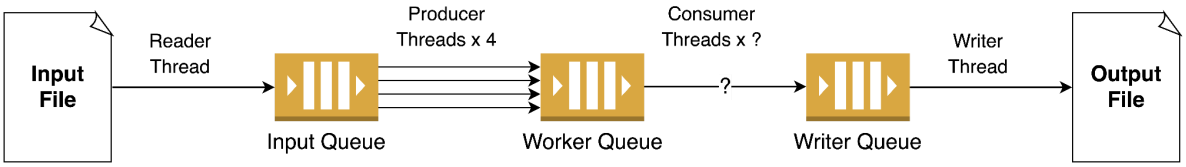
CS342301 2021 Operating System
Pthreads Programming Assignment: Producer-Consumer Problem
Deadline: 2022/1/16 23:59

I. Goal

1. Understand how to work in the Linux environment.
2. Understand how multi-thread programming can be done with the pthread library.

II. Description

The producer-consumer problem is a classic programming exercise in computer science. In this project, you will be given some Items, each Item contains an opcode, a key and a value. The producer and consumer are responsible to perform a transform function on the Item's value respectively. Different transform functions are performed based on the Item's opcode.



The program starts with 1 Reader thread, 4 Producer threads, 0 Consumer thread and 1 Writer thread. The buffered size of the Input Queue, Worker Queue and Writer Queue is predefined in the main.cpp file.

The program also starts with 1 ConsumerController thread, which controls the number of consumers. When the number of items in the Worker Queue exceeds the 80% of the Worker Queue size, the number of consumer threads should be scaled up by 1. On the other hand, when the number of items in the Worker Queue falls behind 20% of the Worker Queue size, the number of consumer threads should be scaled down by 1.

The reader thread should end after reading all Items from the input file; the writer thread should end after writing all Items from the output file. The main program should wait for reader thread and writer thread to complete.

III. Assignment

a. Code Structure

1. TSQueue (ts_queue.hpp): A thread safe queue that allows multi-thread access.
   - enqueue: Add an element into the queue.
   - dequeue: Remove the first element from the queue and return the element.
   - get_size: Returns the number of elements in the queue.
2. Reader (reader.hpp): Reader runs in a thread that reads Items from the input file

then puts Items into the Input Queue.

3. Transformer (transformer.hpp): Defines the producer and consumer transformation functions.
4. Producer (producer.hpp): Every Producer runs in a thread that takes Item from the Input Queue, applies the Item with the Transformer::producer_transform function, then puts the result Item into the Worker Queue.
5. Consumer (consumer.hpp): Every Consumer runs in a thread that takes Item from the Worker Queue, applies the Item Transformer::consumer_transform function, then puts the result Item into Output Queue.
6. ConsumerController (consumer_controller.hpp): ConsumerController runs in a thread that controls the number of consumer threads dynamically.
7. Writer (writer.hpp): Writer runs in a thread that reads Item from the Output Queue then writes Items into the output file.

b. Implementation

1. TSQueue: Implement a thread safe queue to support enqueue, dequeue, getsize operations.
   Note: You should use the pthread conditional variable to avoid busy waiting on the queue full when enqueuing, or the queue empty when dequeuing.
2. Producer: Create 4 Producer threads on the program starts. When each Producer thread starts, it repeats to:
   - Take an Item from Input Queue.
   - Use Transformer::producer_transform to perform transform on the Item's value.
   - Put the Item with new value into the Worker Queue.
   - Do not modify the Item's key and the Item's opcode.
3. Consumer: When the Consumer thread starts, it repeats to:
   - Take an Item from the Worker Queue.
   - Use Transformer::consumer_transform to perform transform on the Item's value.
   - Put the Item with new value into the Output Queue.
   - Do not modify the Item's key and the Item's opcode.
4. ConsumerController: Create a ConsumerController when the program starts. When the ConsumerController thread starts, is repeats to:
   - In the beginning, no Consumer will be created by ConsumerController.
   - Check Work Queue status periodically. (The period is defined in main.cpp as CONSUMER_CONTROLLER_CHECK_PERIOD).
     - When the Worker Queue size exceeds *high_threshold*, create a new Consumer thread.
     - When the Worker Queue size falls behind a *low_threshold*, cancel the newest Consumer (by calling Consumer->cancel).
       Note: You should beware that the consumer cannot be canceled at any time since it might be performing transformation to some Item. Use

