

Assignment 4 – Word Blast

Description:

This assignment is to write a C program that will read a text file using Linux file functions, count the number of words the contain six characters or more and store them using multithreading, and process and display the top 10 words that appear in the file.

Approach / What I Did:

First step was to outline what the assignment was requiring from the writeup. This resulted in a two page document that will be attached at the end of this document (also uploaded into the Github repo).

Each identified requirement had to be broken down into two parts:

- What was is actually asking us to do
- What steps need to be taken in order to create what we are being asked for
 - This step had most of the meat of the assignment, since we had to break down the idea into smaller steps to actually implement it.

Additional research was required to understand multithreading and apply what we have learned in lecture into code. This broke down into looking into the man pages for almost all the functions used, analyzing returned output from executing these functions, and using said information to debug and solve problems as they surfaced.

One of the confusing parts of the writeup, in my personal opinion, would be the restriction to only use Linux file functions. The confusion came from misunderstanding what that line was actually saying, and once I was able to get past that it was intended to define how we handled the file processing, I was able to continue pushing forward. It did have me constantly checking the man pages for all the functions one more to make sure they were in the standard library though.

Issues and Resolutions:

My first issue was mainly design confusion, as in how to start tackling the assignment due to the fact the instructions are getting less and less obvious in terms of a roadmap. This is a good thing in our development as computer science students, but it is always the hardest problem to overcome for me.

This was solved by starting to understand what is needed to actually implement a multithreaded function, and led to the design for my **runner** function. Writing that out led to identifying certain variables that needed to be declared with a global scope in order to properly use them, which are pointed out with my comments.

My second issue was understanding what 'critical sections' were, and needed to be understood if we were to actually use multithreading appropriately.

This was solved by understanding what mutex locks were, and how to properly use them within the runner function for each thread. Understanding when and where to utilize them helped me understand and avoid possible race conditions that would have led to inconsistency with the results of the program. This also leads to one of the last issues I ran into.

My third issue was returning to basics with sorting algorithms. I spend a good amount of my time attempting to create an algorithm that would only pass through the wordList 10 times to identify the top 10 words, but the nested if statements became more and more convoluted with each pass. It became necessary, and also simpler to sort the list from highest to lowest.

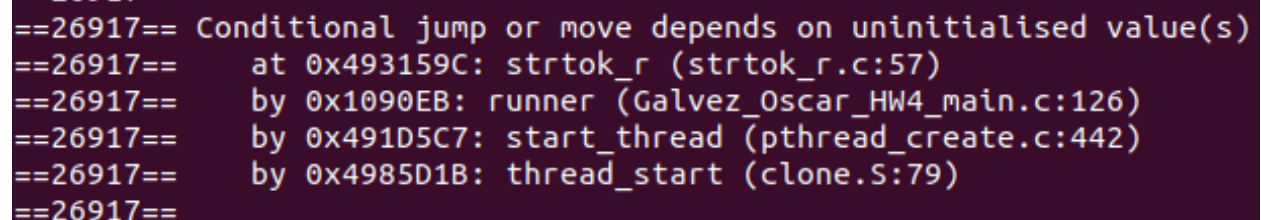
This issue was resolved by experimenting with sorting algorithms, but I ended up settling for a bubble sort. Attempting to implement a merge sort didn't yield good results, although it used similar assignment methods to rearrange the array. The inefficiency and inability to cut the recorded time in half with double the threads is a direct result of sorting this array. Cutting it down would require a multithreaded sorting solution, but I was unable to implement such a solution at the moment.

My final issue was understanding race conditions that could occur. I was attempting to optimize the amount of space required for the buffer by not passing in the left over bytes count (remainingBytes) until the final thread was being declared. After testing, it was discovered that it would not be passed in as planned. If it were, it would be used in the wrong thread.

This issue was resolved by researching race conditions further, and making the decision that the possible memory savings were negligible compared to the perfect environment needed to have it work as planned. Not only that, but being dependent on the result of a race condition goes against the whole point. The decision was made to pass in the remainingBytes variable to all threads to account for the remaining bytes in the file needed for the last thread.

