Xuefeng Guan ID: 920016536
Github: XuefengGuan1 CSC415 Operating Systems

Assignment 4 – Word Blast

Description:
This assignment is to use multiple threads to count the word frequency of the text file WarAndPeace.txt.

Approach:
This project was actually not hard on the concept of using pthread and mutexes. Using thread and mutex is quite straightforward, and easy to implement. What is making me confused so much was to choose the right data structure to use, and manipulate it.

The first step I did for this assignment was to open the file, and store the content into a local buffer. I calculated how many potential characters each thread will be assigned by getting the total characters of the file, and then divide it by the number of threads. The reason I say "potential characters" is because sometimes a word can be split in half by the threads, I didn't realize this at first, but I did later when seeing that using 1 thread can have different results when using 8 threads.

I created a struct to pass as much data as I want since thread_create only allows me to pass in one argument. The struct has the info for me to split the WarAndPeace.txt into a number of threads parts evenly. I check if the last word array every thread has is either a space or not, I also allow extra space for each buffer in case more words are needed.

Next is tokenization. I used strtok_r because it's a thread safe function unlike strtok, which is a non thread safe function. I then pass the tokens into the table I created.

Everything until data structure was not that hard. For data structure, I used a hashtable, and this is where it starts to confuse me. C doesn't have a built-in hashtable function, so I had to implement one myself. I implemented a simple one using an array. After spending time to figure out how the hash table uses "hash number" to insert the arrays and how linear probing works and implementing all these. Each thread has its only local hashtable, because the first time I implemented this, I used a public static hash table to store the data, which caused a lot of "traffic jam". One thread was actually faster than even 8 threads. I realized because a "bottle-neck" situation happened, so I created local hash tables for each thread.

After having local hash tables, I had to merge the table. I used another hash table to store all the merged results. And this time, I used mutex to protect the public static hash table from being changed by multiple threads.

Finally, It's time to sort the hash table. I spent a lot of time thinking about how I can sort the hash table just to realize that it's very hard to sort a hash table. So I stored all the data in the hash table into an array. An array of structs that contains the word name and word count. I first store all the values to the array, then use a built-in quick sort function to sort the array in a descending order. And lastly, print out the results.

Issues and Resolutions:

- My first issue was the use of open, close, pread, and lseek.
- I resolved it by using the man page, it’s quite easy to figure out.

- My second issue was the use of pthreads and mutex.
- I resolved it by finding online examples, and watching the material videos. There were tons of examples, and it was also quite easy to figure out.

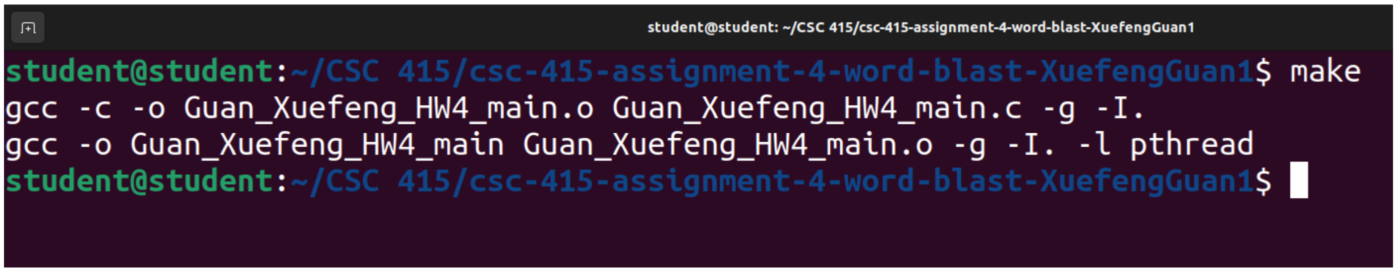
- My third issue was to choose a efficient data structure
- At first I wanted to create a 2D array, but I realized it would be very time consuming to go through the entire array just to store one word. So I looked for something else, I put my eyes on the hash table. I didn’t realize that there are no built-in implementations of hash tables. So I wrote one myself. This took quite some time to get it not running errors.

