CSI3131 Operating Systems

Lab 3 – Java Semphores

Java Class Semaphore

- Counting semaphore
- Also :
  - Can choose to use FIFO or not.
  - Semaphore value represents a number of available permits
  - `require()` increments this value – adds permits.
  - `acquire()` blocks if no permit is available.
  - In principle the number of permits is never negative.
    - One exception, it is possible to initialize the number of permits to a negative value.

Semaphore Constructors

`Semaphore(int permits)`

`Semaphore(int permits, boolean fair)`

- The parameter `permits`: defines the initial number of permits.
- The parameter `fair`: If `fair` is `true` the FIFO discipline is used with the semaphore queue.
- The first constructor gives a value of `false` to the parameter `fair`, that is, does not use FIFO with the queue.

The method `acquire()`

- This method plays the same role as the function wait() studied in class.
- Verifies the number of available permits in the semaphore.
  - If a permit is available (semaphore valeu > 0), the number of semaphores is decremented and the method returns; the thread can continue executing.
  - If no permit is available (semaphore value <= 0), the thread blocks and is placed on a queue.
    - When the permit becomes available, a thread in the queue is removed and allowed to consume the permit.

The method `acquire()`

- This method will throw an exception when a thread is interrupted (with the method `interrupt()`).
  - It throws `InterruptedException`.
  - Use the method in a `try/catch` structure:

```
try{sem.acquire();}
catch (InterruptedException e) { }
```

  - Add `break;` as shown below to break out of a loop.

```
try{sem.acquire();}
catch (InterruptedException e) { break; }
```

The method `acquireUninterruptibly()`

- This method provides the same functionality as `acquire()`,
- But it does not throw an exception with the thread is interrupted.
- Not necessary to use the `try/catch` with this method.
- Use the method with threads that are not to be interrupted.

The method `release()`

- This method plays the same role as the function `signal()` studied in class.
  - It increments the number of permits in the semaphore (i.e. its value).

Other methods in Semaphore

- See the Lab 3 document or Java documentation for more details on other methods available in Semaphore.
  - `tryAcquire()`
  - Other versions of `acquire()`, `acquireUninterruptibly()`, `tryAcquire()`, and `release()`
  - `availablePermits()`
  - `drainPermits()` and `reducePermits()`
  - `isFair()`
  - `hasQueuedThreads()`, `getQueueLength()`, `getQueuedThreads()`

Simulating a Ferry

- The Class Ferry
  - For execution in one thread
  - Transports up to 5 vehicles
  - Travels between 2 ports
  - See rules for boarding and ferry crossing
- The Class Auto
  - 10 threads that execute instantiations of this class
  - After crossing to the port with the Ferry, comes back after a while.
- The Class Ambulance
  - One Ambulance thread
  - Works like autos, but has emergency status.
- The Ferry Transports Autos and Ambulance
  - See the lab description for details.
- Your task – to add semaphores in the code template provided so that the threads simulate properly the behaviour of the ferry, the autos, and the ambulance.