Analysis: (If required for the assignment)

Concerning memory loss in this assignment, all memory has been properly freed. However, most of the errors that occurred from the use of valgrind come from "Conditional jump or move depends on uninitialized value(s)" stemming from the use of strcmp. See photo below



```
==26917== Conditional jump or move depends on uninitialised value(s)
==26917==    at 0x493159C: strtok_r (strtok_r.c:57)
==26917==    by 0x1090EB: runner (Galvez_Oscar_HW4_main.c:126)
==26917==    by 0x491D5C7: start_thread (pthread_create.c:442)
==26917==    by 0x4985D1B: thread_start (clone.S:79)
==26917==
```

Research into the issue reveals that this is understood to be a bug, and is recommended to be suppressed and ignored (as weird as it is to say that).

Compile times: Although there was a difference in time with the use of more threads, it doesn't provide a noticeable difference in performance due to most of the time being taken up by a bubble sort. For the analysis, here are the times if the time were to stop after joining the threads.

1 Thread: 1.145415550 seconds
2 Threads: 0.663885130 seconds
3 Threads: 0.633615916 seconds
4 Threads: 0.606922231 seconds
6 Threads: 0.623812811 seconds

Doubling the threads:

(From 1 to 2) led to a 42% increase in speed.

(From 2 to 4) only led to about a 10% increase in speed, where as the results increasing the threads (From 4 to 6) where negligible, actually increasing the time needed to complete the operation. Small scale example of Amdahl's law, although it may be an incorrect use of it as the time increases.

The only possible way to possibly speed up the sorting process would probably to use a multithreaded sort, or perhaps find a more efficient sorting algorithm. But in doing so, the time wouldn't be dependent on the amount of cores.

Screen shot of compilation:

```
parallels@ubuntu-linux-22-04-desktop:~/Documents/csc415-assignment-4-word-blast-gojilikeog$ make clean
rm *.o Galvez_Oscar_HW4_main
parallels@ubuntu-linux-22-04-desktop:~/Documents/csc415-assignment-4-word-blast-gojilikeog$ make
gcc -c -o Galvez_Oscar_HW4_main.o Galvez_Oscar_HW4_main.c -g -I.
gcc -o Galvez_Oscar_HW4_main Galvez_Oscar_HW4_main.o -g -I. -l pthread
parallels@ubuntu-linux-22-04-desktop:~/Documents/csc415-assignment-4-word-blast-gojilikeog$
```

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

```
parallels@ubuntu-linux-22-04-desktop:~/Documents/csc415-assignment-4-word-blast-gojilikeog$ make run
./Galvez_Oscar_HW4_main WarAndPeace.txt 1

Word Frequency Count on WarAndPeace.txt with 1 threads
Printing top 10 words 6 characters or more.
Number 1 is Pierre with a count of 1963
Number 2 is Prince with a count of 1928
Number 3 is Natásha with a count of 1213
Number 4 is Andrew with a count of 1143
Number 5 is himself with a count of 1020
Number 6 is Princess with a count of 916
Number 7 is French with a count of 881
Number 8 is before with a count of 833
Number 9 is Rostóv with a count of 776
Number 10 is thought with a count of 767
Total Time was 1.241783705 seconds
parallels@ubuntu-linux-22-04-desktop:~/Documents/csc415-assignment-4-word-blast-gojilikeog$
```

```
parallels@ubuntu-linux-22-04-desktop:~/Documents/csc415-assignment-4-word-blast-gojilikeog$ make run
./Galvez_Oscar_HW4_main WarAndPeace.txt 2

Word Frequency Count on WarAndPeace.txt with 2 threads
Printing top 10 words 6 characters or more.
Number 1 is Pierre with a count of 1963
Number 2 is Prince with a count of 1928
Number 3 is Natásha with a count of 1213
Number 4 is Andrew with a count of 1143
Number 5 is himself with a count of 1020
Number 6 is Princess with a count of 916
Number 7 is French with a count of 881
Number 8 is before with a count of 833
Number 9 is Rostóv with a count of 776
Number 10 is thought with a count of 767
Total Time was 0.750555341 seconds
parallels@ubuntu-linux-22-04-desktop:~/Documents/csc415-assignment-4-word-blast-gojilikeog$
```

```
parallels@ubuntu-linux-22-04-desktop:~/Documents/csc415-assignment-4-word-blast-gojilikeog$ make run
./Galvez_Oscar_HW4_main WarAndPeace.txt 4

Word Frequency Count on WarAndPeace.txt with 4 threads
Printing top 10 words 6 characters or more.
Number 1 is Pierre with a count of 1963
Number 2 is Prince with a count of 1928
Number 3 is Natásha with a count of 1212
Number 4 is Andrew with a count of 1143
Number 5 is himself with a count of 1020
Number 6 is Princess with a count of 916
Number 7 is French with a count of 881
Number 8 is before with a count of 833
Number 9 is Rostóv with a count of 776
Number 10 is thought with a count of 767
Total Time was 0.672809417 seconds
parallels@ubuntu-linux-22-04-desktop:~/Documents/csc415-assignment-4-word-blast-gojilikeog$
```

```
parallels@ubuntu-linux-22-04-desktop:~/Documents/csc415-assignment-4-word-blast-gojilikeog$ make run
./Galvez_Oscar_HW4_main WarAndPeace.txt 8

Word Frequency Count on WarAndPeace.txt with 8 threads
Printing top 10 words 6 characters or more.
Number 1 is Pierre with a count of 1963
Number 2 is Prince with a count of 1928
Number 3 is Natásha with a count of 1213
Number 4 is Andrew with a count of 1143
Number 5 is himself with a count of 1020
Number 6 is Princess with a count of 916
Number 7 is French with a count of 881
Number 8 is before with a count of 833
Number 9 is Rostóv with a count of 776
Number 10 is thought with a count of 767
Total Time was 0.713145860 seconds
parallels@ubuntu-linux-22-04-desktop:~/Documents/csc415-assignment-4-word-blast-gojilikeog$
```

