Quantization and De-quantization with Multi-threading

1. Goals

This assignment is designed to help you:

a) get familiar with multi-threaded programming using pthread
b) get familiar with mutual exclusion enforcement using mutexes
c) get familiar with synchronization using semaphores

2. Background

Quantization and de-quantization are basic modules in image/video compression aiming to achieve a compact data representation of the original data and then restore them. Considering a simplified scenario, the **camera** loads captured frames with float values (between 0 and 1) into the **cache**. A captured frame is usually represented by a m×n matrix where m is the number of rows and n is the number of columns. For simplicity, it is flattened into a one-dimension vector before being loaded into the cache. Next, the **transformer** extracts the flattened frames in arrival order from the cache into the **temporary frame recorder** and then compresses the extracted frame by quantization and de-quantization in place in the temporary frame recorder. Last, to view the distortion of compression, the **estimator** calculates the mean squared error (MSE) between the original frame and the compressed frame (the result of quantization and de-quantization), and then deletes the original frame from the cache to save space for new frames.

3. Requirements

In this assignment, you are required to design and implement a multi-threaded C/C++ program to simulate the above simplified scenario as follows.

- You are provided with a function `generate_frame_vector()` to generate flattened frames. The prototype of the function is given below:
```
    double* generate_frame_vector(int length);
```
  The parameter `length` is the length of the flattened frame which equals m×n. In this assignment, we set m = 4, n = 2 and thus l = 8. The return value of the function is a one-dimension array with length `length`, noted as a pointer of the type double.
- The **cache** is implemented as a first-in-first-out (FIFO) queue. There are 5 entries in the cache and each entry can store a flattened frame (a linear array). You need to implement your own queue data structure and its related functions which means that you cannot use any data structure provided by the C/C++ library.
- The **camera** thread is created to load flattened frames into the cache. Specifically, the camera repeatedly calls generate_frame_vector() to generate **one** flattened frame at a time and loads the frame into the cache until the cache is full. It takes the camera **interval** seconds to load a frame into the cache. When the cache is full, the camera has to wait for the signal from the **estimator** after it deletes a frame from the cache. After a certain number

of frames are generated, the function generate_frame_vector() returns NULL and the camera exits.

- The **transformer** thread is created to perform data compression. Specifically, after the camera loads a frame into the cache and signals the transformer, the transformer extracts a flattened frame in arrival order, one at a time, from the cache into the temporary frame recorder and then compresses the extracted frame by quantization and de-quantization in place in the **temporary frame recorder**. It takes the transformer 3 seconds to compress the extracted frame in the temporary frame recorder. Then, the transformer signals the estimator to compute the MSE.

  In the **transformer** thread, you are provided with a function `compression()` to compress data. The prototype of the function is given below:

  ```
  double* compression(double* frame, int length);
  ```

  The compression of data consists of two steps, namely quantization and de-quantization. The following vector A represents a flattened frame with float values (between 0 and 1).

  $$A = (a_1, a_2, \dots, a_l)$$
  $$0.0 \le a_i \le 1.0, \qquad i = 1,2,\dots,l$$

  The data quantization on A is expressed as follows.

  $$a_i' = round(10.0 \times a_i)$$

  Similarly, the de-quantization is expressed as follows.

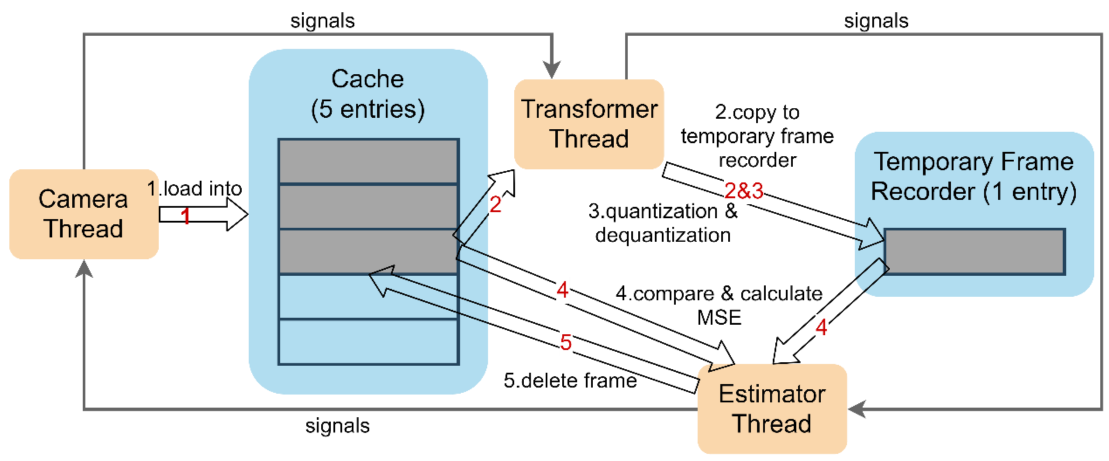
  $$a_i'' = \frac{a_i'}{10.0}$$

- The **estimator** thread is created to calculate the MSE between a pair of original and compressed frames. Specifically, after the transformer compresses the extracted frame in the temporary frame recorder and signals the estimator, the estimator calculates the MSE between the compressed frame in the temporary frame recorder and the corresponding original frame in the cache. Finally, the estimator prints the MSE and deletes the original frame in the cache. We do not require a fixed runtime for MSE calculation and it depends on your implementation.

