CS 511 – Quiz 5: Monitors

1 October 2018

Names: Anthony Rusignuolo, Elliot Wasem
Pledge: I pledge my honor that I have abided by the Stevens Honor System.

Exercise 1

Below is the monitor implementation of a semaphore using the signal and continue strategy (i.e. the one used in Java). Argue why it is not fair to the threads waiting on the condition variable `nonZero`.

```
monitor Semaphore {
  private condition nonZero;
  private int permissions;

  public Semaphore(int n) {
    this.permissions = n;
  }

  public void acquire() {
    while (permissions == 0)
      nonZero.wait();
    permissions--;
  }

  public void release() {
    permissions++;
    nonZero.signal();
  }
}
```

Exercise 2

In an attempt to make it fair, the following implementation is proposed.

```
monitor mySemaphore {
  private condition nonZero;
  private int permissions;

  public Semaphore(int n) {
    this.permissions = n;
  }

  public void acquire() {
1   while (permissions == 0)
2     nonZero.wait();
3   permissions--;
  }

public void release() {
4   if (nonZero.empty())
5     permissions++;
    else
6     nonZero.signal();
  }
}
```

Happy with our implementation, we attempt to use it in our own code. Here we want to ensure that 'A' is always printed before 'B'

```
mySemaphore s = new mySemaphore(0);

thread P: {          thread Q: {
  print('A');          s.acquire();
  s.release();         print('B');
}                    }
```

The problem is that, sometimes, 'B' is never printed at all. Why? Justify your answer by enumerating, starting from 1, the steps that can lead to `B` never being printed.

Exercise 3 (*extra-credit*)

Complete the following code in order to obtain a fair implementation of semaphores. Fill in `?1` and `?2` with an assignment to variable `fromOutside` and `?2` with an appropriate boolean condition.

```
monitor Semaphore {
  private int permissions;
  private int waiting=0;
  private int passedPermissions=0;
  private boolean fromOutside;

  public Semaphore(int n) {
      this.permissions = n;
  }

  public void acquire() {
      ?1
      while (permissions==0 && ?2) {
          waiting++;
          nonZero.wait();
          ?3
          waiting--;
      }
      if (passed_permissions>0) {
           passed_permissions--;
      } else {
          permissions--;
      }
  }

  public void release() {
      if (waiting>0) {
          nonZero.signal();
          passed_permissions++;
      } else {
          permissions++;
      }
  }
}
```

Exercise 1 Ans.
The threads waiting on the condition variable nonZero do not have any priority. Therefore they can lose the competition for scheduling time to newer processes and potentially never execute in the following scenario:
1. With zero permissions available, a new thread T1 enters acquire() and waits on nonZero.
2. A second thread T2 enters release(), incrementing permissions to 1.
3. A third thread T3 enters acquire(), seeing a permission available, and decrementing permissions.
4. T1 is stuck at nonZero.wait().

Exercise 2 Ans.
B is never printed because permissions aren't incremented. Consider the situation that Thread Q is the first to enter the monitor. (See line numbers up top)
(P-, Q1)
(P-, Q2)
(P4, Q2)
(P6, Q2)
(P-, Q2)
(P-, Q1)
Then the cycle repeats, never printing B

Exercise 3 Ans.
?1: fromOutside = true;
?2: fromOutside;
?3: fromOutside = false;