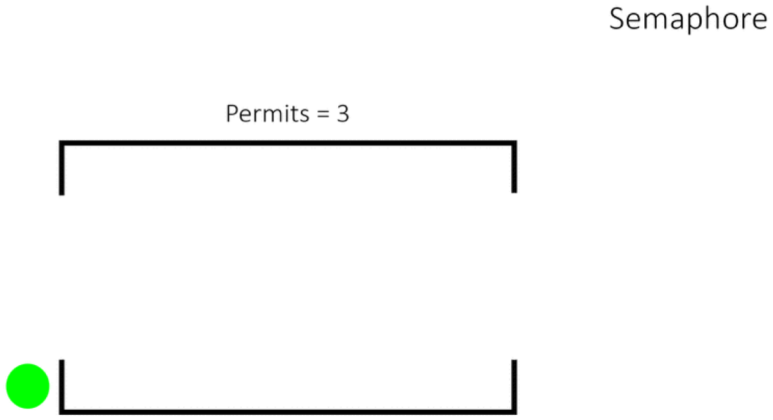
Semaphore управляет доступом к общему ресурсу с помощью счётчика. Если счётчик больше 0, доступ разрешается, после чего значение счётчика уменьшается на 1, а если он равен 0, то в доступе будет отказано. В действительности этот счётчик подсчитывает разрешения, открывающие доступ к общему ресурсу. Как следствие, чтобы получить доступ к ресурсу, поток исполнения должен получить у семафора разрешение на доступ.

Конструкторы Semaphore:

Semaphore(int permits)
Semaphore(int permits, boolean fair)

, где permits - исходное значение счётчика разрешений.

Если присвоить параметру fair значение true, то можно гарантировать, что разрешения будут предоставляться ожидающим потокам исполнения в том порядке, в каком они запрашивали доступ. По умолчанию ожидающим потокам исполнения предоставляется разрешение в неопределенном порядке



Синхронизаторы. Класс Semaphore

Основные методы для работы с Semaphore:

Чтобы получить разрешение, необходимо обратить к методу:

- void acquire() - запрос на единичное разрешение
- void acquire(int permits) - запрос на несколько разрешений

Если разрешение не будет предоставлено во время вызова метода, то исполнение вызывающего потока будет приостановлено до тех пор, пока не будет получено разрешение.

Чтобы освободить разрешение, необходимо обратиться к методу:

- void release() - запрос на единичное освобождение
- void release(int permits) - запрос на несколько освобождений

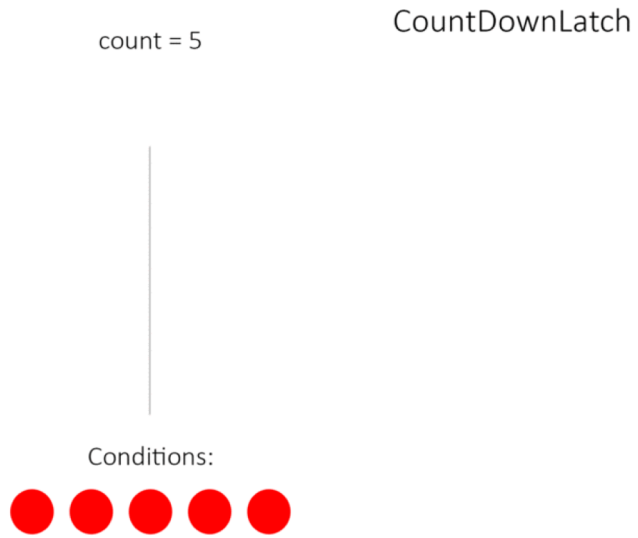
Синхронизаторы. Класс CountDownLatch (замок с обратным отсчётом)

Иногда необходимо, чтобы потоки находились в стадии ожидания до тех пор, пока не будут произведены какие-либо действия (например, предобработка/препроцессинг данных). Для этих целей используется CountDownLatch класс, реализующий самоблокировку с обратным отсчётом.

Конструктор CountDownLatch:

```
CountDownLatch(int count)
```

, где count - количество операций, которое должно быть завершено, чтобы данный механизм позволил продолжать работать временно заблокированным потокам. Когда счётчик равняется 0, все потоки в статусе ожидания “размораживаются” и продолжают свою работу



Синхронизаторы. Класс CountDownLatch (замок с обратным отсчётом)

основные методы для работы с CountDownLatch:

для ожидания по самоблокировке:

- void await() - ожидание длится до тех пор, пока отсчёт, связанный с вызывающим объектом типа CountDownLatch не достигнет 0

- boolean await(long timeout, TimeUnit unit) - ожидание длится только в течение определённого периода времени, определяемого параметром timeout, единицы измерения обозначаются через параметр unit. Данный метод возвращает false (если достигнут предел времени ожидания) или true (если обратный отсчёт достигает 0)

Чтобы известить о событии, следует вызвать метод:

- void countDown() - всякий раз, когда вызывается данный метод, отсчёт, связанный с вызывающим объектом уменьшается на 1

