Video Binary Quantization with Multi-threading

1. Goals

This assignment is designed to help you:

a. get familiar with multi-threaded programming with pthreads
b. get familiar with mutual exclusion enforcement with mutexes

2. Background

Quantization is a basic module in image/video compression aiming to achieve a compact data representation of the original data. Considering a simple scenario of video capture and quantization, the ***camera*** transmits captured frames with float values (between 0 and 1) to the ***quantizer*** for quantization, in which the float values will be rounded to 0 or 1. The captured frame is usually represented by a $m \times n$ 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 camera cache. Since the cache has limited size and could hold only a few frames, the camera feeds data into the cache for quantization only when there is available space in the cache. Meanwhile, the quantizer will quantize the frames stored in cache, print the quantization results, and delete the quantized frames 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 using pthreads and mutexes to simulate the simplified binary quantization 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 l);
  ```
  The parameter `l` is the length of the flattened frame which equals $m \times n$. In this assignment, we set $m = 4$, $n = 2$ and thus `l` = 8. The function returns a one-dimension array with length l, denoted as a pointer to the type *double*.
- The camera **cache** is implemented as a first-in-first-out (FIFO) queue. Each entry in the cache can store a flattened frame (a linear array) and the cache size is **5** frames. **You need to implement your own queue data structure and its related functions** which means that you cannot use the queue data structure provided by the C/C++ library.
- The ***camera*** thread is created to load frames into the cache. Specifically, the camera thread checks the cache every `interval` seconds, which is a program input parameter. If the cache is **not full**, the camera thread calls *generate_frame_vector*() to generate one flattened frame and loads the frame into the cache. Otherwise, the camera thread retries after `interval` seconds. After a certain number of frames are generated, the function *generate_frame_vector*() returns NULL and the camera thread ends.
- The ***quantizer*** thread is created to perform data quantization in the cache. Specifically, the quantizer thread continuously checks the cache. If the cache is **not empty**, the quantizer thread quantizes one frame in the cache in arrival order. Note that the quantization has to be done in place, which means that the quantizer thread modifies the data in the cache directly. The quantization takes **3** seconds per frame. After quantization, the quantizer thread prints the quantization result on the screen (see Input/Output Sample below) and deletes the quantized frame from the cache. If the cache is empty, the quantizer thread checks again after **1** second. When the quantizer thread finds the cache empty in **3** consecutive checks, it is assumed that all frames are quantized and the quantizer thread ends.

- The data quantization operation is defined as follows:
$$\mathrm{A} = (a_1, a_2, \ldots, a_l)$$
$$0.0 \le a_i \le 1.0, i = 1,2,\ldots,l$$
where the vector *A* represents a flattened frame and the quantization on *A* is expressed as follows:
$$a_i' = \begin{cases} 0 & a_i \le 0.5 \\ 1 & a_i > 0.5 \end{cases}, 1 \le \mathrm{i} \le l$$
- Since both threads need to access some shared resources such as variable or data structure, the code segment for accessing these shared resources which is a **critical section must be protected with a mutex**.

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 cs3103-01.cs.cityu.edu.hk.
- The function *generate_frame_vector*() is provided in the file *generate_frame_vector.cpp*. Compile it 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, and **the output must be formatted as shown below.**

```
$ g++ generate_frame_vector.cpp 51234567.cpp -lpthread -o 51234567
$ ./51234567 2
1.0 0.0 0.0 0.0 1.0 1.0 0.0 1.0
0.0 1.0 1.0 0.0 1.0 0.0 0.0 1.0
1.0 1.0 1.0 1.0 0.0 1.0 0.0 1.0
```

6. Marking Scheme

Your program will be tested on our CSLab Linux servers (cs3103-01, cs3103-02, cs3103-03). 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, your program will be graded according to the following marking scheme.

- Design and use of multithreading (**30%**)
  - Thread-safe multithreaded design and correct use of thread-management functions
  - Non-multithreaded implementation (0%)
- Design and use of mutexes (**30%**)
  - Complete, correct and non-excessive use of mutexes
  - No or useless/unnecessary use of mutexes (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: both threads can access the cache but works on different frames (cache entries) at a time.
  - No concurrency (0%)

- Program correctness (**15%**)
  - Complete and correct implementation of other features such as
    - correct logic and coding of thread functions and other functions such as quantization
    - correct coding of queue and related operations
    - passing parameter 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 CSLab 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. Bonus (20%)

Students who want to earn the bonus need to have their programs to satisfy the following additional requirements and indicate this in the readme file.

- Support two quantizer threads to allow two frames in the cache to be quantized concurrently.
- The output order of the quantized frames on the screen must be consistent with the input order.

8. 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) and a readme text file (.txt), telling us how to compile your program 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, 20-MAR-2020 (Friday)**. No late submission will be accepted.

9. 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 #2".
- To avoid possible plagiarism, do not post your source code on the Discussion board.
- If necessary, you may also contact Mr. WANG Shurun at srwang3-c@my.cityu.edu.hk or Mr. WEN Zihao at zihaowen2-c@my.cityu.edu.hk.