ITP 30002-02 Operating System, Spring 2020

Homework 3

Park Ye Gyeom/21600277/21600277@handong.edu

1. Introduction

The goal of this project is to learn how to create a multi-threading environment through the process of implementing mtsp.c, a multithreading server of TSP, and how to control the number of threads through the process of generating and canceling each thread. There was a problem with the use of global variables when each thread shared a global variable as a feature of shared memory during the task. In addition, in order to protect the critical section by utilizing the mutex in the code, it was also necessary to study the mutex and the critical section. The types of threads required by the task were the main threads, the producer threads, and the consumer threads. Therefor main thread operates in the main function and allows the user to enter commands, while the producer thread and consumer operate in their respective functions, creating prefixes, putting them in buffers, and taking them out of buffers to perform the process of finding routes.

The implementation of num N, the command to control the number of threads, required learning and consideration of pthread_cancel() to implement the process of inserting the canceled prefix back into the buffer. And implementation of signal handler was also needed for the program to end normally under ctrl+c and termination conditions.

2. Approach

Let's look at the workflow of the main-thread, producer-thread, consumer-thread. The task first defined the work flow of the main-thread as follows:

1. When the program starts, it generates one producer thread and as many consumer threads as the user initially entered.
2. The user enters three commands: stat, threads, and num, and when entering other commands, calls an error message and receives the command again. Make sure that you have navigated all paths before receiving the command again, and if you still have a path to navigate, enter the command and exit the program if all paths have been discovered.
3. If there is no longer a path to perform, print the best values and the total count without performing the fork(). The parent process should not end until all child processes are terminated, so the wait process in the SIGCHLD handler ensures that the parent updates the values to the last minute and outputs the best value.

The work flow of the producer thread is then as follows.

1. The producer thread produced by the main thread is operated in a function called the producer(). Producer sets the initial value of the prefix pointer to update the prefix value and the prefix value for the initial operation, insert msg in the in-array form into the buffer, and set the following peak value through the find_prefix() function.
2. Producer thread is a thread that sets a prefix value and inserts it into the buffer. If prefix is entered normally in the buffer, perform count++. If prefix is generated as much as max_num_prefix, the producer thread is terminated.

The work flow of the consumer thread is then as follows.

1. Each consumer thread generated by the main thread is assigned a room_num to store the value it has obtained.
2. The prefix produced by the propucer is imported without duplication from the buff protected with mutex. Set the path and use values through the prefix value and obtain the minimum path through the travel function.
3. The minimum value update and the number of checked routes are performed within the travel function. At this time, since multiple consumer threads use the corresponding travel function, updating the minimum value and updating the number of checked routes should be performed with only one thread allowed by the mutex.
4. If the number of prefixes processed by Consumer thread is max_num_prefix, no more prefixes are requested and the job is terminated. At this point, the consumer thread, which terminates the work at the end, raises and terminates the SIGINT signal, allowing the program to output and terminate the results normally.

Adjusting the Number of Consumer Threads

- The process of adjusting the number of Consumer threads is performed when the number N command is entered in the main thread. When the user entered num N, it was implemented considering two situations.

- 1) Increasing the number of currently active consumer threads.
- Add as many consumer threads as you want to increase through pthread_create(). The prefix of the consumer thread is not duplicated when performing the requested operation.
- 2) Decreasing the number of currently active consumer threads.
- Through pthread_cancel(), you can specify the threads you want to cancel. The function was implemented as a way to cancel the last generated thread. In order to eliminate the omission of prefix, we added the process of requeuing the prefix of the thread whose work was canceled.

3. Evaluation

Functionalities, Program Design

Main thread is implemented to operate in the main function of mtsp. When the main function is executed, the producer thread and the condumer thread are generated through the pthread_create() function, and the user can enter the command.

When entering Stat, the best results and paths obtained so far shall be printed. In order to implement this function, the value can be updated if new results are found each time in the travel() where the consumer thread operates. And it was implemented so that the user can show the best result at the moment when entering the command in the main thread. When each consumer thread updates its value, the global variables result and result_route must be updated. When updating the values of these variables, the update() was used to update the values of one consumer atomically with the mutex lock.

When inputting Threads, it is implemented to print id of current operating Consumer Thread, number of treated subtask, and count of current subtask. Id printed the value obtained through pthread_self(), the subtask count updated through count++ upon termination of travel, and the current count of subtask updated through current_count++ in travel. Thus, when the user entered the threads command, the correct function could be implemented.If a signal is generated through crtl+c during a process run, compare the values stored by each child to see if the best value and store it. And print that value and total count.

When entering Num N, two functions were needed to be implemented.

o If the user enters an N value greater than the current thread number, it generates an additional thread with pthread_create().

o If the user enters an N value less than the current thread number, the thread generated last by the pthread_cancel() function will cancel.

o You can check through threads command to make sure that this function works well.

4. Discussion

Ideas for improving mtsp.c beyond given requirements

The program was implemented in a way that the main thread waits until the process of prefixing the canceled thread through pthread_cancel() is completed. This work takes longer than I thought, so it seems inefficient. So I thought about using a sub buffer to shorten this time. Main thread is implemented so that the prefix value can be entered in the sub buffer and the user can enter the command again. We look forward to using this method to overcome this disadvantage.

5. Conclusion

By implementing Mtsp.c, I was able to experience multithreading environment a little bit. Through Pthread_create we were able to create a thread and through pthread_cancel we were able to cancel the thread operation. The critical section could be protected through pthread_mutex_lock and pthread_mutex_unlock. The lack of regularity in the working flow of Thread made it difficult to understand the flow of code. However, it is very satisfying to be able to read and understand the flow of code to some extent while investing a lot of time. And I think it helped me to organize and understand complex concepts once again, such as implementing signal handler. And it was worthwhile to use shared memory characteristics between threads to update the value more efficiently than unnamed pipes and to implement more efficient programs than ptsp.c. by exchanging prefixes with bounded buffer.