Sabanci University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Summer 2017-2018

Homework 5 – Waiting for the Worms
Due: 13 August 2018 11.55pm

DISCLAIMER:

Your program should be a robust one such that you have to consider all relevant user mistakes and extreme cases; you are expected to take actions accordingly!

Only checking the sample run cases might not be sufficient as your solution will be checked against a variety of samples different than the provided samples; however checking these cases are highly encouraged and recommended.

You can NOT collaborate with your friends and discuss your solutions with each other. You have to write down the code on your own. Plagiarism will not be tolerated AND cooperation is not an excuse!

Introduction

The aim of this homework is to practice on multithreaded operations and particularly producer-consumer pattern. In this homework, you will simulate a system where encrypted files are supplied by two producer threads to a main queue and three consumer threads dequeue these file names and decrypt these files of given names. Let's see if cryptography is fun!

Input to Your Program

Input to your program will be several encrypted files, which are given in the homework bundle. Before launching your producer and consumer threads, your main function will read the names of all files the program will use. Typing only a dash "-" will stop receiving file names. Your program will not immediately check whether the file names are correct or not; your consumer threads will check it when they dequeue a file name from the main queue.

Format of the Input Files

The input files may be in any format and contain any character but you can only assume that they **do not** contain uppercase letters. This assumption is important for your decryption algorithm.

Threads and what they do

First of all, your program needs to read the file names before launching any threads, as explained in the previous sections. Your program may store these file names in any data structure of your choice. (For sample runs, a stack is used for this purpose).

After that, you need to launch 2 producer and 3 consumer threads from the main thread. Do not forget that producer threads need access to the file names.

Producer threads: These threads have one simple duty: They need to get a file name from the structure of your choice and place it into the main queue. Make sure that you understand that the file name structure and the main queue are shared by multiple threads so you need to take some precautions before and after using each of them :) We all know that this precaution is called a “mutex”. Make sure to delete file names from your structure so that they are not placed into the queue multiple times.

Producer threads need to sleep **before** doing some work. This sleep time will be randomly calculated each time by your program. You can use the RandGen library given to you for picking a random integer between 1000 and 4000, in milliseconds. The producer threads will terminate when there are no more files left to enqueue to the main queue.

Consumer threads: These threads love sleeping as well, as we all do. They sleep for a random time between 2500 to 3500 milliseconds before **trying** to do anything.

These threads will first **try** to dequeue a file name from the main queue. Bear in mind that trying to dequeue an empty queue will result in crash. So, always check its emptiness beforewards.
If the dequeuing is successful (i.e, queue is not empty), the thread will open the file, decrypt it and write the result to another file. Details of the decryption algorithm is discussed in the following sections. If the file name is not valid, the thread will print an error message and will not dequeue another item immediately (i.e, it will repeat the process which starts with sleeping)

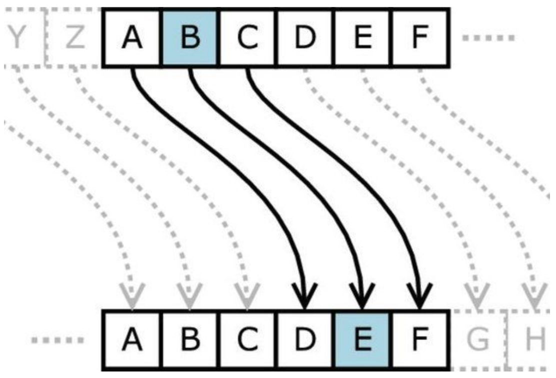
If the queue is empty, there can be two reasons: (i) there are files left but they are not enqueued yet; so the consumer threads should not terminate yet, (ii) there are no files left and the consumer threads may terminate. Your program can maintain this information by setting a global variable by producer threads about whether the files are exhausted or not and letting your consumer threads read from this variable. As producers only write to this variable and consumers only read, no locking mechanism would be needed for this variable.

All the actions that your threads could or could not take should be written to the console. Sample runs provide examples for the outputs that you should produce. Do not forget that *cout* is an object that you share between threads, hence you should use it with care, as well :)

A very basic string queue implementation is given to you with the homework bundle. You **must** use it, without changing anything. Note that this queue is of unlimited size.

Note: RandGen library is not reliable in multithreading. Some threads may get the same random number if requested in close intervals. Do not mind, this problem will not affect your grades ;)

Caesar's Cipher



Caesar's Cipher pattern with shift = +3

In cryptography, Caesar's cipher is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions up or down the alphabet.

