ITP 30002-XX Operating System, Spring 2020

Homework 1

Your name : Lee EunJong and Student ID : 21700556, Email address : 21700556@handong.edu

1. Introduction

Homework 3 is that solve the tsp by using multi-thread. To solve tsp that using multi-thread, I used 3 types of threads. One is the producer thread that makes prefix and sends prefix to consumer thread. Another is the consumer thread that receives prefix from the producer thread and process remain parts. The other is the main thread that creates the consumer thread and producer thread. The main thread also receives user command and generate response about the user command in the middle of solving. To give and receive the prefix, communication between consumer threads and producer thread is needed. To generate response about user command, communication between main thread and consumer threads is needed. To communicate between threads, threads just read and write the data in the same process. However, threads should not read and write the same data at the same time, because it can make unwanted result. I implemented mutual exclusion to protect threads that can make side effects approach critical section at the same time.

2. Approach

Workflows of each threads

Main thread :

1. main threads get thread counts and file path from argv.
2. open gr--.tsp file and make map.
3. Initialize bounded buffer and reader writer lock structure. (bounded buffer and reader writer lock structure will be dealt with in detail later)
4. Make producer thread and consumer threads and store thread's ids in array(prod array and cons array)
5. Wait user command
6. If main thread receives user command, main thread get reader lock and generate response about the command. then, Release reader lock.
7. If producer thread ends generating prefix(end_produce_flag is 1), wait all threads are terminated and terminate process.

   Else go to 5

Producer thread :

1. Make prefix by using recursion.

   if unchecked city count is 11, send path to bounded buffer(using enqueue function).

   else traverse all possible cases recursively
2. If all possible prefix is made, then end_produce_flag change 1(originally, end_produce_flag is 0).

consumer thread :

1. Get prefix from bounded buffer(using dequeue function)
2. Check this consumer thread's id(using pthread_self function) and find index of same thread id from the cons array. The index is write_to variable. Write_to variable means that this thread will access i-th(i=write_to) element in shared array.
3. Consumer thread process remain parts that except prefix.

   if unchecked city count is 0, get writer lock and update min, best path data in thread_min[write_to] and thread_best_path[write_to]. Release writer lock.

   else traverse all possible cases recursively.
4. If produce end flag is 1, terminate thread.

   Else go to 1.

Shared data and management

To communicate between main thread and consumer threads, I used shared array(thread_best_path, thread_min, thread_sub_count, etc). i-th consumer thread can only access i-th element of the array, so each consumer threads can write without interference from other consumer threads.

However when main thread want to read consumer thread's data, mutual exclusion between main thread and consumer threads is needed. To implement that I used reader lock and writer lock.

To implement reader lock, I used condition variable. In reader lock function, checks that some threads which are writing or reading are exist. if exist, reader lock waiting. Else(no writing or reading thread), we can access user consumer's data by access shared array(thread_best_path, etc). after reading, in reader unlock function send signal to

writer condition variable or reader condition.

To implement writer lock, I also used condition variable. In writer lock function, checks whether some threads which wants to read or reading. If exist, writer lock wait. Else, consumer thread can write at the shared array. and then send signal to reader condition variable.

I used reader lock and writer lock for each consumer thread, because if there are one reader writer lock and shared by consumer threads, only one thread get mutex_lock and others have to wait to get mutex_lock. It can decrease program's throughput, so I used several reader and writer lock.

To communicate between producer thread and consumer thread, I used bounded buffer with condition variable. When producer thread wants enqueue data, bounded buffer get mutex_lock and checks whether capacity is full or not. If buffer is full, wait. Else, insert data to bounded buffer and release mutex_lock. When consumer thread wants deque data, bounded buffer get mutex_lock and checks whether buffer is empty or not. If buffer is empty, wait. Else, bring data from bounded buffer and release mutex lock.

To generate response about the user command

To process Stat command

Main process gets reader lock and updates min cost data, best_path, total checked count. And then show those stats to user. Then, release reader lock.

To process Threads command

This part is similar with the part of processing stat command

To process Num N(N is number) command

if original thread count is bigger than N, just increase thread count(max_count variable), and create new threads.

if original thread count is smaller than N, cancel (original thread count - N) threads. To cancel threads, I used pthread_cancel function. we can change cancel state and type. Default type and state of pthread_cancel is thread cancel enable and deffered type. To use deffered type, I set cancellationpoint by using pthread_testcancel function. Then I had to enqueue prefix which is from terminated thread to bounded buffer. However if bounded buffer is full, user need to wait, so I make another buffer to store canceled thread's prefix and to send to bounded buffer.

3. Evaluation

Criteria

1. If command's response is act well, I fulfilled requirement about interactive user command part.
2. If my program's throughput is increase as the thread count increases, my program works well in multi-thread environment.

When I enter stat and threads command, program shows proper data well. When I enter Num N, program decrease threads or increase threads well. Thread count change can be checked by using ps –p (pid) -T command.

To check 2nd criteria I checked throughput for 30seconds when thread count is 4. And I checked throughput for 30seconds when thread count is 8.(I measured 5time for each condition). Follow table is the result

throughput test			
	1threads	4 threads	8 threads
for 30 seconds	142634726	155650010	175697665
	4thread/1threads	8threads/1threads	
ratio	1.091249055	1.231801469	

We can show the throughput of my program is slightly increased as thread count is increases. I think one reason of not good increasement of throughput is using bounded buffer. Only one thread can access bounded buffer at the same time, and the other threads are wait, so I think bounded buffer can decrease throughput.

4. Discussion

When I implement multi-thread program, I met a lot of unwanted dead locks. I feel why multi-threading is not easy and why dead lock finding tool is needed.

As I mentioned at evaluation part, I learned that many use of mutex_lock can decrease the program's throughput. When I used only one reader writer lock and threads count is 8, the throughput of the program is 1/8 throughput that thread count is 1. I also learned that reduce critical section as possible as can increase throughput of the program.

I got curious that when multi-thread program is effective than multi process program. I also wonder how find effective thread number to achieve the highest efficient(would the more threads the better?).

I think several producer is exist, and several bounded buffer is exist, the performance of this program is increase.

5. Conclusion

I implement multi-thread program that solve the tsp problem. To solve tsp problem, communication with threads are needed. To communicate with threads, I had to implement mutual exclusion. To implement mutual exclusion, I used bounded buffer and reader and writer lock. I learned how to use mutex and condition variable. I also dealt with thread create and cancel and wait threads.