Ryan Connors and Rulx Sainlus

COMP 399

Assignment 1

**Task 3 Report**

In Task 2, we faced a number of challenges, mostly concerned with setting up the class, which was named FileThread, that we would use as our threads. The first challenge faced was deciding what methods should be located in FileThread. At first the only code we had in the Main class was the code to start new threads for the text files in a directory. (Shown below)

*clearFile*(); //clears data from results.txt  
File[] files = *getFiles*(); //generates the File objects representing the txt files  
Vector<Thread> threads = new Vector<>();  
  
for (File file : files) {  
 Runnable run = new FileThread(file);  
 threads.addElement(new Thread(run));  
}  
  
for (Thread thread : threads) {  
 thread.start();  
}

The reason we used a for-each loop with the *files* array was to make the program more universal; it can be used for any number of text files. A new FileThread is started and added to a vector of Threads for each file that exists in a given directory. We started this way for one simple reason – we didn’t know what we were doing. In our first attempts at completing Task 2, we were also putting a Hashtable *wordCounts* into the FileThread class to add all the words and counts to. Looking back now, this didn’t make sense because each time we created a new FileThread to run, it would have its own *wordCounts* variable in which it added words to. We realized this was an issue because although the program was counting words correctly and writing to an output file, we were finding repeat words which each had their own counts. For example, the 3 files we tested on all contained instances of the word “bed,” file1 had one instance of the word, file2 had two, and file3 had four. A correct result would have written “Bed:7” to the output file, but instead we were getting “Bed:1”, “Bed:2”, and “Bed:4” in various spots in the output file each time we ran the program. We eventually decided to email the professor as neither of us had worked with threads before so we needed some extra guidance, and this turned out to be extremely helpful as we were able to get our program to work correctly within ten minutes of trying the new code.

**Running Time**

When running the program with one text file. The running time was not see to go below 0.075 seconds. This was calculated using System.nanoTime(), which returns a long variable and was used at the start and end of the program to calculate the time difference. The below chart shows how the running time changes based on the number of Threads run. The number of threads in our program is directly correlated to the number of files present in a directory, which was explained above. The program was run 5 times for each number of files (threads). In order to keep the results more uniform, copies of one file (as opposed to files of different lengths) were used as the number of threads increased.