Jordan University of Science and Technology

Computer Engineering Department

CPE 473: Operating Systems

Programming Assignment #2: Threading

Notes:

1- **The due date is 23/12/2023 at 11:55 PM.**
2- **Late submissions (i.e. after the due date) will not be accepted.**
3- **Submit the source code files and the PDF report. Write your name and ID at the top**
4- **Groups are allowed (up to 3 students, it is OK to be from different sections)**
5- **Only one submission needed per group. All group members must attend discussion.**

In this assignment, you will implement a multi-threaded program (using C/C++). **The project is a simplified search engine (e.g. Google).** The program will create T worker threads that will operate on a list of files and searching for a specific character (CHAR). Each thread will be responsible for a number of files. For each file, the corresponding thread will count the number of times it found CHAR in the file, and will update the global variables described next. The program will read an input (TH) which is the threshold that will be used to calculate the global variables.

Your program should have at least four global shared variables to report the statistics of files your search engine is looking into:

1. TotalFound: Will track the total number of times CHAR was found in all files.
2. AboveThreshold: Will track the total number of files where CHAR was found more than TH times
3. EqualsThreshold: Will track the total number of files where CHAR was found exactly TH times
4. BelowThreshold: Will track the total number of files where CHAR was found less than TH times

When any of the threads starts executing, it will print its number (0 to T-1), and then the range in the files list that it is operating on. For each of these files, the thread will print the statistics regarding this file. Then, when the thread is done, it will print a line indicating its conclusion. Finally, when all threads are done, the main should print the statistics of all the files in the entire list. See the sample output/input for more details. **You should write two versions of the program: The first one doesn't consider race conditions, and the other one is thread-safe.**

The input will be provided in an input file (in.txt), and the output should be printed to an output file (out.txt). The number of worker threads (T) and the targeted character (CHAR) will be passed through the command line, as mentioned earlier. The input in (in.txt) will start with a line containing two numbers, an integer value N, representing the number of files in the list, and an integer TH representing the threshold that will be used by the global variables, as discussed earlier. All the files that will be searched (i.e. included within in.txt) should be located in the same directory were your program is located.

All the printing messages about statistics (from threads and main) will be printed to the standard output (STDOUT). The details about the files will be printed to (out.txt) as a sorted files list, where all the files are listed based on the number of occurrences of CHAR (from higher to lower) with each file listed along with its found count.

Tasks:

In this assignment, **you will submit your source code files for the thread-safe and thread-unsafe versions, in addition to a report (PDF file).** The report should show the following:

1. Screenshot of the main code
2. Screenshot of the thread function(s)
3. Screenshot highlighting the parts of the code that were added to make the code thread-safe, with explanations on the need for them
4. Screenshot of the output of the two versions of your code (thread-safe vs. non-thread-safe), when running passing the following number of threads (T): 1, 2, 4, 8, 16.
5. Based on your code, how many computing units (e.g. cores, hyper-threads) does your machine have? Provide screenshots of how you arrived at this conclusion, and a screenshot of the actual properties of your machine to validate your conclusion. It is OK if your conclusion doesn't match the actual properties, as long as your conclusion is reasonable.

Hints:

1. **Read this document carefully multiple times to make sure you understand it well. Do you still have more questions? Ask us during our helping sessions, we'll be happy to help!**
2. Plan well before coding. Especially on how to divide the range over worker threads. How to synchronize accessing the variables/files.
3. For passing the number of threads (T) to the code, you will need to use *argc*, and *argv* as parameters for the main function. For example, the Linux command for running your code with two worker threads (i.e. T=4, and CHAR='r') will be something like: "./a.out 4 r"
4. All values can be up to 100,000,000. If N>T, you can assume that the variables (N) will be divisible by (T). However, If the T>N, then the remaining (T-N) threads should simply print a line indicating that they have no work to do and then exit. You can also assume that no file will be searched by multiple threads.
5. For answering Task #5 regarding the number of computing units (e.g. cores, hyper-threads) in your machine, search about "diminishing returns". You also might need to use the Linux command "*time*" while answering Task #4, and use input with many large input files.
6. You will, obviously, need to use pthread library and Linux. I suggest you use the threads coding slides to help you with the syntax of the pthread library
7. If two or more files have the same number of occurrences of the search target CHAR, then they will be appear in the sorted list in the opposite order they originally appeared in the (in.txt) file.

Sample Input (in.txt), assuming passing T=4 and CHAR=r in the command line (i.e. ./a.out 4 r):

```
4 2
inputFile1.txt
inputFile2.txt
inputFile3.txt
inputFile4.txt
```

Files to search for (should be located in the same directory where your program is located)

inputFile1.txt	**inputFile2.txt**	**inputFile3.txt**	**inputFile4.txt**
abb aatbbarbbbc abbb aabtbarbcc ababbb arbbbabcc	baatbRbb	1 r @ a	Hello World!

Sample Output (STDOUT terminal part): → Notice that some of threads' output got reordered, that is fine!

```
Main   -->   Search Engine searching for (r) in 4 files, using 4 threads (with threshold=2)
TID0   -->   Starting thread firstItem=0, lastItem=1
TID1   -->   Starting thread firstItem=1, lastItem=2
TID2   -->   Starting thread firstItem=2, lastItem=3
TID0   -->   File: inputFile1.txt, (r) found=3
TID0   -->   Ending thread firstItem=0, lastItem=1
TID3   -->   Starting thread firstItem=3, lastItem=4
TID2   -->   File: inputFile3.txt, (r) found=1
TID2   -->   Ending thread firstItem=2, lastItem=3
TID1   -->   File: inputFile2.txt, (r) found=0
TID1   -->   Ending thread firstItem=1, lastItem=2
TID3   -->   File: inputFile4.txt, (r) found=1
TID3   -->   Ending thread firstItem=3, lastItem=4

Main   -->   TotalFound=5, AboveThreshold=1, EqualsThreshold=0, BelowThreshold=3
Main   -->   Sorted list of files in out.txt
```

Sample Output (out.txt part):

```
Sorted list of files:
1.[ inputFile1.txt ] (found = 3)
2.[ inputFile4.txt ] (found = 1)
3.[ inputFile3.txt ] (found = 1)
4.[ inputFile2.txt ] (found = 0)
```