ITP 30002-02 Operating System, Spring 2020

Homework 3

Eunjun Jang (21800633), jangej1031@naver.com

1. Introduction

Traveling Salesman Problem (TSP) is a famous NP-Completeness problem that finding the shortest path traversing all cities.

In this paper, I will introduce my solution about TSP using multi-threading in the Linux environment using pthread library.

To simplify the problem, I assumed that it is an undirected graph. Receiving the length of the file and path containing the graph adjacency matrix information is done through the Command Line Interface (CLI). Multi-threaded TSP (MTSP) has a producer-consumer design. In the Producer thread, while searching the path, a subtask is created at the point where the number of cases becomes 11!. The producer thread puts the subtask in bounded buffer. The subtask is retrieved from the bounded buffer and the remaining path is searched by consumer thread. Producer and consumer are defined in a 1:N relationship. It is implemented thread-safe using mutex lock and semaphore. The main thread is designed to receive and process commands from the user for some information.

2. Approach

First of all, I will define some symbols. After that, I will describe detail implementations of MTSP.

m is adjacency matrix which contains weights for each path. N is number of cities. T is maximum number of consumer thread. Through user command, T can change in this bound ($1 \leq T \leq 8$). Let $task$ be the TSP data composed of N size array of current path also N size array which represents city was visited, and current distance. min_path is global shortest path and min_dist is its distance. gc is global count which represents all checked path.

2.1. Producer-Consumer and Bounded Buffer

$\mathcal{B}$ is bounded buffer which contains $task$ with thread-safety using semaphore and mutex lock. $\mathcal{B}$ has two semaphore which are empty and filled. The empty semaphore is for blocking enqueue operation. This semaphore posted when $\mathcal{B}$ dequeue one element. The filled semaphore is for blocking dequeue operation. This semaphore posted when $\mathcal{B}$ enqueue one element so it means consumer can dequeue one $task$ from $\mathcal{B}$.

The producer thread spawned with detached mode. The producer thread simply creates $task$s and enqueue to $\mathcal{B}$. Producing tasks is not that hard work, so just one producer thread is fine.

C represents consumer thread data which contains thread ID, processed $task$ count and in current processing $task$'s checked path count and mutex lock to synchronize with main thread. The consumer thread spawned by main thread with detached mode. Main thread has array of C which size is T. Main thread creates and initializes one of empty C element and pass through created consumer thread. Created consumer thread simply dequeue one $task$ and visits remaining cities. When consumer thread reaches to end of path, it checks if current path dist is less than current min_dist and try to update it. After this work, consumer thread makes cancellation points using pthread_testcencel(). This cancellation point will discuss in Discuss section.

2.2. Minimum Path Synchronization

min_path and min_dist and gc are global variable and consumer threads are access them to update minimum path. Therefore, they need synchronization. I just use one mutex lock. All threads acquire that mutex lock before the accessing global variables. So, min_path and min_dist and gc always guaranteed this path is optimum value.

2.3. User Command Processing

MTSP provides three commands: stat, threads and num T. These commands processed in main thread. This work processed after spawning producer threads and all consumer threads. First, stat command is printing out current min_path and min_dist and gc to the console. Second, threads command simply shows all running consumer thread and its C data. This task done with acquiring/release mutex lock in the C. Finally, num T command updates existing T value to provided T value. If provided T value is small than existing T value, main thread tries to cancel the consumer threads using pthread_cancel() function. If provided T is larger than existing T value, it simply creates more consumer threads.

3. Evaluation

In this section, I will show MTSP works properly with some experiments and result. All experiments are done in peace server in handing.

First, I will test increasing T value.

