CS3103: Operating Systems
(Spring 2022)
Programming Assignment 2
Video Binary Quantization with Multi-threading

1 Goals

This assignment is designed to help you:

a. Get familiar with multi-threaded programming with pthread
b. Get familiar with mutual exclusion enforcement with mutex

2 Background

Quantization is a basic module in image/video compression aiming to achieve a compact data representation compared with original data. Considering a simple scenario in video capture and quantization, the ***camera*** could transmit the captured frames with float values (between 0 and 1) to the ***quantizer*** for quantization. The captured frames are usually represented by an $m \times n$ matrix where m is the number of rows and the n is the number of columns. For simplicity, it should be flattened into a 1-d vector before being pushed into the cache. Since the camera cache has limited size and could hold only a few frames, the camera should feed data into the cache for quantization only when there is available space in cache. Meanwhile, a quantizer will quantize the frames stored in cache, print the quantization result, and delete the quantized frames from cache to save cache space for new frames.

3 Components and Requirements:

You are required to using pthread, design and implement **a camera cache**, **a camera thread** and **a quantizer thread in C/C++ on Linux** (other languages are not allowed). Mutual exclusion **must** be done with **mutex** provided in library <pthread.h>. Your work will be tested on the Linux server (gateway.cs.cityu.edu.hk).

3.1 generate_frame_vector.c

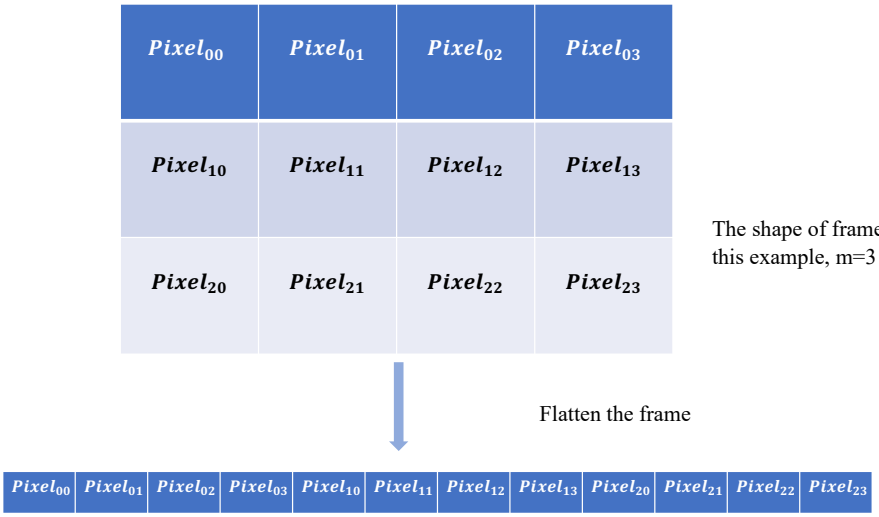
We provide a file *generate_frame_vector.c* which contains a function *generate_frame_vector*() to generate flattened frames. Compile it along with your source code (See Input/Output Sample). The prototype of the function is given below:

```
double* generate_frame_vector(int length);
```

The parameter *length* is the length of the flatten frame set by you:

$$length = m \times n$$

where *m* is the number of row in the frame and *n* is the number of column in the frame. To facilitate data storage, frames are usually expanded into a 1-d array. The illustration is shown below. The return value of the function is a 1-d array with *length*, denoted as a pointer of type *double*. The value of the variable ***length*** should **be assigned by you**, which is the product of frame shape(m, n) you want.



After the flatten operation, the frame become a 1d-array and the length is m*n which is 12 in our example.

3.2 camera cache (you need to implement)

The **camera cache** is implemented as a FIFO (first-in-first-out) queue. The basic element of FIFO is a frame vector and the cache can store 8 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.

3.3 camera thread (you need to implement)

The **camera thread** is responsible to manage the cache.

