Due before:	11:59 PM (before Midnight) on Wed Oct 14 2020

CSE-381: Systems 2

Homework #6

Due: Wed October 14 2020 before 11:59 PM

Email-based help Cutoff: 5:00 PM on Tue, Oct 13 2020

Maximum Points: 25

Submission Instructions

This homework assignment must be turned-in electronically via Canvas CODE plug-in. Ensure your C++ source code is named `MUID_hw6.cpp`, where `MUID` is your Miami University Unique ID. Ensure your program compiles without any warnings or style violations. Ensure you thoroughly test operations of your program as indicated. Once you have tested your implementation, upload the following:

1. The 1 C++ source file developed for this homework.

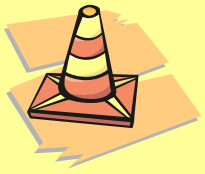
General Note: Upload each file associated with homework individually. Do not upload archive file formats such as zip/tar/gz/7zip/rar etc.

Objective

The objective of this homework is to:

- Develop a data parallel (without any synchronization) multithreaded C++ program for simple Natural Language Processing (NLP).
- Continue to gain familiarity with use of HTTP protocol for Inter-Process Communication (IPC)
- Continue to gain familiarity with I/O streams & string processing
- Continue to gain deeper understanding of containers (`std::vector` and `std::unordered_map`)
- Work on simple problem-solving aspects

Grading Rubric:



The program submitted for this homework **must pass necessary base case test(s) in order to qualify for earning any score at all**. Programs that do not meet base case requirements will be assigned zero score! Program that do not compile, have a method longer than 25 lines, bad formatting including }}}, or just some skeleton code will be assigned zero score.

- Base case (No multithreading. Only simple string processing): **10 points**
- Data parallel (*i.e.*, no synchronization) multithreading: **10 points**
- Awesome formatting and documentation: **3 + 2 = 5 Points:** Reserved for good documentation, overall structure, organization, **conciseness**, variable-names, etc. **If your methods are not concise points will be deducted**. Maximum of 3 points for just base case implementation.

NOTE: Violating CSE programming style guidelines is an error! Your program should not have any style violations.

Starter code

The starter code supplied for this homework is barebones and only has the necessary `#includes`. Most of the networking operations have been done in previous labs and homework. Of course, you can (and probably should) get the necessary logic from previous work we have done.

Problem statement

This project requires you to do basic processing of files/data downloaded from a given web-site, given a list of 1-or-more files as command-line arguments. Each file **must be processed by a separate thread**. For each file, your program must print the number of words and number of valid-English words. You are supplied a dictionary (see `english.txt`) to validate English words. Ignore punctuations and convert words to lower case when processing words. The statistics must be printed in **exactly the same order** as the command-line arguments supplied to your program. See sample outputs below.

Web-site to be used: The contents of each file should be obtained from the following base-URL: `http://os1.csi.miamioh.edu/~raodm/cse381/hw4/SlowGet.cgi?file=`. For example, given a file `ex3.html`, the full URL will be: `http://os1.csi.miamioh.edu/~raodm/cse381/hw4/SlowGet.cgi?file=ex3.html`. **Note: Even though the URLs are show (for testing/checking) it is not necessary to construct the full URL in your programs**. You may also assume that the remote server will not use chunked format to return contents of the file you requested.

Required output format: The statistics for each file must be printed in the following format. The result of processing files should be displayed in exactly the same order in which they were specified as command-line arguments. For each file, the program should generate 1 line of output, in the following format: `<FileName>:␣words=<#Words>,␣English␣words=<#EngWords>`, where ␣ is 1 blank space. Strings in blue are literal constants. See sample outputs below for examples.

Suggestions and Tips

1. First plan (*i.e.*, write comments and pseudocode) on how you are planning to solve the problem.

2. Then get the base case without any multithreading working. The base case should be able generated the required output without multithreading.
    a. Downloading data from a given URL has been done in prior homework. Get the code from there.
    b. Load the words in the given `english.txt` into an `unordered_map` to ease checking for valid English words. A simple while-loop will do the work here.

3. The contents of the file should be processed line-by-line. Before processing words in a line. For each line, start by removing punctuations and convert it to lowercase as suggested below:

```
// Remove punctuations in a line
std::replace_if(line.begin(), line.end(), ::ispunct, ' ');
```

In addition, convert all the characters to lowercase. This is done to ease checking for valid English words, because the English dictionary has words in lowercase.

```
// Convert the word to lower case to check against the dictionary.
std::transform(line.begin(), line.end(), line.begin(), ::tolower);
```

4. Using the above regularized line, the number of words and number of valid English words must be computed (hint: `std::istringstream`). A dictionary is supplied to determine valid English words. It is best to load the dictionary once into a global-constant `std::unordered_map` as suggested below:

```
// Shortcut an unordered_map of words.
using Dictionary = std::unordered_map<std::string, bool>;

// Forward declaration for method.
Dictionary loadDictionary(const std::string& filePath);

// The global dictionary of valid words
const Dictionary dictionary = loadDictionary("english.txt");
```

5. **Multithreading tips:**
   a. Ensure all the processing is done in threads.
   b. It may be easiest to setup a `threadMain` method in the form `void threadMain(const std::string& filename, std::string& result);`
   c. Use an `std::vector<std::string>`, initialized to the necessary number of blank entries, to collect results from each thread.
   d. Remember you need 2-loops:
      i. One to start the threads
      ii. One loop to join all the threads
   e. Print the results (from the above vector) only after all the threads have been joined.

Sample inputs and outputs

The command typed in is shown in bold. Video on setting command-line arguments in `NetBeans` is available on Canvas.

The base case does not require any multithreading and is relatively straightforward (pieces can be copy-pasted from prior homework). So, invest time and/or seek help to learn this basic string processing right. Since the program is straightforward, if the base case does not operate as expected, as per the course policy, the program will be assigned zero score.

Base case #1:

```
$ ./homework6 ex3.html
ex3.html: words=39, English words=16
```

Base case #2:

```
$ ./homework6 ex3.html cpp.txt
ex3.html: words=39, English words=16
cpp.txt: words=6983, English words=5810
```

Multithreading test:

```
$ ./homework6 ex3.html cpp.txt us_constitution.txt miami_university.txt
ex3.html: words=39, English words=16
cpp.txt: words=6983, English words=5810
us_constitution.txt: words=7667, English words=7422
miami_university.txt: words=32331, English words=15348
```

Submit to Canvas

This homework assignment must be turned-in electronically via Canvas CODE plug-in. Ensure your C++ source files are named appropriately. Ensure your program compiles (without any warnings or style errors) successfully. Ensure you have tested operations of your program as indicated. Once you have tested your implementation, upload the following:

- The 1 C++ source file you developed for this homework
- **Note: 5 Points:** Are reserved for good program structure and documentation. These 5-points are possibly the hardest to earn.

Upload the C++ source files to onto Canvas. Do not submit zip/7zip/tar/gzip files. Upload each file independently.