ECS 36C: Report for Program 4

By Ye Wang & Faustine Yiu

map_tester.cc

In map tester, in order to determine the coverage of the tester, we looked at the functionalities of map: insert one unique key mapped to one value; remove a map from its key; get a map's value by its key; and throwing exceptions of anything otherwise. First, we inserted a series of unique keys mapped to a series of values, with one value per key, and they successfully printed. Next, we removed a map by its key, where it successfully removed the specified key and its value and printed out the rest. Finally, we got (get) a key, and it successfully printed. To test exceptions, we tried inserting a repetitive key with a different value, and it successfully threw a logical error exception and printed that "Key already inserted". We also tried a runtime error exception by getting a key that does not exist, and it printed that "Key doesn't exist" successfully.

multimap_tester.cc

In multimap tester, in order to determine the coverage of the tester, we looked at the functionalities of multimap: insert one unique key mapped to one or more values; remove a value from its key (the removal is FIFO); get a map's value by its key (get the first value associated with the key); and throwing exceptions of anything otherwise. First, we inserted a series of keys mapped to a series of values and inserted more than one value to some specific keys. As a result, we were able to print out unique keys with more than one value, along with the rest. Next, we removed a key with one value to successfully print out the new series of key correctly. Then we removed a key with two values, and it resulted in removing the first value mapped to the key. Finally, we got (get) a key and returned its only value and also another key that returned its first value, when it has one or more values. To test exceptions, we tried getting a key that does not exist and was able to throw a runtime exception and print out the error message "key doesn't exist".

cfs_sched.cc

The logic of the cfs_sched was determined by the necessities of the CFS scheduler: having a timeline to store tasks (values) with its min_vruntime as the key in order to run each task evenly through a method of rotation (loop); printing out the progress at each tick; and recognizing a finished task by terminating each task and eventually terminating the program successfully. Through a class of Tasks and Scheduler to manage data, our choice of data structure resulted in the red-black tree to store the multimaps of min_vruntime as the key and tasks as values. Using the tree's unique characteristics, we are able to easily remove nodes when we no longer need certain min_vruntime keys and add nodes as we increment min_vruntime keys, as well as move the tasks around within the nodes, when necessary.

The choice of the various functions was mainly determined by the CFSloop, or the loop instructions provided in the prompt. We created helper functions that CFSloop could utilize to not have too much of a cluster. In addition, we added print functions to correctly display the progress with its associated ticks. Finally, to catch errors, we made a flag function and returned appropriate error messages. Overall, cfs_sched's logic provides a complete program as requested by the prompt of this assignment.