- My fourth Issue was that I didn’t know how to sort the result.
- I didn’t know how to sort a hash table. But I realized that it’s hard to sort a hash table, so I resolved it by transferring the data in the hash table to an array, and then use qsort(quick sort) to sort the array. And finally print out the results.

- There were some bigger problems I faced, like adding one thread would actually make the run time longer.
- I resolved this by reshaping the structure of my program. I did a lot in this program, including moving the tokenization and buffer split process from the main to the thread function. Remodifying the merge hash table function many times, and so on.

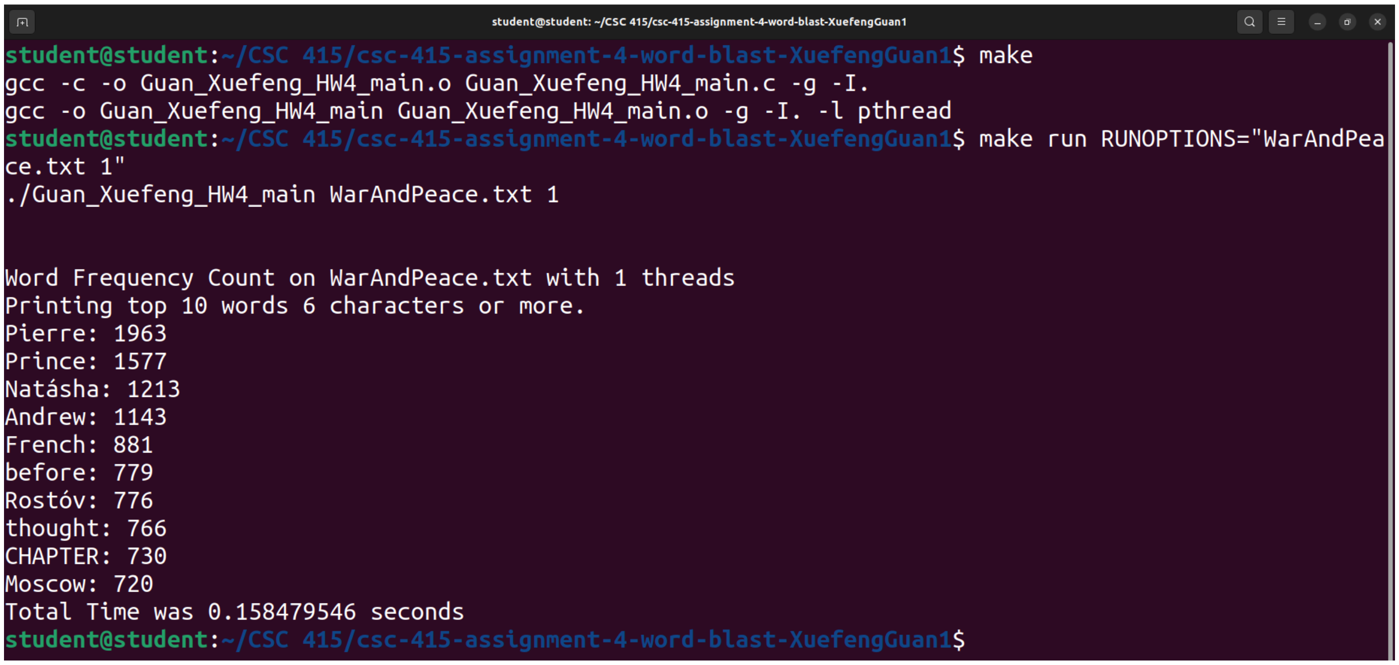
Xuefeng Guan   ID: 920016536
Github: XuefengGuan1   CSC415 Operating Systems

Screen shot of compilation:

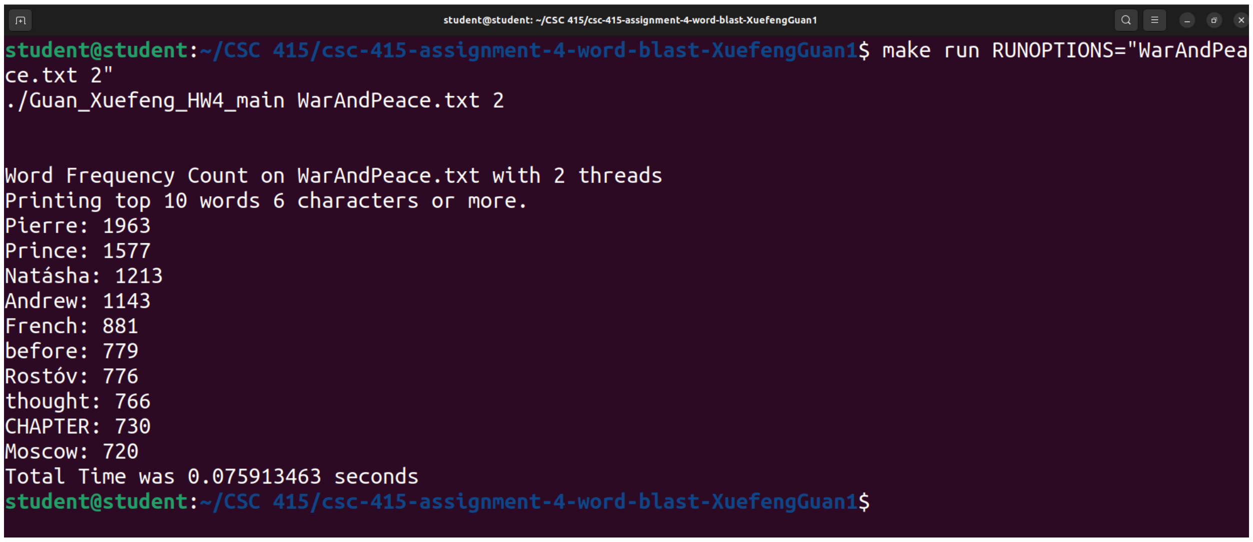


```
student@student: ~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1
student@student:~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1$ make
gcc -c -o Guan_Xuefeng_HW4_main.o Guan_Xuefeng_HW4_main.c -g -I.
gcc -o Guan_Xuefeng_HW4_main Guan_Xuefeng_HW4_main.o -g -I. -l pthread
student@student:~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1$
```

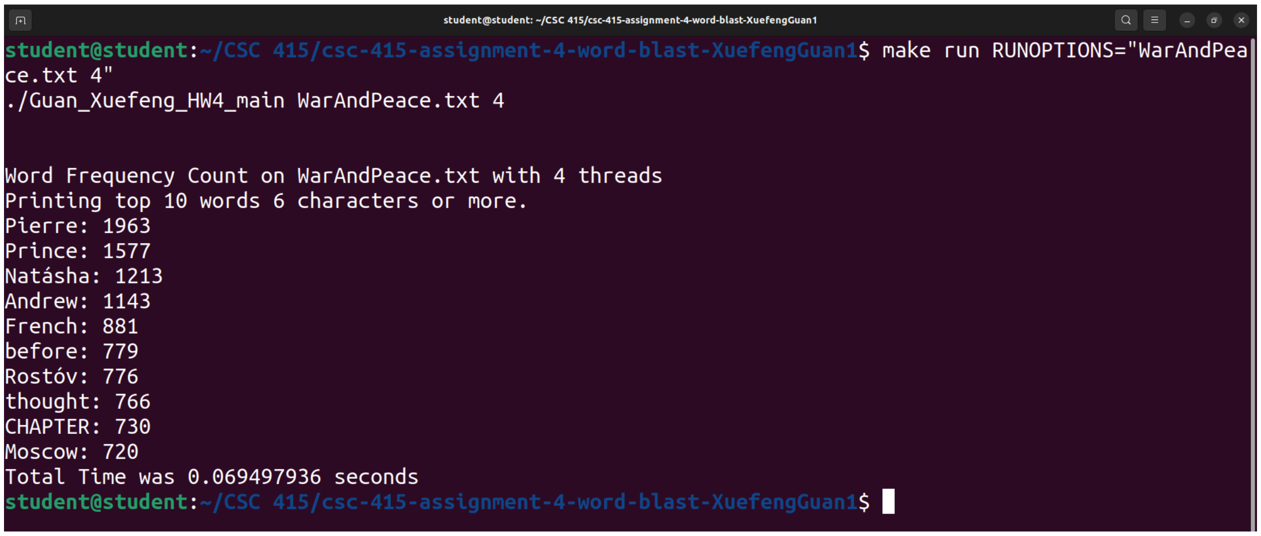
Screen shot(s) of the execution of the program:



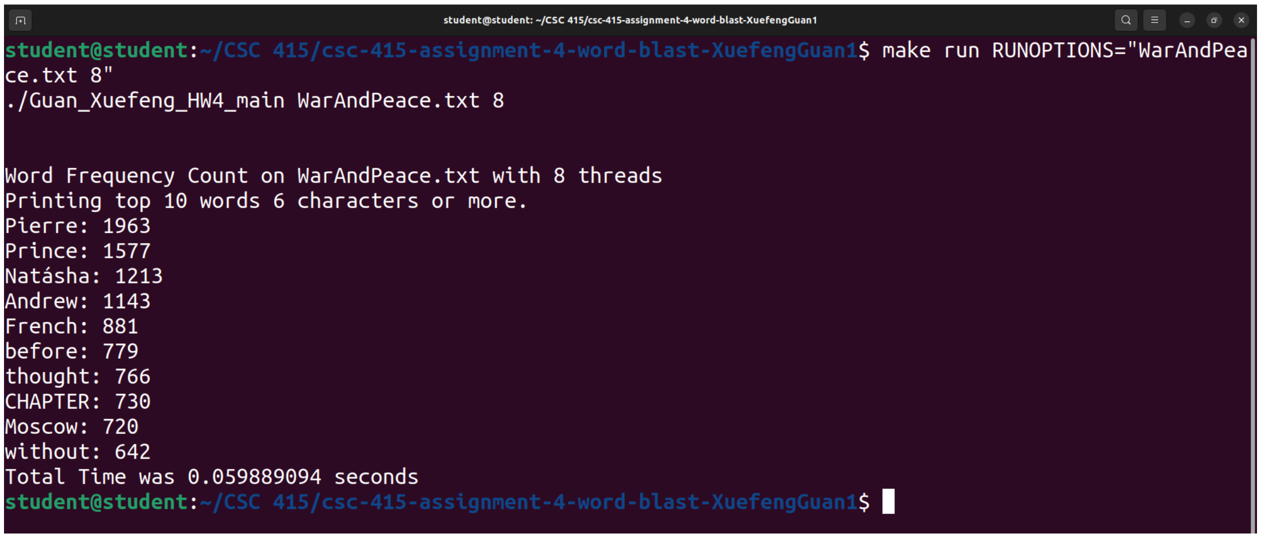
```
student@student: ~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1
student@student:~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1$ make
gcc -c -o Guan_Xuefeng_HW4_main.o Guan_Xuefeng_HW4_main.c -g -I.
gcc -o Guan_Xuefeng_HW4_main Guan_Xuefeng_HW4_main.o -g -I. -l pthread
student@student:~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1$ make run RUNOPTIONS="WarAndPea
ce.txt 1"
./Guan_Xuefeng_HW4_main WarAndPeace.txt 1


Word Frequency Count on WarAndPeace.txt with 1 threads
Printing top 10 words 6 characters or more.
Pierre: 1963
Prince: 1577
Natásha: 1213
Andrew: 1143
French: 881
before: 779
Rostóv: 776
thought: 766
CHAPTER: 730
Moscow: 720
Total Time was 0.158479546 seconds
student@student:~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1$
```



```
student@student: ~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1
student@student:~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1$ make run RUNOPTIONS="WarAndPea
ce.txt 2"
./Guan_Xuefeng_HW4_main WarAndPeace.txt 2


Word Frequency Count on WarAndPeace.txt with 2 threads
Printing top 10 words 6 characters or more.
Pierre: 1963
Prince: 1577
Natásha: 1213
Andrew: 1143
French: 881
before: 779
Rostóv: 776
thought: 766
CHAPTER: 730
Moscow: 720
Total Time was 0.075913463 seconds
student@student:~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1$
```