`pthread_setcanceltype` and `pthread_setcancelstate` to prevent the thread being canceled at any time.
    - ***Once you have created a Consumer, make sure there is at least one Consumer until the program ends.***
    - Maintains the list of running Consumers in ConsumerController::consumers.
5. Writer: Create a Writer when the program starts. On the Writer thread starts, it repeats to:
    - Take an Item from the Output Queue.
    - Write the Item to output file with the same format as the input file ({key} {value} {opcode}).
6. main.cpp: Implement your main function, the main function ends after reader thread and writer thread to complete.

c. Notes & Hints

1. Search for all `TODO:`, that's all you need to do.
2. It is allowed to implement code in .hpp files or in separate .cpp files, but you should make sure that the Makefile is working properly. We will compile your code with a single `make` command.
3. All `*_test.cpp` files are only for unit testing. You can decide whether to use them for verifying or not. We will not use them to verify your code.
4. You should output a message to *stdout* when scaling up and scaling down that shows the number of consumers before and after scaling. For example: “Scaling up consumers from 1 to 2” or “Scaling down consumers from 5 to 4”.
5. You can implement your own transformer.cpp and input file. We will replace the transformer.cpp file when verifying your code.

d. Compile, Run and Test

1. Run `cp -r /home/os2021/share/NTHU-OS-Pthreads ~/.` to copy the project folder to your home directory.
2. Run `cd ~/NTHU-OS-PThreads` to enter the project folder. Then implement your solutions.
3. Run `./scripts/auto_gen_transformer --input ./tests/00_spec.json --output transformer.cpp` to generate transformer.cpp file.
4. Run `make clean` to clean executable files.
5. Run `make docker-build` to compile your source code. (Note that if you are developing on a host with gcc and pthread installed, then run `make` command to compile your source code).
6. Run `./main 200 ./tests/00.in ./tests/00.out` to run your main program and generate your output result to the output file.
7. Run `./scripts/verify --output ./tests/00.out --answer ./tests/00.ans` to verify your answer.

8. We already provide 2 test cases. You can change and test other test cases by changing all 00 in the commands to 01 above. You can also add your own test cases.

IV. Grading

a. Implementation Correctness - 50%

1. **Must implement this homework through the class we supplied.**
2. **Must use Pthread library to implement multi-thread programming.**
3. **No Plagiarism. If you copied someone's code and cannot explain your implementation, you will get 0 points.**
4. **Late submissions will not be accepted. Refer to the course syllabus for detailed homework rules and policies.**
5. **Your code should be placed directly under your home directory with the directory name `NTHU-OS-Pthreads`, else you will get 0 points on this part.**

b. Experiment - 20%

Do the following experiments and write down your experiment result in your report.

1. Different values of CONSUMER_CONTROLLER_CHECK_PERIOD.
2. Different values of CONSUMER_CONTROLLER_LOW_THRESHOLD_PERCENTAGE and CONSUMER_CONTROLLER_HIGH_THRESHOLD_PERCENTAGE.
3. Different values of WORKER_QUEUE_SIZE.
4. What happens if WRITER_QUEUE_SIZE is very small?
5. What happens if READER_QUEUE_SIZE is very small?

c. Report - 20%

1. Cover page, including team member list, team member contributions.
2. Explain your implementation as requested.
3. Explain what experiments you have done and what are the results. You are encouraged to do more experiments.
4. What difficulties did you encounter when implementing this assignment?
5. Any feedback you would like to let us know.
6. Name your report **Pthreads_Report_{Group Number}.pdf** and upload to eeclass.

d. Demo - 10%

1. We will ask several questions about your codes.