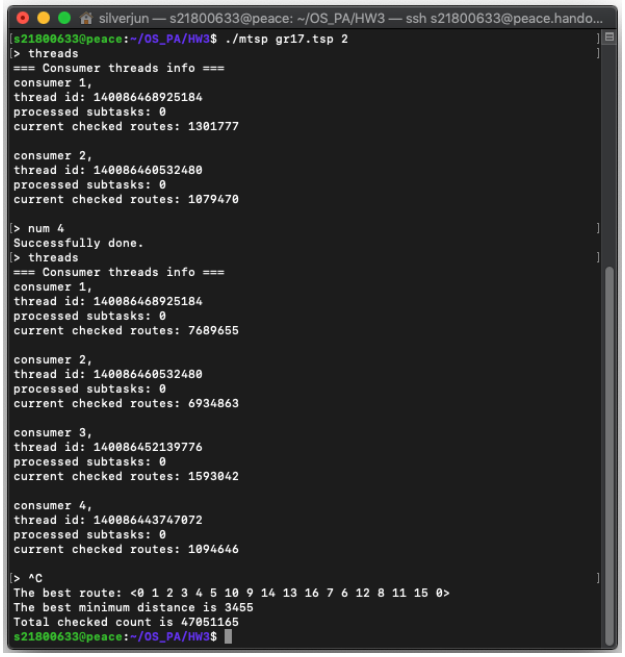

Figure 1. First Experiment Screenshot

MTSP starts with gr17.tsp file and two T size. Therefore, when I type threads, it shows two current threads. After num 4, It shows additional two threads.

Second experiment will be decreasing T value.

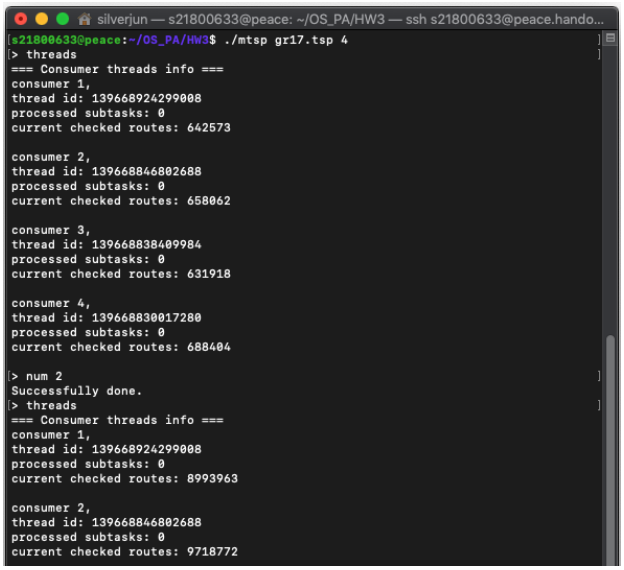

Figure 2. Second Experiment Screenshot

This result shows MTSP successfully decreasing threads. When executing MTSP, it starts with four threads. After num 2, current running threads count is decreased into two.

Third experiment will address dequeuing task after done current task.

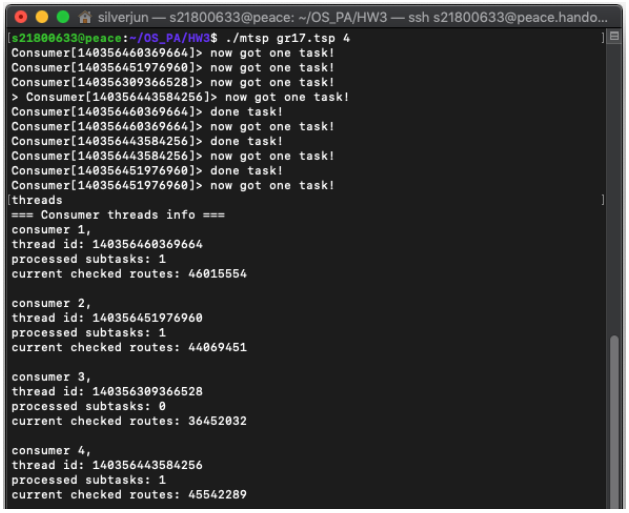

Figure 3. Third Experiment Screenshot

In this experiment, I add additional printf code in the consumer thread logic before/after TSP searching. It shows very work well.

4. Discussion

In this section, I will discuss with some experiences.

First thing to talk is cancellation points. Default pthread cancel mode is PTHREAD_CANCEL_DEFERED, it means thread only cancel when thread reaches the cancellation point. So, I created cancelation point using pthread_testcancel(). Another cancel mode is PTHREAD _CANCEL_ASYNCHRONOUS. This mode can thread cancel immediately. But this has some problem. If thread lock some mutexes, then that mutex is never can unlock. In consumer thread, it has many locking and unlocking, so I made a safe cancel point which is never suffering with deadlock.

Second is about spurious wakeup. Spurious wakeup is mean thread can wakeup even not have a signal. This is very tricky to me. Semaphore does not have this kind of thing. Comparing with semaphore, I think condition variable is not that much useful. Also checking condition using while loop is very odd thing. So, I used bounded buffer implemented with semaphore version.

5. Conclusion

In this paper, I introduced my MTSP work. While doing this homework, I was very excited. I noticed that I prefer programming assignment to writing test.