

Assignment 4 – Word Blast

Description:

This program is to read a text file and count and sort each of the words that are 6 or more characters long, then will print the top ten most repetitive words. I set up a Marco definition about the number of displays, user can easily change the number of words they want to display.

Approach / What I Did:

1. Git clone the assignment 4 from github
2. Read README.md, understand what we need to do for each step and start working on the assignment
3. Look at the argument by using printf function within the for loop, found that the argument result will return the file name, text file name, number of thread from each command line argument.
4. Initialized an integer to store the value of number of thread(argv[2]), having warning when store argv[2] to thread_num. (Explained in Issues and Resolutions No. 1)
5. Printout the text file's name and the number of thread by calling the command line argument base on the example output display since we've already find that it should be store in the command line argument.
6. Since the README shows the requirement said that we will only can use the Linux file function instead of using the library, which means fopen(), fget() function can not be used anymore. I tried to search how does the open(), close(), lseek() function can do from the Linux manual page online. (Explained in Issues and Resolutions No.2)
7. Get the file descriptor by using open() function, store the file descriptor value into lseek() function to get the file size. (Explained in Issues and Resolutions No.3)

8. Get the total size of the file, then do divided file by thread. Since the divided size should be big enough that can hold all of the bytes in the file and we do know the total file size is 3359549 which can no divisible by 2, 4, 6, 8 (still having remainders), I decided to set the divided size become “total file size / number of thread + 1”.
9. Initialized the mutex by using `pthread_mutex_t` (as global variable), also mutex must be initialized it's default value to ensure that the mutex is in a valid by using `pthread_mutex_init()` before used the mutex. Using if-statement to check if initial mutex completing successfully, if it doesn't return zero, print “error occur” and exit the program.
10. Creating array of the threads since we're going to have multiple threads and the number of threads should depends on the value in the command line arguments (already store the value as an integer named “thead_num”).
11. Begin to create thread process, since we're going to do multiple threads in our program base on the requirements. Decided to use a for loop to create each thread run. Creating independent threads, and each of those will execute a function to read and store the words.
12. Ensure when all thread begin to work and have actually finished, I have to call `pthread_join()` function, otherwise, terminating this program will terminate all the unfinish thread abruptly.
13. Finish creating the thread and use `pthread_join()` function to wait all threads finish before moving to the next step of the main. Then, should be woking on the function we need to execute for each thread.
14. Since in the thread working function, we need to read the file first (conclusion we get before) and get each words in the text file, finding the word if the letters is larger or equal to six, then store all of this words and the time they appears in the whole text file. I decided to use a struct to hold each word within a char variable “word” to store the word and an integer value “count” to calculate the total appear times. Then, I started to think about to make an array that can hold all of the words we want to keep, that's the reason why I initialized a new struct words array, and for the size of the array, it should be big enough that can hold all of the words. Supposed all of the words' length are larger than

six (worth case: 2 threads -> each thread having $3359549/2/2+1 = 839888.25$), I decided the value become the biggest value which is “839889”.

15. Begin to write each thread working function. Initialized a buffer to allocate the memory is big enough to hold each thread working bytes. Also, do free(buff) to de-allocate the memory and set the buff becomes NULL at the end of the function first.
16. Inside of this each thread working function(store_words() function), using read() function that we've already decided before (Explained in Issue and Resolutions No. 2). Since we need to using file descriptor and divided_buffer_size for the read() function, I set up those two values become global variables. (Explained in Issues and Resolutions No.4)
17. Finished the read() function to read the text file and store the all of the data into the buff, then need to figure out how to break each words from the buffer by using Professor provides delimiter. (Explained in Issues and Resolutions No. 5)
18. Get the length of the words we get from the buff (ptr), and check if the length of the word is larger than equal to six. I set up the MAX_CHAR Marco definition become 6, in this case, it will be helpful for users if they want to change the maximum characters of the words they want to display easily in the future.
19. Then, I need to find if this word has already store into the struct since words appears multiple times in the text file. I used a for loop to check each of words in the struct by using strcasecmp() function to compare the words we get from the struct and the word we are store in the ptr. Since we're going to have multiple threads in this program, in case mutex currently locked by another thread, we need to used pthread_mutex_lock() and pthread_mutex_unlock() to lock and unlock the threads. Found error that cannot get any output for my code, and tried to using lseek() function again in main. (Explained in Issues and Resolutions No.6)
20. Using if-statement to check if the mutex lock and mutex unlock success or not, if it failed, print “error occur” and exit the program.
21. Using two if-statement to check the strcasecmp() function return value. (Explained in Issues and Resolutions No. 7)
22. Everything seems done in each thread working function. Run the program and met error (nothing output) and run failed. (Explained in Issues and Resolutions No.8)