- Camera thread will periodically detect whether the cache is full.
- If **not full**, camera thread will call ***double *generate_frame_vector(int length)*** to generate **ONE** flattened frame and loads the frame into the cache.
- If the cache is **full**, the camera thread will sleep for a while to wait for the vacancy shown in cache due to the execution of the quantizer thread. (Quantizer thread will delete the frame in cache after the quantization.) The sleep time should be defined by variable named ***interval*** in your code which can be assigned value by you. Also, the ***interval*** should be given as a command line argument while running your program, details are in the input/output sample part.
- There is a parameter MAXIMUM_FRAME_NUM in generate_frame_vector.c. After generating the maximum number of frames, the function *generate_frame_vector(int length)* return NULL. The camera thread will end when it receives the NULL signal.

3.4 quantizer thread (you need to implement)

The **quantizer thread** is responsible to quantize the value in cache.

- First, quantizer will detect whether the cache is empty.
- If not, quantizer thread would quantize **ONE** frame in the cache in arrival order. Note that the quantization should be done in place, which means you should modify the data in the cache directly. After quantization, the thread should print the quantization result of that particular frame (refer to the input/output sample below) and then delete the quantized frame from cache. In reality, the quantization will cost some time, you need to use *usleep()* function in quantizer thread to simulate the time taken for quantization.
- If the cache is empty, the quantizer thread checks again after 1 second. After the maximum number of consecutive attempts is exceeded, it is assumed that all frames are quantized and the quantizer thread ends. The maximum attempts number is a custom value defined by you like 3.
  - The data quantization operation could be defined as follow:

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

The vector *A* represents one flattened frame. The quantization on *A* could be expressed as below:

$$a_i' = \begin{cases} 0 & a_i \le 0.5 \\ 1 & a_i > 0.5 \end{cases}, 1 \le \mathrm{i} \le l$$

3.5 Mutex (you need to implement)

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.**

3.6 Input/ Output sample

> gcc –lpthread generate_frame_vector.c Student_ID.c –o Student_ID
>./Student_ID interval

for example:
> gcc -lpthread generate_frame_vector.c 51234567.c -o 51234567
>./51234567 2
1 0 0 0
1 1 0 1
0 1 1 0
1 0 0 1
1 1 1 1
0 1 0 1

Here you can use gcc or g++ to compile the program. Your program has to accept one input argument: *interval:* a double value for how many seconds you want the

camera thread sleep. The output here means that there are 6 frames, the length of each frame is 4(m*n=4).

4 Important Notes

- You may NOT copy the codes from *generate_frame_vector.c* to use the *generate_frame_vector(int length)* function. In fact, you only need to declare it in your codes, outside the *main()* like declaring a normal function. For example,

  double generate_frame_vector(int length);*

- The given code will generate a certain number of frame vectors and after that, the *generate_frame_vector(int length)* function would return NULL.
- The number of frames that the cache could load at most is **fixed as 8** in this assignment.
- The value in cache should be modified in place.
- In reality, the operations of camera thread and quantizer thread will take some time to finish. We use usleep() function in two threads to simulate the operation time. For camera thread, the sleep time is ***interval*** ms which should be a custom value given by you. For quantizer thread, the sleep time should be 3000ms.

