Assignment Concurrency 21/22 (draft)

Reza Hassanpour
Ahmad Omar
Afshin Amighi
Andrea Minuto

Subject:

The course of concurrency requires to create a concurrent effective **message decryption system** that handles several provided encrypted messages by a client program.

Groups:

An assignment can be carried out by a group of 1 or 2 students (no exceptions). All the members of the group must be within the same class (same teacher). Each group needs to provide the implementation of a **concurrent** program that can process and decrypt a list provided challenges.

Scenario:

There is a list of encrypted strings (check file *challenges.txt*) that needs to be decrypted. A decryption algorithm is already implemented and provided (check class *TaskDecryption, no need to add another one or make it more efficient*).

- **Worker**: Worker objects (check class *Worker*) is responsible to execute the decryption algorithm on the given challenges (check *TaskDecryption::Execute( )*).
- **Provider**: One provider (check class *Provider*) reads all the challenges and in every "*t*" milli seconds, sends one challenge (check *Provider::SendTasks( )*). "*t*" is a random number between *minSendIntervalTime* and *maxSendIntervalTime* (check class *WorkingParams*). Smaller values for *t* means higher speed for the provider.
- **Buffer**: The provider and worker objects are communicating using the shared memory mechanism (check class *Buffer*). The *provider* object feeds the shared *buffer* with the tasks. Each task contains an input (a string to be decrypted) and an output field which should contain the result of the processing.

A sequential version of the processing program is already implemented (check class *Launch*). This sequential version is not efficient/concurrent.

Assignment details:

Exercise: Run the sequential version and check the execution time. There is a need for a more efficient/concurrent program **without changing the decryption algorithm executed** by worker object (check class *Worker*).

Your assignment will be to add concurrent features to the provided package using the following guidelines:

- There will be **only one provider**. Therefore, **the provider must not change**.
- To increase efficiency you must employ **several concurrent workers**.

- The shared buffer needs **to support simultaneous accesses** from one provider and multiple workers. Therefore, class *ConcurrentTaskBuffer* must be complete.
- There is already a sequential execution implemented within the class *Launch*. This class must be extended with concurrent execution. Therefore, class *ConcLaunch* must be completed.

What will be evaluated:

- Respecting the requirements above.
- Minimal decency in the code quality.
- The program should be thread safe.
- It should have proper concurrent implementation (not serial).

Submissions:

Instructions for the submissions:

- Submission Deadline: **January 28, 2022 - 23:59**.
- Programming Language: C# is the official language for the project.
- The size of the buffer will be changing during the execution. The maximum size that the buffer experiences during the execution is printed at the end.
- Submitted file MUST contain the group's requested information. Check class *WorkingParams* for more details.
- *In your final submission you must tune three parameters (check WorkingParams): numOfWorkers, minSendIntervalTime and maxSendIntervalTime such that the maximum of* ***the buffer size*** *stays between 50 and 100.* Minimum accepted value for *numOfWorkers* is 5. *Submitted solution must contain log.txt of the student's last execution.*
- The assignment will receive a passing grade if the concurrent implementation reflects all the teachings of the course, that means it must implement a **thread safe concurrent program**.
- Submission Procedure: Will be communicated through Teams.

Reflection points:

- **What is printed in the console during the execution?** During the execution of the program, you will recognise two types of the information printed in the console: One is the size of the buffer. For example *Buffer#2* means the buffer at this moment contains two tasks; and the other is the result of the worker object, for example *TaskID:23[(challenge:x,Hash:y), Keys:(p1,p2)]* means that task number 23 with values x and y has finished and the result is p1 and p2.
- **What is printed in the console at the end?** There is a simple visualisation implemented in the given code. This visualisation shows an indication of the buffer and its growth during the execution. This can give you an idea of the efficiency of the workers. At the end of the execution the main program will print and log these information: students number, group number, number of workers, a visualisation of the buffer growth, total number of tasks processed by the workers (**this must be equal to the number of provided challenges**

plus one, i.e. 501), execution time for the sequential version and concurrent versions.

- **Parameters**: There are two classes defined for parameters. The values of the *WorkingParams* must be determined by the student(s) in the final submission. The values of the *FixedParameters* must not change in the final submission. However, students can change the values of FixedParams for their own experiments. For example, lowering *maxNumOfChallenges* can give a quick result. But, the final submission must keep the original values.

Extra notes:

- The implementation requires different tuning depending on your own processor. A lot of testing is required.
- The delivery should be in a zip file named as: *studentnumber1_studentnumber2_infcoc.zip* (the name saved online will add also the name of the account that submitted the assignment).
- In the delivery there should e no trace of any runtime compiled file. Solutions should be cleaned.
- Error at runtime as “null pointer exceptions”, or any other runtime error from unhandled data, rewritten memory or other, is not considered a sufficient assignment even if it is a very easy fix.
- Altering the original data in the delivery as well as adding external resources of any kind is forbidden. Variation of the existing parameters is encouraged, but should be put back in the original values once done.
- Be aware that the delivery is final, you can deliver anytime till the end deadline. Resubmission is not possible (consider it an exam in class).