23. To do the cleanup: close the file descriptor, deallocate the memory we used in the words_list struct (Explained in Issues and Resolutions No.8), and destroy the mutex thread to make sure it can not be used anymore. Program finished.

Issues and Resolutions:

1. When I initialized the thread_num to become an integer variable, I found that I can not store the value(argv[2]) to thread_num without a warning shows that “-Wint-conversion”, which means the command line argument should be present by a string even though it is a number. We need to convert it to become an integer value that can be used in the future when we try to initialize the size of each thread. That’s the reason why I try to use atoi() function to convert string become an integer, and it can store the value to thread_num without warning.
2. When I was thinking about how to read and get the text file contents only using the Linux file function, I found that the open() function can return the file descriptor of the file and the lseek() function can get the total size of bytes of the file, also the read() function can read the data written on the file. Moreover, the lseek() function requires the descriptor of the file, my idea is that I may use those functions to get the total bytes of the whole file and then we can decide the thread size that we can use later to allocate the memory size for the buffer, then we used in the read() function. Firstly, I looked what kind of value we should use when we are using the read function. I found that read() function requires the char * path to find the file name, that’s why I put the argv[1] here, because I found that the argv[1] presents the file name before, and it also requires an integer value of oflag which is defined in <fcntl.h>. Since we just only need to read the file, so I decided to use O_RDONLY for the oflag value. I also checked what is the open() function run failed, because open() function will return the file descriptor of the file which is a non-negative integer value, so I used an if-statement to check the

return value of the `open()` function (`fd`) is less than zero or not, if it is less than zero, then will print “error occur” and exit the program.

3. Finding the `lseek()` function is used to reposition read/write file offset, and require the value of the file descriptor, offset bytes, and the whence the offset begin. Since I’ve already using the `read()` function find the file descriptor “`fd`” we can use, next I think I have to figure out what is the offset bytes and the “whence” we need to set in this function. As I known, the offset should describe the location of a piece of data compared to another location, and we need to get the total size of the text file. Also, the return value of the `lseek()` function is the resulting offset location as measured in bytes form the beginning of the file. Based on those condition, I decided to use 0 for the offset and the `SEEK_END` for the whence value, because it will moves the point 0 potion back from the end of the file which means the return value for that is the total bytes of the text file we need. Then, I initialized this file size value named `fs` and print out this value to take a look. The return value for it is “3359549” bytes, and I open this text file’s properties to take a look its size, the number of size shows in the property is “3.4M (3,359,549 bytes)”, which is same as what I get from the `lseek()` function.
4. By reading the Linux Programmer’s Manual, I found that when we’re using the `read()` function, we need three values which are file descriptor, a void pointer buff to store the output, and the total bytes size for how large we would like to read. That’s the reason why I decided to move the file descriptor(`fs`) and the total bytes size for `read(devided_buffer_size)` to become global variables to make sure we can store and keep the value for all of this program. I also initialized an integer variable named `read_return_value` to store the return value for the `read()` function when I call it. `Read()` function will return -1 if it is failed, so I wrote an if-statement to check if the return value is equal to -1 or not, if it is equal to negative one which means read failed, program should print “error occur” and exit the program.