Assignment #4 - Word Blast

Goal: Read 'War's Peace' to count and tally each word longer than 6 characters or longer.

Only using Linux file functions (open, close, read, lseek, pread) no additional libraries

But: must do this using threads (This assignment will be using pthreads)

- Each thread will take a chunk of the file and process it, returning its results to the main which tallies.
- The main will print the ten, 6 or more character, words with the highest tallies, from highest to lowest.
- Program should take two parameters - COMPLETE
 - FileName is the name of the file to read - WarAndPeace.txt ✓ Read-Only mode
 - ThreadCount is the number of threads you should spawn to evenly divide the work ✓
- These threads should be launched together in a loop

- * HINTS:
- Do not forget to protect critical sections
 - Make sure to use thread safe library calls
 - You need to know how long the input file is (use lseek)
 - Do not use pipes. You must use mutex locks
 - strtok_r is a tokenizer that is thread safe

- TO DO:
- Accepting parameters - COMPLETE
 - FileName - Read info opening files w/ c ✓
 - ThreadCount as int (accepting argv[2] as ThreadCount ✓
 - Argv is a pointer to a string - used atoi function to convert to int
 - You need to know how long the input file is (use lseek) Determine file size ✓
 - Partition must be determined (Divisor by number of threads) ✓
 - Creating a structure to the top 10 six-character word tally.
 - Option: Create a hashmap to count each word occurrence
 - < key: word, Value: Occurrence Increment >
 - Define the structure - Needs string/char* for word, int for count
 - Threading
 - Thread Pool - Create a number of threads at startup and use as needed? ✓
 - Method or code block must be written that each thread will execute (Instructions or algorithm)
 - pthread_create, pthread_join, pthread_exit?
 - What needs to be used to destroy the threads after use? Mutex-lock & buffers ✓
 - Function must be created to run pthread_create (Code block to execute)
 - Close file - Release allocated memory

• Treanding (Cont.)

- Function must be created to run pthread_create (Code block to execute)
 - There should be a buffer to hold chunks allocated to thread (malloc use) ✓
 - Tokeniser used (strtok_r)
 - Pointer used to point to token

~~Each thread will take a chunk of the file and process it, returning its results to the main which tallies.~~

- ↳
 - Determine the chunk - Requires file size & number of threads ✓
 - Load chunks into a buffer - malloc buffer to load for each thread - using read() ✓
 - Pointer need to indicate position - NOT NEEDED ✓ CHECK AFTER COMPLETION
 - Are there bytes left over? - Test ✓
 - If so, modify buffer size and amount to pass it. Have to allocate for this? ✓
 - Varies per thread count. Allocate memory in buffer to handle leftover for final thread? ✓
- - Check token if it is a six letter word
 - If so, check if it is in data structure - **Compare string function - case sensitive**
 - If so, increment
 - If not, add and increment
 - Continue until token returns null