For example, with a shift of +3, 'a' would be replaced by 'd', 'b' would become 'e', and so on. The shift may also be negative or zero (whereas zero shift means not encrypting :D) The method is named after Julius Caesar, who used it in his private conversations.

This method is circular, which means that, 'z' encrypted with a shift of +5 would become 'e' and 'c' encrypted with a shift of -4 would become 'y'.

In this homework, you are given 15 encrypted files. The original files (also the encrypted versions that you have) do not contain any uppercase letters and only the lowercase letters are ciphered with Caesar's method. The files contain whitespaces and other characters such as punctuations, but these characters are not encrypted and copied as they are, so no need to deal with them.

You need to hack the files and produce their originals. The problem is: We do not know what amount of shift applied to each file! One file may be encrypted with a shift of +10 where the other might be -20. So, we need a way to guess the shift of the encrypted file :/

The golden information comes: The original files are some text written in *English.* You may think, “How is this useful?”. The most common letter in English words is ‘e’. If you find the most common letter in the encrypted file, probably it corresponds to ‘e’, and you have the shift amount!

Disclaimer: The originals of all files given to you have the most common letter as ‘e’ :)

I see Mutexes

Critical information accessed by threads always have problem of validation faults. What do you think will happen if your thread K is stopped by the operating system when K had read an item from the queue but had not deleted it yet? Two consumer threads would read the same item and do the decryption multiple times. This is not the worst case. Afterwards, they would try to delete the same node twice and crash the program :/ Now, you got the spirit!

Any resource that can be reached by threads and cannot be used in an atomic way should be restricted with mutexes. Examples of such situations in this homework are: (i) access to the file name structure, (ii) access to the main queue, (iii) printing to the console.

In order not to have inefficiencies, use seperate mutexes for each resource. Making another thread wait for the dequeuing just because you are printing something is not fair!

Another important point is that, make sure you lock minimum amount of necessary operations with mutexes. Lock the mutexes right before your thread starts accessing a mentioned resource and unlock it as soon as it is done dealing with that resource.

Output Files

Input files are given you with .txt extension. The output files should contain the decrypted version of the corresponding input file and should end with “_decrypted.txt”. As an example: decrypted version of “file1.txt” should be written to a file named “file1_decrypted.txt”.

Sample Runs

Below, we provide some sample runs of the program that you will develop. The *italic* and **bold** phrases are the standard inputs (cin) taken from the user (i.e., like ***this***). You have to display the required information with the same words as here.

The ordering of the operations, scenarios and outputs may vary depending on anything. We will take this into consideration, for sure.

We are not giving you the resulting output files. You can immediately understand that you are doing it right once you see some English text as the result (although they are meaningless :D)

The shift amounts are removed from the outputs (--), apart from the one of file3.txt . Those shift amounts should be found by your program using the most common letter property mentioned above. Do not try to hardcode anything as your programs will be tested with different files.

Sample Run 1

```
Please enter a file name: file3.txt
Please enter a file name: file4.txt
Please enter a file name: file5.txt
Please enter a file name: asdasd
Please enter a file name: --
Please enter a file name: *
Please enter a file name: -
Producer Thread - 1 starts sleeping for 1715 milliseconds
Producer Thread - 2 starts sleeping for 1003 milliseconds
Consumer Thread - 1 starts sleeping for 2501 milliseconds
Consumer Thread - 2 starts sleeping for 2501 milliseconds
Consumer Thread - 3 starts sleeping for 2501 milliseconds
Producer Thread - 2 is now enqueuing "*"
Producer Thread - 2 starts sleeping for 2691 milliseconds
Producer Thread - 1 is now enqueuing "--"
Producer Thread - 1 starts sleeping for 2865 milliseconds
Consumer Thread - 1 is now handling "*"
Consumer Thread - 2 is now handling "--"
Consumer Thread - 1 cannot process "*", there is no such file!
Consumer Thread - 3 cannot find any file to dequeue for now.
Consumer Thread - 2 cannot process "--", there is no such file!
Consumer Thread - 1 starts sleeping for 3064 milliseconds
Consumer Thread - 3 starts sleeping for 3064 milliseconds
Consumer Thread - 2 starts sleeping for 3064 milliseconds
Producer Thread - 2 is now enqueuing "asdasd"
Producer Thread - 2 starts sleeping for 1580 milliseconds
Producer Thread - 1 is now enqueuing "file5.txt"
Producer Thread - 1 starts sleeping for 1783 milliseconds
Producer Thread - 2 is now enqueuing "file4.txt"
Producer Thread - 2 starts sleeping for 3426 milliseconds
Consumer Thread - 1 is now handling "asdasd"
Consumer Thread - 3 is now handling "file5.txt"
Consumer Thread - 2 is now handling "file4.txt"
Consumer Thread - 1 cannot process "asdasd", there is no such file!
Consumer Thread - 2 is done handling "file4.txt" with a shift of -- and written the result to "file4_decrypted.txt"
Consumer Thread - 1 starts sleeping for 2693 milliseconds
Consumer Thread - 3 is done handling "file5.txt" with a shift of -- and written the result to "file5_decrypted.txt"
Consumer Thread - 2 starts sleeping for 2693 milliseconds
Consumer Thread - 3 starts sleeping for 2693 milliseconds
Producer Thread - 1 is now enqueuing "file3.txt"
Producer Thread - 1 starts sleeping for 1695 milliseconds
Producer Thread - 1 cannot find any file to enqueue, joining...
Consumer Thread - 1 is now handling "file3.txt"
Consumer Thread - 1 is done handling "file3.txt" with a shift of 13 and written the result to "file3_decrypted.txt"
Consumer Thread - 2 cannot find any file to dequeue, joining...
Consumer Thread - 1 starts sleeping for 3309 milliseconds
Consumer Thread - 3 cannot find any file to dequeue, joining...
Producer Thread - 2 cannot find any file to enqueue, joining...
Consumer Thread - 1 cannot find any file to dequeue, joining...
All threads have joined with main, exiting...
Press any key to continue . . .
```