5. Since I used `strtok()` function in assignment 3 to break a whole string separated by space, I decided to use `strtok()` function to break each words in buff into a new char pointer variables named ptr and then I can check each words' letters and store it into the struct. I used `strtok()` function to break each words which are in the buff into ptr char variable separated by delimiter which Professor already set at the top of the program by using a while loop. While loop should run until all words store into the ptr, and no more words should be store from the buffer.
6. Since we're using multiple threads for this program, we need to in case what if this mutex currently is locked by another one. `Pthread_mutex_lock()` function will lock simply to ensure an atomic update of the shared variables, such as the word and the count number in the struct. And also, we need to use `pthread_mutex_unlock()` function to unlock mutex after we finish updating the shared variables. I question I meet is I have no idea about where I need to put the mutex lock and mutex unlock in this function. At the first time, I put the mutex lock after the whole while loop, and the mutex unlock before the while loop finish. Even it can work, I found that my run time are more than two minutes for completing all program. I started to think about what kind of values in this program are shared values, maybe I can only set up those variables inside of the lock. The store words function basically is update the words into the struct and the total number of counts in the struct. I tried to put the mutex lock and mutex unlock before and after when I need to update those value. In this case, I need to set up two mutex lock and mutex unlock inside of my program, because I need to check if the word contains in the struct or not first, then decide to count the words appears times plus one or add these words into the struct. It makes make program run correctly and faster than before I change my one mutex lock and mutex unlock becomes two and put them into both if-statement (check if the words are contains in struct or not statement). Since `strcasecmp()` function require the word should be a pointer, I update my word variables in the struct becomes a pointer to make sure it can works correctly.

Another problem I met in this part is that I was trying to printout all of the words (characters are larger than 6) before I put them into the struct. I would like to make sure if I was on the right way. I found that I can not get any output from my code, even though I just want to output all of the words from the text file. I found that maybe I didn't do the correct way to read the whole file at first time. I went back to my main and take a look each step. I found that when I was trying to printout the lseek() return value again after I get the total size of the file, it will return zero instead of return 3359549 bytes (whole file size). I looked the Linux manual again and notice that it might be the reason why I can not get anything from my thread working function. I looked around the lseek() function whence variable in the website and notice that I may need set up the value back after I get the size of the file(SEEK_END), because we're at the end of the file, so we cannot get anything from it. I used SEEK_SET to set it to the beginning of the file and run the program again. Words shows in the output successfully.

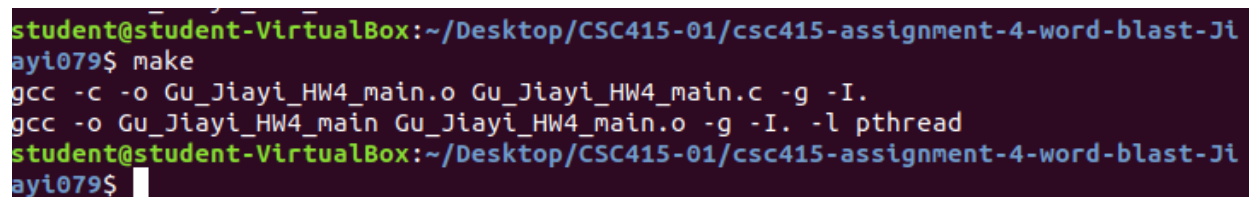
7. At the first time, I was trying to use an if-else statement to check if the word has already contained in the word struct, and I found that it doesn't work because after we check the word is not contains in the struct, we cannot just put it into the struct cause we're inside of a for loop (check EACH word in the struct). If I used if-else statement to do it, it will add each time it the first word in the struct is not those words (ptr) which is wrong. Then, I decided to put one if-statement (if the word is not in the array) outside of the for loop to make sure we've already checked each word in the struct and it doesn't contain in the struct. Then, we can add this word into the struct. In this way, I need to set up two mutex lock and mutex unlock, one should be inside of the for loop to wait count plus one successfully (word already in the struct), another one should be outside of the for loop to wait add this new word to the struct and set the count become 1(word doesn't contain in the struct).
8. I stuck on this error almost one day and can not figure it out. I decided to take look the PowerPoint and the lecture record to find my issue. Then, I found where's the

error is. I have to initialize the word struct to make sure clean up all of the garbage value inside of it. Then I trying to add this part in main (line232 – line239), I set up memory allocate for words_list's word variables, and the size should be the larger size of the word in this words (45 bytes). Then, error fixed. Since we have to de-allocate the memory we've already allocate, I wrote a for loop at the end of the program to set each word in the words_list free and becomes NULL.