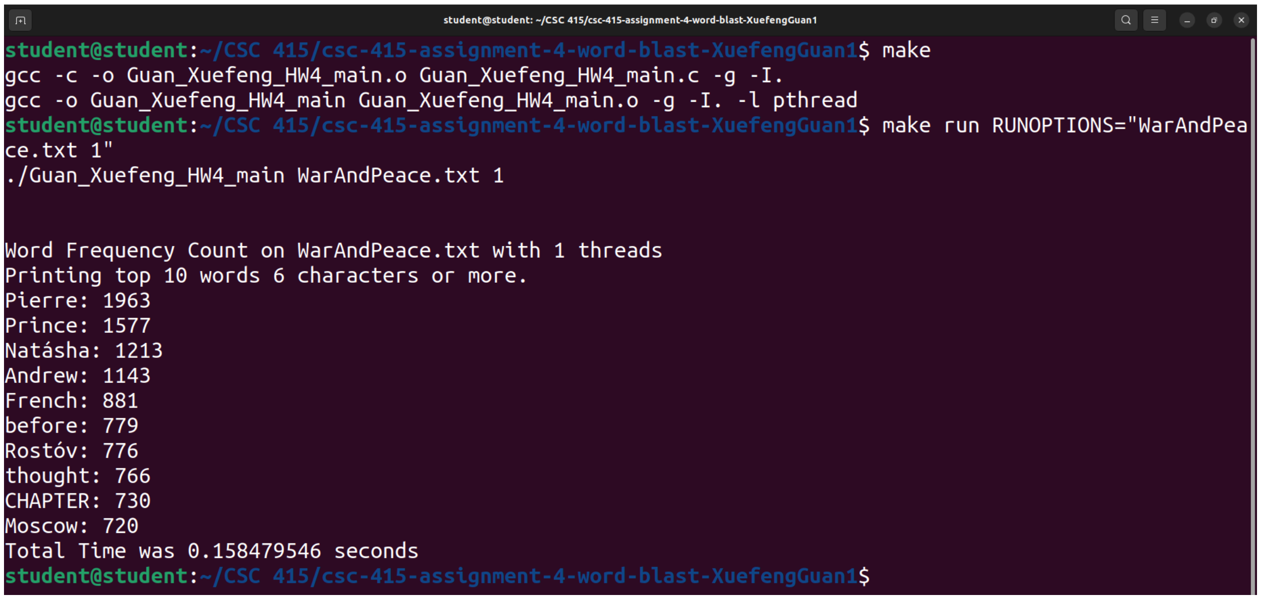


```
student@student: ~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1
student@student:~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1$ make run RUNOPTIONS="WarAndPea
ce.txt 4"
./Guan_Xuefeng_HW4_main WarAndPeace.txt 4


Word Frequency Count on WarAndPeace.txt with 4 threads
Printing top 10 words 6 characters or more.
Pierre: 1963
Prince: 1577
Natásha: 1213
Andrew: 1143
French: 881
before: 779
Rostóv: 776
thought: 766
CHAPTER: 730
Moscow: 720
Total Time was 0.069497936 seconds
student@student:~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1$
```



