Multiprocessor Programming

Final Project Report

Ho-Ren Chuang [horenc@vt.edu](mailto:horenc@vt.edu)

**1. Problem**

**2. Introduction**

**3. Experiments**

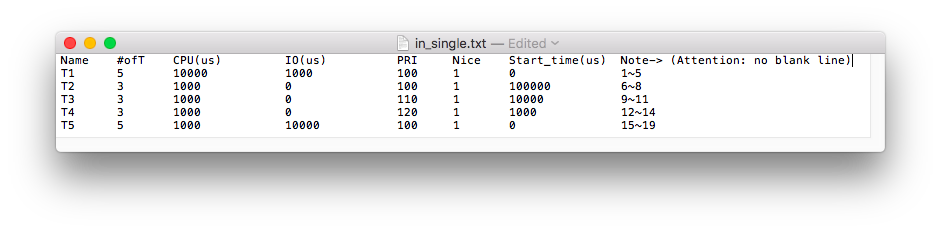
We

**4. Experimental results**

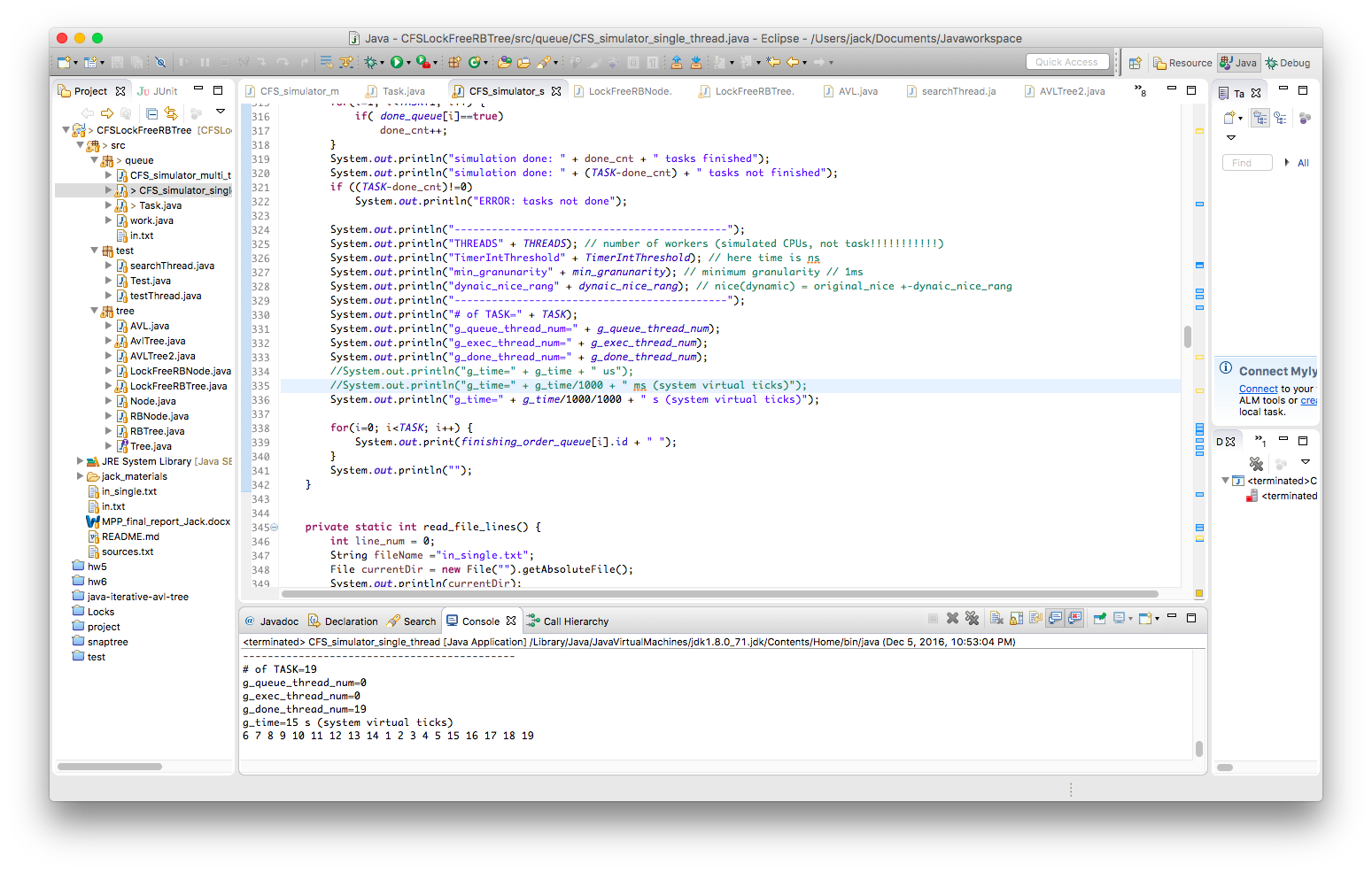
**4.1. CFS part**

**Single-threaded simulator**

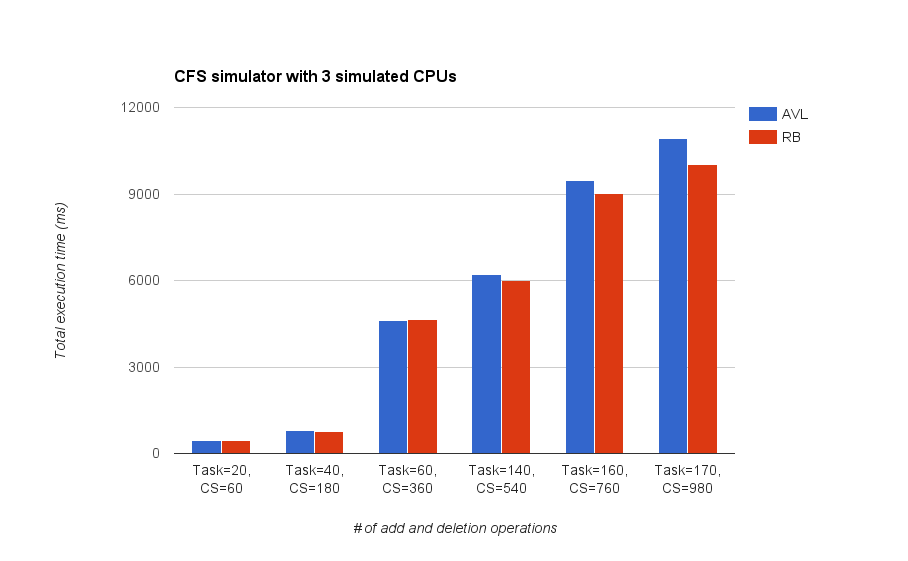
Input task:



Single thread simulator result:

  
Since this is a single task program. As long as the in.txt the same, the output order will be also the same. In this experiment, we want to demonstrate out CFS implementation works correctly. As we can see in

**Multi-threaded simulator**



The result shows that as increase number of Critical Sections (1 CS = 1 add + 1 del operations), RB-tree starts to take advantage of its RB property. I think this is since AVL tree doesn’t use color or marker to make additional information, its rotation operation can be very expensive when the tree is large (complex). Whereas RB-tree can enjoy the benefit of keeping maintaining tree’s color when the tree is large (complex). The RB-tree’s worst case execution time for an insert operation is 2 times rotation and for a deletion operation, 3 times rotation. This merit can help RB-tree outperform AVL-tree while the tree is large (complex).