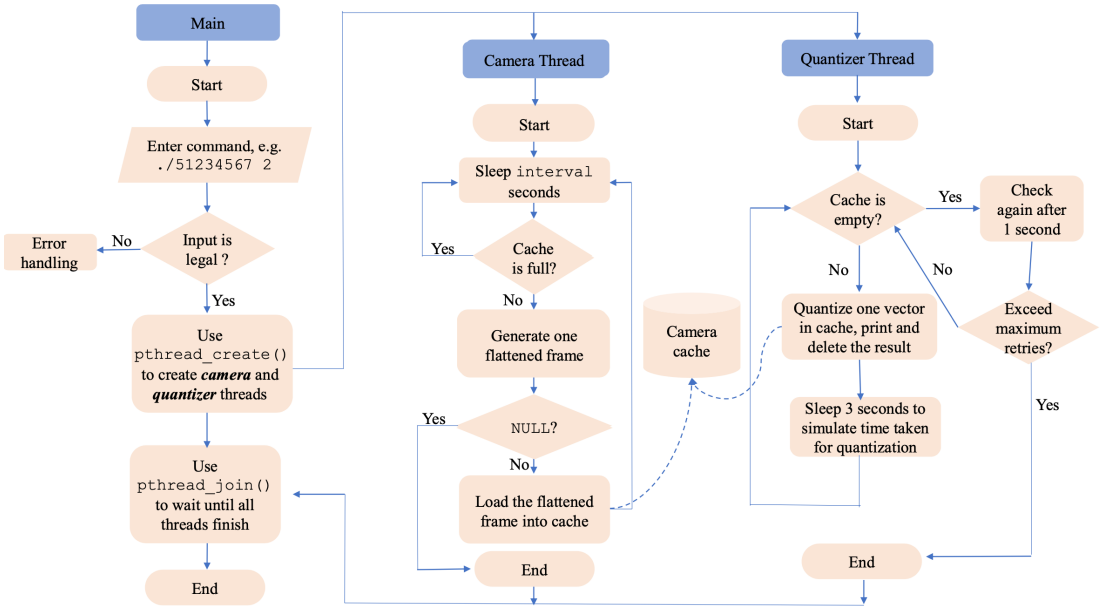
  Brief introduction to usleep():

  #include <unistd.h>
  int usleep(useconds_t useconds)

  The usleep() function suspends the calling thread until argument *useconds* have elapsed. The value of argument *useconds* is the number of **microseconds** that you want to process to sleep for, which must be less than 1000000.

5 Problem flowchart

The whole process is shown in the flowchart below.



6 Marking scheme

1) Design and use multi-thread (30%)
   a. Create and terminate camera thread
   b. Create and terminate quantizer thread
   c. Implement quantization correctly
   d. Correct output print
   e. No multi-thread implementation (0%)
2) Design and use mutex (30%)
   a. Complete, correct and non-excessive use of mutexes
   b. No or useless/unnecessary use of mutexes (0%)
3) Program correctness (20%)
   a. Complete and correct implementation of features such like:
      - 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.
3) Programming style and documentation (20%)
   a. Good programming style
   b. Clear comments to describe the design and logic (no need to submit a separate file for documentation)
   c. Unreadable program without any comment (0%)

7 Bonus (15 points):

This part is for students who want to get a higher grade in this programming assignment. Your work needs to satisfy the following additional requirements and indicate this in the readme file.

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

Hints:

1) There should be two quantizer threads operating on two different cache frames simultaneously.
2) In-order means that the output must be in the same order of frames in the camera cache. For example, if thread1 reads cache_frame_0 and the other one, thread2, reads cache_frame_1, the quantized output frames should be quantized_frame_0 and then the quantized_frame_1.

8 Submission

- The assignment could be finished by a group of TWO students or individually. 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 (.c) 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.
- Submission should include a source program file (.c file) named as Student_ID1_ID2.c. Besides, ID1 and ID2 is your student ID of the groups. You could also name the source file like Student_ID.c if you finished the assignment by yourself. Example, the final file could be named as 5xxxxxxx_5yyyyyyy.c or 5xxxxxxx.c. You may ignore the version number appended by Canvas to your files.
- **Only one** submission is required for each group.
- Submit the file in 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 2022.3.12

9 Questions?

- If you have any questions, please submit your questions to the Discussion board "Programming Assignment #2" on Canvas.
- To avoid possible plagiarism, do not post your source code on the Discussion board.
- If necessary, you may also contact Ms Weiwei Fu at weiweifu2-c@my.cityu .edu.hk