CSCI423/501 Project

NOTE1: The project is optional for undergraduate students (to take bonus points upon the completion of the project) and mandatory for graduate students.

NOTE2: This is a programming project, not a programming assignment, and will take significantly more time than a programming assignment. So start to work on it as early as you can.

NOTE3: A writing project report MUST be submitted together with the code. The report will count 50% of the your project grade. For the detailed guideline and requirement about the writing report, please refer to the course webpage under the Project section.

Part A

Write a multithread C program to count the frequency of words in a text file. In the program, the main thread should get the input file name from the command line arguments, open the file, do necessary global variable initialization, and perform necessary error checking during these steps. The main thread then creates a child thread to read each word from the text file. If the word has not appeared before, the child thread should access one global variable to store this word and set its frequency to 1. If the word has appeared before, the child thread should access the global variable to increase its frequency by 1. After the entire file is processed, the main thread then accesses the global variable and output the word-frequency data on the screen. The output should be one word each line, in the form of "word frequency". And all the words should be output alphabetically. For example, assume the compiled program is named as "a.out":

```
./a.out test2.txt
a 3
and 6
can 3
file 3
have 3
is 6
it 9
large 3
line 3
lines 3
multiple 3
one 3
or 3
```

```
single 3
test 3
very 3
```

where the content of the input text file is

```
It is a "test" file, and it is very large.
And it can have one single line or multiple lines.

It is a "test" file, and it is very large.
And it can have one single line or multiple lines.

It is a "test" file, and it is very large.
And it can have one single line or multiple lines.
```

Your program need not distinguish the upper and lower cases, e.g., "we" and "We" are deemed as the same word. And different forms of a word can be treated differently, e.g., "cat" and "cats" are treated as two different words, as well as "take", "took" and "taken" are treated as three different words.

Since both the main thread and the created child thread may access the file and the global variables, you may need to use mutex and/or other mechanisms to avoid the race conditions.

The entire txt file may be very large and not able to be held in the main memory.

For error checking, you need to consider whether the number of command line arguments is correct and whether the input text file can be opened. For example, assume the compiled program is named as "b.out":

```
./b.out
Usage: ./b.out <input text file name>

./b.out input.txt output.txt
Usage: ./b.out <input text file name>

./b.out input.txt
Input file input.txt cannot be opened.
```

The output messages for error checking should follow the format in the above examples.

If you use any dynamic memory management, you need to free the dynamically allocated memory space before your program terminates.

Hint:

1. You may need to use string manipulation functions such as strlen(), strcpy(), strncpy(), strcmp(), strstr(), etc. by include "string.h" in your code.

2. You may want to identify and divide the whole job into different smaller and easy-to-handle tasks (such as error checking, reading a word from the file, changing all letters in a word into lower case, removing punctuation, updating the word frequency, etc.) and implement them in different functions.

3. You may want to refer to the codes of the in-class discussion questions and your programming assignments, adopting and modifying part of them to be used in your project (e.g., create, maintain and update linked list).

4. You may want to start from the one thread version, where all the tasks are done within the main thread. Then create the child thread and move tasks to it.

5. You may assume the maximum length of a word is 512.

6. For apostrophes, you may assume there is no contraction such as "it's", "we're", etc. For possessives, you may ignore the " 's" and only consider the noun before it. E.g., for "Alice's", you only need to count "Alice" for once and ignore the " 's".

Part B

Extend the program in the Part A to create multiple children threads. The number of children threads should be input as another command line argument. For example, assume the compiled program is named as "b.out":

```
./b.out test2.txt 10
a 3
and 6
can 3
file 3
have 3
is 6
it 9
large 3
line 3
lines 3
multiple 3
one 3
or 3
single 3
test 3
```

```
very 3

./b.out
Usage: ./b.out <input text file name> <number of threads>

./b.out input.txt output.txt
Usage: ./b.out %s <input text file name> <number of threads>

./b.out input.txt 8
Input file input.txt cannot be opened.
```

Since all the created children threads will access the text file to read words and access the global variables to update word frequency, you may need to use mutex and/or other mechanisms to avoid the race conditions.

Hint:
You may want to refer to the codes of the in-class discussion questions and your programming assignments, adopting and modifying part of them to be used in your project (e.g., how to use mutex to avoid the race conditions).

Part C (only for **graduate** students)

Extend the program in the Part B to further improve the concurrency and parallelism. In particular, instead of letting each child thread reads word by word from the input file, the child thread now reads several lines (which we call a block) into its local buffer each time. It then reads and processes words from its local buffer, and when the block in the local buffer is processed, it read another block from the input file. In addition, during updating the word frequency, for each child thread, instead of directly updating a global variable, it now temporarily stores and updates the word frequency in its local variable, and then merge its local results into the global variable before it terminates.

The size of a block (i.e., the number of lines each time a child thread reads from the input file) should be input as another command line argument. For example, assume the compiled program is named as "c.out":

```
./c.out test2.txt 10 5
a 3
and 6
can 3
file 3
have 3
is 6
```

```
it 9
large 3
line 3
lines 3
multiple 3
one 3
or 3
single 3
test 3
very 3

./c.out
Usage: ./c.out <input text file name> <number of threads> <number
of lines per block>

./c.out input.txt output.txt
Usage: ./c.out %s <input text file name> <number of threads> <number
of lines per block>

./c.out input.txt 8
Usage: ./c.out %s <input text file name> <number of threads> <number
of lines per block>
```

Hint:
You may assume the maximum number of words per line is 1024.