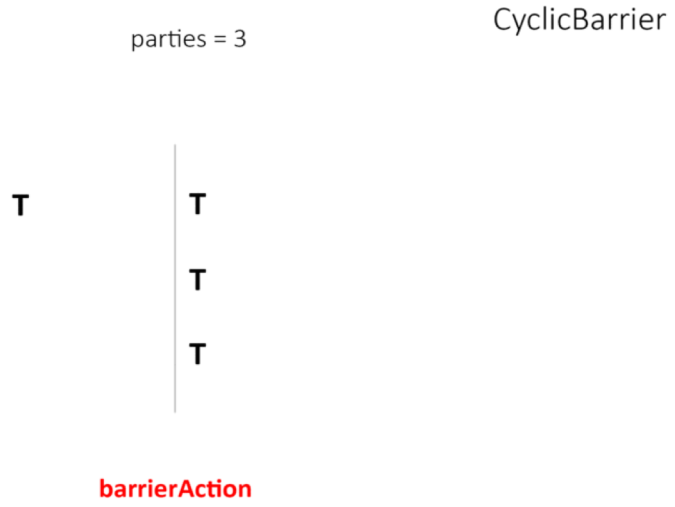
Синхронизаторы. CyclicBarrier

CyclicBarrier - способ работы с несколькими потоками, где группа потоков может быть приостановлена и запущена синхронно в один момент времени. Помимо этого, в момент ожидания группы потоков может быть запущено какое-либо действие (опционально). CyclicBarrier может выглядеть как некая альтернатива методу join, который “собирает” потоки после того, как они выполнились.

Конструктор CyclicBarrier:

CyclicBarrier(int parties)
CyclicBarrier(int parties, Runnable barrierAction)

, где parties - количество потоков, которые хотелось бы обработать перед так называемым барьером



Синхронизаторы. CyclicBarrier

Основные методы для работы с CyclicBarrier:

- int await() - ожидание длится до тех пор, пока каждый поток исполнения не достигнет барьерной точки
- int await(long timeout, TimeUnit unit) - ожидание длится только в течение определённого периода времени, определяемого параметром timeout, единицы измерения обозначаются через параметр unit

Класс CyclicBarrier можно использовать повторно, т.к. он освобождает ожидающие потоки исполнения каждый раз, когда метод await() вызывается из заданного количества потоков исполнения.

Синхронизаторы. Exchanger

Данный способ взаимодействия с потоками может понадобиться тогда, когда необходимо обменяться данными между двумя потоками в определенном месте (например, перед синхронизацией данных с БД или перед стартом какой-либо продолжительной вычислительной операции)

Exchanger является точкой синхронизации пары потоков. Можно вызвать метод exchange(), тем самым заблокировав поток с целью ожидания второго для обмена информацией. В момент, когда второй поток обратиться к методу exchange() произойдёт обмен объектами. Можно передавать null, таким образом можно сконструировать такой сценарий, когда обмен информации будет односторонним (в обратную сторону будет отправлен null).

Exchanger<V>

T

Методы:

- exchange(V x)
- exchange(V x, long timeout, TimeUnit unit)

, где x - ссылка на обмениваемые данные

Синхронизаторы. Phaser

Механизм работы с параллельными потоками Physer похож на CyclicBarrier, но Phaser более гибкий в его применении и настройке.

Особенности Phaser (в сравнении с CyclicBerrier):

- каждый цикл синхронизации (фаза синхронизации) имеет номер
- поток не обязан ожидать, пока все остальные потоки будут в точке синхронизации, чтоб бы продолжить свою работу потоку достаточно сообщить о своём прибытии в точку синхронизации
- у Phaser могут быть наблюдатели (свидетели), которые могут следить за активностью в точке синхронизации
- количество потоков не регламентировано - поток может быть зарегистрирован в качестве потока-участника, а так же может отменять своё участие
- потоку необязательно быть участником Phaser, чтобы ожидать его преодоления
- у Phaser нет опционального действия

Синхронизаторы. Phaser

Конструкторы Phaser:

Phaser() Phaser(int parties)
Phaser(Phaser parent) Phaser(Phaser parent, int parties)

, где parties - определяет количество потоков-участников

Основные методы для работы Phaser:

- int register() - метод для регистрации стороны, вернёт номер регистрируемой фазы
- int getPhase() — возвращает номер текущей фазы
- int arrive() - стороны может вызывать метод arrive() для того, чтобы сообщить о завершении фазы. Данный метод вернет текущий номер фазы. Когда количество достижений конца фазы сравняется с количеством зарегистрированных сторон, фаза завершится и объект класса Phaser перейдёт к следующей фазе (если она есть)
- int arriveAndAwaitAdvance() - указывает что поток завершил выполнение фазы. Поток приостанавливается до момента, пока все остальные стороны не закончат выполнять данную фазу.
- int arriveAndDeregister() — сообщает о завершении всех фаз стороной и снимает ее с регистрации. Возвращает номер текущей фазы
- int awaitAdvance(int phase) — если phase равно номеру текущей фазы, приостанавливает вызвавший его поток до её окончания. В противном случае сразу возвращает аргумент