Analysis:

These are my output within different numbers of the threads. I though it may faster if I used 8 threads than 2 thread and faster more than $\frac{1}{4}$ of the times, but it seems not. I think it may because I put pthread_create() and pthread_join() inside of one for loop to do it, creating the thread and wait the thread at the same time makes my program doesn't work as what I expect. However, when I tried to put the pthread_join() outside to another for loop (wait until every thread run first), my program will not count all of the words inside of the text file. That might because I set up the ptr variables outside of the mutex lock and mutex unlock function, and it will not wait until other thread finish the shared variables. Then, I try to figure out this issue by moving the mutex lock and unlock outside of the whole while loop. It can work, but the time still doesn't change when I change the number of threads.

Screen shot of compilation:



```
student@student-VirtualBox:~/Desktop/CSC415-01/csc415-assignment-4-word-blast-Jiayi079$ make
gcc -c -o Gu_Jiayi_HW4_main.o Gu_Jiayi_HW4_main.c -g -I.
gcc -o Gu_Jiayi_HW4_main Gu_Jiayi_HW4_main.o -g -I. -l pthread
student@student-VirtualBox:~/Desktop/CSC415-01/csc415-assignment-4-word-blast-Jiayi079$
```

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


```
student@student-VirtualBox:~/Desktop/CSC415-01/csc415-assignment-4-word-blast-Jiayi079$ make run
gcc -c -o Gu_Jiayi_HW4_main.o Gu_Jiayi_HW4_main.c -g -I.
gcc -o Gu_Jiayi_HW4_main Gu_Jiayi_HW4_main.o -g -I. -l pthread
./Gu_Jiayi_HW4_main WarAndPeace.txt 2

Word Frequency Count on WarAndPeace.txt with 2 threads
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 1.278318446 seconds
student@student-VirtualBox:~/Desktop/CSC415-01/csc415-assignment-4-word-blast-Jiayi079$
```

```
student@student-VirtualBox:~/Desktop/CSC415-01/csc415-assignment-4-word-blast-Jiayi079$ make run
gcc -c -o Gu_Jiayi_HW4_main.o Gu_Jiayi_HW4_main.c -g -I.
gcc -o Gu_Jiayi_HW4_main Gu_Jiayi_HW4_main.o -g -I. -l pthread
./Gu_Jiayi_HW4_main WarAndPeace.txt 4

Word Frequency Count on WarAndPeace.txt with 4 threads
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 1.268394203 seconds
student@student-VirtualBox:~/Desktop/CSC415-01/csc415-assignment-4-word-blast-Jiayi079$
```

```
student@student-VirtualBox:~/Desktop/CSC415-01/csc415-assignment-4-word-blast-Jiayi079$ make run
gcc -c -o Gu_Jiayi_HW4_main.o Gu_Jiayi_HW4_main.c -g -I.
gcc -o Gu_Jiayi_HW4_main Gu_Jiayi_HW4_main.o -g -I. -l pthread
./Gu_Jiayi_HW4_main WarAndPeace.txt 6

Word Frequency Count on WarAndPeace.txt with 6 threads
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 1.257501281 seconds
student@student-VirtualBox:~/Desktop/CSC415-01/csc415-assignment-4-word-blast-Jiayi079$
```

```
student@student-VirtualBox:~/Desktop/CSC415-01/csc415-assignment-4-word-blast-Jiayi079$ make run
gcc -c -o Gu_Jiayi_HW4_main.o Gu_Jiayi_HW4_main.c -g -I.
gcc -o Gu_Jiayi_HW4_main Gu_Jiayi_HW4_main.o -g -I. -l pthread
./Gu_Jiayi_HW4_main WarAndPeace.txt 8

Word Frequency Count on WarAndPeace.txt with 8 threads
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 1.233280664 seconds
student@student-VirtualBox:~/Desktop/CSC415-01/csc415-assignment-4-word-blast-Jiayi079$
```