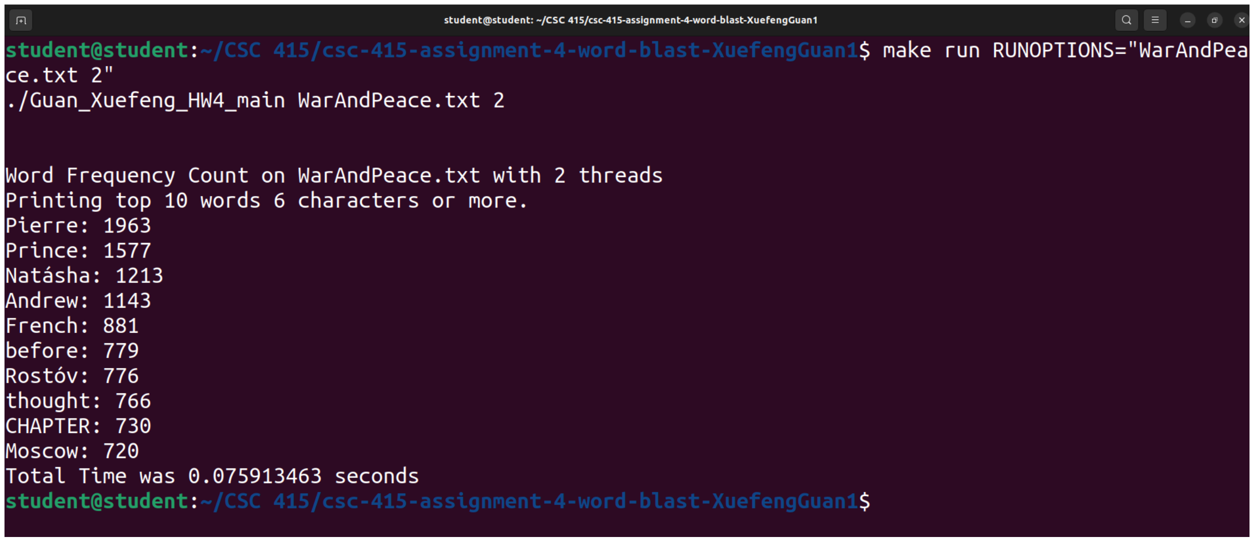
```
student@student: ~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1
student@student:~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1$ make run RUNOPTIONS="WarAndPea
ce.txt 8"
./Guan_Xuefeng_HW4_main WarAndPeace.txt 8


Word Frequency Count on WarAndPeace.txt with 8 threads
Printing top 10 words 6 characters or more.
Pierre: 1963
Prince: 1577
Natásha: 1213
Andrew: 1143
French: 881
before: 779
thought: 766
CHAPTER: 730
Moscow: 720
without: 642
Total Time was 0.059889094 seconds
student@student:~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1$
```

Analysis:



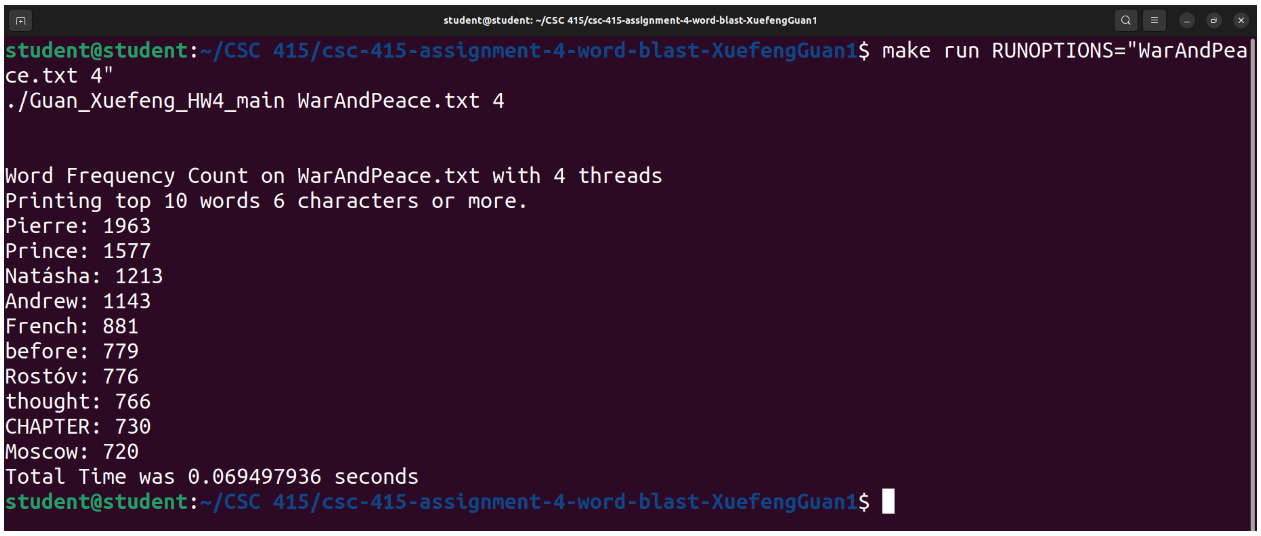
```
student@student:~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1$ make
gcc -c -o Guan_Xuefeng_HW4_main.o Guan_Xuefeng_HW4_main.c -g -I.
gcc -o Guan_Xuefeng_HW4_main Guan_Xuefeng_HW4_main.o -g -I. -l pthread
student@student:~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1$ make run RUNOPTIONS="WarAndPea
ce.txt 1"
./Guan_Xuefeng_HW4_main WarAndPeace.txt 1


Word Frequency Count on WarAndPeace.txt with 1 threads
Printing top 10 words 6 characters or more.
Pierre: 1963
Prince: 1577
Natásha: 1213
Andrew: 1143
French: 881
before: 779
Rostóv: 776
thought: 766
CHAPTER: 730
Moscow: 720
Total Time was 0.158479546 seconds
student@student:~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1$
```

Above is when my program is running on a single thread. The total run time is about 0.158 seconds. There isn't much to talk about, since my program just use a single thread and process the text file with no multi-threads involved.

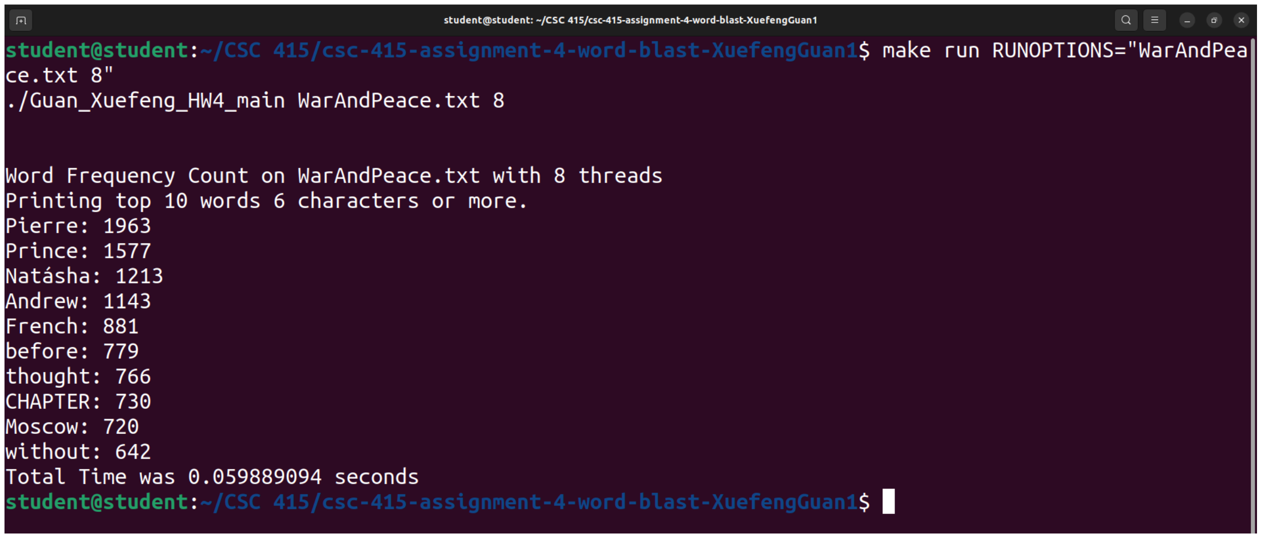


```
student@student:~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1$ make run RUNOPTIONS="WarAndPea
ce.txt 2"
./Guan_Xuefeng_HW4_main WarAndPeace.txt 2


Word Frequency Count on WarAndPeace.txt with 2 threads
Printing top 10 words 6 characters or more.
Pierre: 1963
Prince: 1577
Natásha: 1213
Andrew: 1143
French: 881
before: 779
Rostóv: 776
thought: 766
CHAPTER: 730
Moscow: 720
Total Time was 0.075913463 seconds
student@student:~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1$
```