Sample Run 2

```
Please enter a file name: -
Producer Thread - 1 starts sleeping for 2692 milliseconds
Producer Thread - 2 starts sleeping for 1003 milliseconds
Consumer Thread - 1 starts sleeping for 2501 milliseconds
Consumer Thread - 2 starts sleeping for 2501 milliseconds
Consumer Thread - 3 starts sleeping for 2501 milliseconds
Producer Thread - 2 cannot find any file to enqueue, joining...
Consumer Thread - 1 cannot find any file to dequeue, joining...
Consumer Thread - 2 cannot find any file to dequeue, joining...
Consumer Thread - 3 cannot find any file to dequeue, joining...
Producer Thread - 1 cannot find any file to enqueue, joining...
All threads have joined with main, exiting...
```

```
Press any key to continue . . .
```

Sample Run 3

```
Please enter a file name: file1.txt
Please enter a file name: file2.txt
Please enter a file name: file3.txt
Please enter a file name: file4.txt
Please enter a file name: file5.txt
Please enter a file name: file6.txt
Please enter a file name: file7.txt
Please enter a file name: file8.txt
Please enter a file name: file9.txt
Please enter a file name: file10.txt
Please enter a file name: file11.txt
Please enter a file name: file12.txt
Please enter a file name: file13.txt
Please enter a file name: file14.txt
Please enter a file name: file15.txt
Please enter a file name: file16.txt
Please enter a file name: -
Producer Thread - 1 starts sleeping for 1727 milliseconds
Producer Thread - 2 starts sleeping for 1003 milliseconds
Consumer Thread - 1 starts sleeping for 2501 milliseconds
Consumer Thread - 2 starts sleeping for 2501 milliseconds
Consumer Thread - 3 starts sleeping for 2501 milliseconds
Producer Thread - 2 is now enqueuing "file16.txt"
Producer Thread - 2 starts sleeping for 2691 milliseconds
Producer Thread - 1 is now enqueuing "file15.txt"
Producer Thread - 1 starts sleeping for 2592 milliseconds
Consumer Thread - 1 is now handling "file16.txt"
Consumer Thread - 1 cannot process "file16.txt", there is no such file!
Consumer Thread - 2 is now handling "file15.txt"
Consumer Thread - 1 starts sleeping for 3064 milliseconds
Consumer Thread - 3 cannot find any file to dequeue for now.
Consumer Thread - 3 starts sleeping for 3064 milliseconds
Consumer Thread - 2 is done handling "file15.txt" with a shift of -- and written the result to "file15_decrypted.txt"
Consumer Thread - 2 starts sleeping for 3064 milliseconds
```

```
Producer Thread - 2 is now enqueuing "file14.txt"
Producer Thread - 2 starts sleeping for 1580 milliseconds
Producer Thread - 1 is now enqueuing "file13.txt"
Producer Thread - 1 starts sleeping for 2458 milliseconds
Producer Thread - 2 is now enqueuing "file12.txt"
Producer Thread - 2 starts sleeping for 3426 milliseconds
Consumer Thread - 1 is now handling "file14.txt"
Consumer Thread - 3 is now handling "file13.txt"
Consumer Thread - 1 is done handling "file14.txt" with a shift of -- and written the result to
"file14_decrypted.txt"
Consumer Thread - 3 is done handling "file13.txt" with a shift of -- and written the result to
"file13_decrypted.txt"
Consumer Thread - 1 starts sleeping for 2693 milliseconds
Consumer Thread - 3 starts sleeping for 2693 milliseconds
Consumer Thread - 2 is now handling "file12.txt"
Consumer Thread - 2 is done handling "file12.txt" with a shift of -- and written the result to
"file12_decrypted.txt"
Consumer Thread - 2 starts sleeping for 2693 milliseconds
Producer Thread - 1 is now enqueuing "file11.txt"
Producer Thread - 1 starts sleeping for 1161 milliseconds
Producer Thread - 1 is now enqueuing "file10.txt"
Producer Thread - 1 starts sleeping for 3601 milliseconds
Consumer Thread - 1 is now handling "file11.txt"
Consumer Thread - 3 is now handling "file10.txt"
Consumer Thread - 1 is done handling "file11.txt" with a shift of -- and written the result to
"file11_decrypted.txt"
Consumer Thread - 1 starts sleeping for 3309 milliseconds
Consumer Thread - 3 is done handling "file10.txt" with a shift of -- and written the result to