And during the low workloads, it’s hard to tell which data structure perform better. More importantly, hardly do users feel the differences. We think the reason why Linux chooses RB-tree as queue data structure is that both data structure don’t have much difference during low workload scenario, but RB-tree can significantly outperform AVL-tree while high workload scenario.

**Conclusion**

**Appendix**

1. CFS simulator

**How to run CFS simulator**

CFS simulator single-threaded version:

CFS\_simulator\_single\_thread.java

CFS simulator multi-threaded version:

CFS\_simulator\_multi\_thread.java

**Choose one run\_queue data structure**

**static boolean *IS\_RBTREE* = false; // run with AVLTree**

**static boolean *IS\_RBTREE* = true; // run with RBTree**

**Variables in code**

**static** **int** *THREADS* = 3; // # of workers (simulated CPUs, not task/jobs)

**static** **int** *TimerIntThreshold* = 1000\*1000; // timer interrupt ticks = 1ms

**static** **int** *min\_granunarity* = 1000\*1000; // minimum granularity = 1ms

**static** **int** *dynaic\_nice\_rang* = 5; //nice(dynamic)=original\_nice+-dynaic\_nice\_rang

**Variables in input file**

Assign Tasks (Jobs) for single-thread simulator:

$ vi in\_single.txt

Assign Tasks (Jobs) for multi-threaded simulator:

$ vi in.txt