  In the **estimator** thread, regarding to the original frame A and the compressed frame A'' with size l, MSE between them is calculated as follows.

  $$MSE(A, A'') = \sum_{i=1}^{l} (a - a'')^2 / l$$

- Termination condition. The program terminates ONLY when ALL frames are processed.
- Since all the threads need to access some shared resources such as a variable or a data structure, the code segment for accessing these shared resources which is a critical section must be protected with mutex.
- There are synchronization issues among camera, transformer and estimator. You need to use semaphore to fulfill such synchronizations. NO busy waiting can be used.
- The figure below may help you understand the problem:


signals
signals
Cache
(5 entries)
Transformer Thread
2.copy to temporary frame recorder
Temporary Frame Recorder (1 entry)
Camera Thread
1.load into
1
2
2&3
3.quantization & dequantization
4
4.compare & calculate MSE
4
5
5.delete frame
Estimator Thread
signals

4. Important Notes

- You are required to implement your solution in C/C++ on Linux (other languages are NOT allowed). Your work will be tested on the Linux server gateway.cs.cityu.edu.hk.
- The functions `generate_frame_vector()` and `compression()` are provided in files generate_frame_vector.c and compression.c separately. Compile them along with your source code (See Input/Output Sample below). Do NOT copy the code to your own file. In fact, you only need to declare its prototype (see Requirements above) in your code, outside main() just like declaring a normal function.

5. Input/Output Sample

Your program needs to accept one integer input, `interval` in seconds, as shown below. The following is a sample input/output with 10 frames generated.

```
> g++ generate_frame_vector.c compression.c 51234567.cpp -lpthread -o
51234567
>./51234567 2
mse = 0.000541
mse = 0.000578
mse = 0.001112
mse = 0.000924
mse = 0.001083
mse = 0.000901
mse = 0.000354
mse = 0.001406
mse = 0.000589
mse = 0.000377
```

6. Marking Scheme

You program will be tested on our CSLab Linux server (gateway.cs.cityu.edu.hk). You should describe clearly how to compile and run your program in the readme file. **If an executable file cannot be generated and running successfully on our Linux servers, it will be considered**

as unsuccessful. If the program can be run successfully and output is in the correct form, your program will be graded according to the following marking scheme.

- Design and use of multithreading (**15%**)
  - Thread-safe multithreaded design and correct use of thread-management functions
  - Non-multithreaded implementation (0%)
- Design and use of mutexes (**15%**)
  - Complete, correct and non-excessive use of mutexes
  - Useless/unnecessary use of mutexes (0%)
- Design and use of semaphore (**30%**)
  - Complete, correct and non-excessive use of semaphore
  - Useless/unnecessary use of semaphore (0%)
- Degree of concurrency (**15%**)
  - A design with higher concurrency is preferable to one with lower concurrency.
  - An example of lower concurrency: only one thread can access the cache at a time.
  - An example of higher concurrency: the **camera** thread and the **transformer** thread can access the cache at the same time but works on different frames (cache locations).
  - No concurrency (0%)
- Program correctness (**15%**)
  - Complete and correct implementation of other features including
    - correct logic and coding of thread functions
    - correct coding of queue and related operations
    - passing parameters to the program on the command line
    - program input and output conform to the format of the sample
    - successful program termination
  - Fail to pass the g++ complier on our Linux servers to generate a runnable executable file(0%)
- Programming style and documentation (**10%**)
  - Good programming style
  - Clear comments in the program to describe the design and logic (no need to submit a separate file for documentation)
  - Unreadable program without any comment (0%)

7. Submission

- This assignment is to be done individually or by a group of two students. You are encouraged to discuss the high-level design of your solution with your classmates but you **must implement the program on your own**. Academic dishonesty such as copying another student's work or allowing another student to copy your work, is regarded as a serious academic offence.
- Each submission consists of two files: a source program file (.cpp/.c) and a readme text file (.txt), telling us how to compile your code and possible outputs produced by your program.
- Write down your name(s), eid(s) and student ID(s) in the first few lines of your program as comment.

- Use your student ID(s) to name your submitted files, such as 5xxxxxxx.cpp and 5xxxxxxx.txt for individual submission, or 5xxxxxxx_5yyyyyyy.cpp and 5xxxxxxx_5yyyyyyy.txt for group submission. You may ignore the version number appended by Canvas to your files. Only one submission is required for each group.
- Submit the files to Canvas. As far as you follow the above submission procedure, there is no need to add comment to repeat your information in Canvas.
- The deadline is **11:00 am, 12-APR-2022 (Tuesday**). No late submission will be accepted.

8. Question?

- This is not a programming course. You are encouraged to debug the program on your own first.
- If you have any questions, please submit your questions to the Discussion board "Programming Assignment #3".
- To avoid possible plagiarism, do not post your source code on the Discussion board.
- If necessary, you may also contact Mr. FENG Xingbo at xingbfeng3-c@my.cityu.edu.hk.