"file10_decrypted.txt"
Consumer Thread - 3 starts sleeping for 3309 milliseconds
Consumer Thread - 2 cannot find any file to dequeue for now.
Consumer Thread - 2 starts sleeping for 3309 milliseconds
Producer Thread - 2 is now enqueuing "file9.txt"
Producer Thread - 2 starts sleeping for 2755 milliseconds
Producer Thread - 2 is now enqueuing "file8.txt"
Producer Thread - 2 starts sleeping for 2440 milliseconds
Producer Thread - 1 is now enqueuing "file7.txt"
Producer Thread - 1 starts sleeping for 3771 milliseconds
Consumer Thread - 1 is now handling "file9.txt"
Consumer Thread - 3 is now handling "file8.txt"
Consumer Thread - 2 is now handling "file7.txt"
Consumer Thread - 1 is done handling "file9.txt" with a shift of -- and written the result to
"file9_decrypted.txt"
Consumer Thread - 3 is done handling "file8.txt" with a shift of -- and written the result to
"file8_decrypted.txt"
Consumer Thread - 2 is done handling "file7.txt" with a shift of -- and written the result to
"file7_decrypted.txt"
Consumer Thread - 1 starts sleeping for 3085 milliseconds
Consumer Thread - 3 starts sleeping for 3085 milliseconds
Consumer Thread - 2 starts sleeping for 3085 milliseconds
Producer Thread - 2 is now enqueuing "file6.txt"
Producer Thread - 2 starts sleeping for 2051 milliseconds
Consumer Thread - 1 is now handling "file6.txt"
Consumer Thread - 3 cannot find any file to dequeue for now.
Consumer Thread - 3 starts sleeping for 2980 milliseconds
Consumer Thread - 2 cannot find any file to dequeue for now.
Consumer Thread - 1 is done handling "file6.txt" with a shift of -- and written the result to
"file6_decrypted.txt"
Consumer Thread - 2 starts sleeping for 2980 milliseconds
Consumer Thread - 1 starts sleeping for 2980 milliseconds
Producer Thread - 1 is now enqueuing "file5.txt"
Producer Thread - 1 starts sleeping for 2881 milliseconds
Producer Thread - 2 is now enqueuing "file4.txt"
Producer Thread - 2 starts sleeping for 3688 milliseconds
Consumer Thread - 3 is now handling "file5.txt"
Consumer Thread - 3 is done handling "file5.txt" with a shift of -- and written the result to
"file5_decrypted.txt"
Consumer Thread - 2 is now handling "file4.txt"
Consumer Thread - 3 starts sleeping for 2850 milliseconds
Consumer Thread - 1 cannot find any file to dequeue for now.
Consumer Thread - 1 starts sleeping for 2850 milliseconds
Consumer Thread - 2 is done handling "file4.txt" with a shift of -- and written the result to
"file4_decrypted.txt"
Consumer Thread - 2 starts sleeping for 2850 milliseconds
Producer Thread - 1 is now enqueuing "file3.txt"
Producer Thread - 1 starts sleeping for 1942 milliseconds
Producer Thread - 2 is now enqueuing "file2.txt"
Producer Thread - 2 starts sleeping for 3469 milliseconds
Producer Thread - 1 is now enqueuing "file1.txt"
Producer Thread - 1 starts sleeping for 1327 milliseconds
Consumer Thread - 3 is now handling "file3.txt"
Consumer Thread - 1 is now handling "file2.txt"
Consumer Thread - 3 is done handling "file3.txt" with a shift of 13 and written the result to
"file3_decrypted.txt"
Consumer Thread - 3 starts sleeping for 3396 milliseconds
Consumer Thread - 1 is done handling "file2.txt" with a shift of -- and written the result to
"file2_decrypted.txt"
Consumer Thread - 2 is now handling "file1.txt"
Consumer Thread - 1 starts sleeping for 3396 milliseconds
Consumer Thread - 2 is done handling "file1.txt" with a shift of -- and written the result to
"file1_decrypted.txt"
Consumer Thread - 2 starts sleeping for 3396 milliseconds
Producer Thread - 1 cannot find any file to enqueue, joining...
Producer Thread - 2 cannot find any file to enqueue, joining...
Consumer Thread - 3 cannot find any file to dequeue, joining...
Consumer Thread - 1 cannot find any file to dequeue, joining...
Consumer Thread - 2 cannot find any file to dequeue, joining...
All threads have joined with main, exiting...
Press any key to continue . . .
```

