Testing

Samuel Jackson
slj11@aber.ac.uk

May 2, 2013

1 Testing the Schedulers

This document outlines the testing used with the scheduling algorithms in this assignment to ensure correct performance. To test the scheduler implementations a set of JUnit tests have been created and are available in a separate package called "Tests" located with the sources code. The rest of this document provides evidence of carrying out these tests on the algorithms.

1.1 Shortest Remaining Time

Shortest Remaining Time orders jobs according to the amount of processing they still need to do by when a job is added to the queue or and when it is returned to the scheduler after some processing. In the under lying code, both of these methods call the same method ($findPlace$) in order to place a job in the correct place on the queue. In order to test that the scheduler is working correctly I created JUnit tests that added the shortest job first, the shortest job second and two equally sized jobs to the queue. In each of these tests I then check that the correct job is returned from the $getNextJob$ method. I have also provided a test for the $returnJob$. As the code is almost identical underneath the hood as the $addNewJob$ I have only done a single test to check that a job that had a higher time remaining but now has a lower time remaining is moved further up the queue. Below is a screen image showing the the Shortest Remaining Time scheduler passing all JUnit tests.

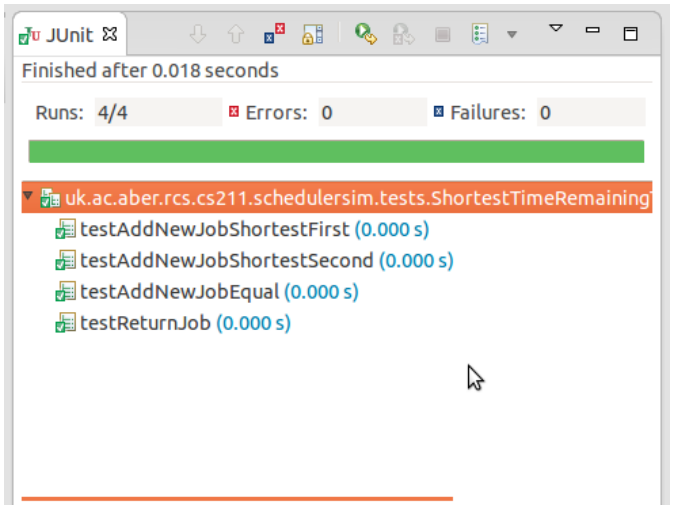

Figure 1: Shortest Remaining Time passing all JUnit tests.

1.2 Round Robin

Round Robin orders jobs by taking the first job in the queue, performing some processing with it and then returning it to the back of the scheduler queue. Round Robin only uses the $returnJob$ method to order its jobs. In order to test that the scheduler works I created three tests each testing the scheduler with one, two and three jobs respectively. The following screen image shows all three of these tests passing.

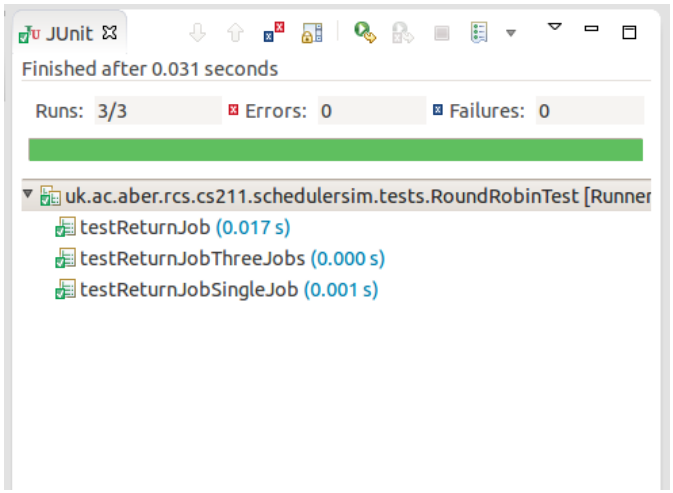

Figure 2: Round Robin passing all JUnit tests.

1.3 Lottery Scheduling

Lottery Scheduling orders jobs according to a weight determined by the job's priority. Because the jobs are selected randomly testing lottery scheduling is more difficult than with the other two algorithms. In order to reliably generate the same sequence of random numbers for every test I used the exact same seed each time when setting up the test. This meant that the scheduler would produce repeatable output which could be measured against an expected criteria. To test this scheduler I carried out similar tests to each of the other algorithms such as testing the scheduler using one, two and three jobs when adding a new job and when returning a job to the scheduler. Below is a screen image showing the lottery scheduler class passing all tests.

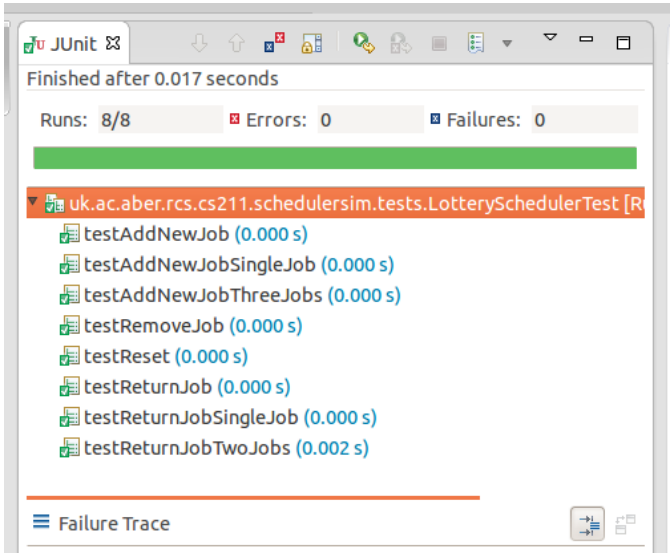

Figure 3: Lottery Scheduling passing all JUnit tests.