From here, when the second thread kicks in, my program performance is boosted by a lot. The total run time is 2x faster. I believe this is because there are two concurrent threads running and counting the word frequency at the same time. And this is why I see a big performance boost.


student@student: ~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1

```
student@student:~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1$ make run RUNOPTIONS="WarAndPeace.txt 4"
./Guan_Xuefeng_HW4_main WarAndPeace.txt 4


Word Frequency Count on WarAndPeace.txt with 4 threads
Printing top 10 words 6 characters or more.
Pierre: 1963
Prince: 1577
Natásha: 1213
Andrew: 1143
French: 881
before: 779
Rostóv: 776
thought: 766
CHAPTER: 730
Moscow: 720
Total Time was 0.069497936 seconds
student@student:~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1$ 
```


student@student: ~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1

```
student@student:~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1$ make run RUNOPTIONS="WarAndPeace.txt 8"
./Guan_Xuefeng_HW4_main WarAndPeace.txt 8


Word Frequency Count on WarAndPeace.txt with 8 threads
Printing top 10 words 6 characters or more.
Pierre: 1963
Prince: 1577
Natásha: 1213
Andrew: 1143
French: 881
before: 779
thought: 766
CHAPTER: 730
Moscow: 720
without: 642
Total Time was 0.059889094 seconds
student@student:~/CSC 415/csc-415-assignment-4-word-blast-XuefengGuan1$ 
```

These two screenshots are when 4 and 8 threads are running at the same time. I noticed that the total run time isn't like 2 times faster or 4 times faster than when using 2 threads. 4 threads is only about 0.01 seconds faster than 2 threads, and 8 threads is only about 0.01 seconds faster than 4 threads too. I believe it is due to my program optimization problem. I had to use mutex to prevent multiple threads accessing the critical region. And this creates a bottleneck, the more threads coming in, the more apparent the bottleneck is. And due to this bottleneck of when merging the local hash table to the main hash table. Adding something like 40 threads if there are that many may not create a faster runtime than using only 4 threads. So this is why I believe that adding more threads isn't increasing the performance by that much.

Another thing that might be a factor is that my program isn't increasing performance as thread number goes up by a lot because I have other things in the main function. Even though I have 4 or 8 threads running the frequency check at the same time. The main function, which is single-threaded, still requires the same time to run. Maybe the main function itself takes 0.05 seconds to run, and that causes increasing thread numbers don't make a big performance boost. I tried to make my main function as short as possible, so that it can distribute the works out to the threads.