Some Important Rules

Although some of the information is given below, please also read the homework submission and grading policies from the lecture notes of the first week. In order to get a full credit, your program must be efficient, modular (with the use of functions), well commented and indented. Besides, you also have to use understandable identifier names. Presence of any redundant computation, bad indentation, meaningless identifiers or missing/irrelevant comments may decrease your grade in case that we detect them.

When we grade your homeworks, we pay attention to these issues. Moreover, in order to observe the real performance of your codes, we are going to run your programs in Release mode and **we may test your programs with very large test cases**. Hence, take into consideration the efficiency of your algorithms other than correctness.

How to get help?

You may ask your questions to TAs or to the instructor. Information regarding the office hours of the TAs and the instructor are available at SUCourse.

WE WILL NOT USE GRADE CHECKER FOR THIS HOMEWORK!

Because of the multithreaded nature of the homework, we will not use grade checker for this homework. 15 text files are given for you to test your implementation and you can create more files yourselves for testing, as well (https://cryptii.com/caesar-cipher). We will **not** use the same input files during grading. But you **must** still submit these text files along with your implementation.

Submit via SUCourse ONLY! Paper, e-mail or any other methods are not acceptable.
The internal clock of SUCourse might be a couple of minutes skewed, so make sure you do not leave the submission to the last minute. In the case of failing to submit your homework on time:

"No successful submission on SUCourse on time = A grade of 0 directly."

What and where to submit (PLEASE READ, IMPORTANT)

You should prepare your program using MS Visual Studio 2012, as we will use it while testing your homework this time.

It'd be a good idea to write your name and lastname in the program (as a comment line of course). Do not use any Turkish characters anywhere in your code (not even in comment parts). If your full name is "Duygu Karaoğlan Altop", and if you want to write it as comment; then you must type it as follows:

// Duygu Karaoglan Altop

Submission guidelines are below. Since the grading process will be semi-automatic, you are expected to strictly follow these guidelines. If you do not follow these guidelines, your grade will be zero. The lack of even one space character in the output will result in your grade being zero, so please test your programs yourself.

- Name the main.cpp into:

  "SUCourseUserName_hw5.cpp"

  Your SUCourse user name is actually your SUNet username which is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SU e-mail address is **atam@sabanciuniv.edu**, then the file name must be: **"atam_hw5.cpp"**.

- Please make sure that the files are the latest versions of your homework program.

- You should upload the sample txt files, header and cpp files of the classes given to you (randgen and queue) and your main cpp file.

- Do not zip any of the documents but upload them as separate files only.

- Submit your work **<u>through SUCourse only</u>**!

You may visit the office hours if you have any questions regarding submissions.

Plagiarism

Plagiarism is checked by automated tools and we are very capable of detecting such cases. Be careful with that...

Exchange of abstract ideas are totally okay but once you start sharing the code with each other, it is very probable to get caught by plagiarism. So, do <u>NOT</u> send any part of your code to your friends by any means or you might be charged as well, although you have done your homework by yourself. Homeworks are to be done personally and you have to submit your own work. **Cooperation will NOT be counted as an excuse.**

In case of plagiarism, the rules on the Syllabus apply.

Good Luck!
Tolga Atam, Duygu K. Altop