ITP 30002-01 Operating System, Spring 2020

Homework 3

Name : Min Jae Yi (이민재) **Student number** : 21800511 **Email Address** : 21800511@handong.edu

1. Introduction

The main part of the hw3 can divided into three parts making "producer", making "consumers" and getting user command through main function.

The requirement or function for the c program can be shown as below.

Producer

1-1. Create next prefix and allocate to the bounded buffer.

1-2. If all of subtasks are finished, give the termination signal to the Consumer.

Consumer

2-1. Receive the prefix by deallocating the bounded queue and calculate the best solution from the given prefix

2-2. If the user changes the number of threads increase or decrease the thread number.

2-3. Update the information about current state of the thread (Thread ID, number of subtasks processed and number of checked routes in current subtask).

Main

3-1. Create Producer thread and Consumer threads.

3-2. Get user command and print out the result.

3-3. If all of the program functions are ended join threads and terminate the process.

3-4. If ctr-c or all of subtask are finished print best solution and total number of checked routes.

Others

4.1 Bounded buffer should guarantee mutual exclusion between the consumers and producer.

Table 1 : problem define

Brief approach for each problem.

a. 1-1 can be handled by give_next_Prefix() function and bounded_buffer_queue().

b. 1-2 can be handled by allocating signal through the bounded buffer as much as the consumer thread number.

c. 2-1 can be handled by give_next_Prefix() function and bounded_buffer_dequeue().

d. 2-2 can be handled through the pthread_create() or pthread_cancel(). (When pthread_cancel() is used use pthread_cleanup_push() to handle the terminating thread)

e. 2-3 can be handled by the shared memory.

f. 3-1 can be handled by pthread_create().

g. 3-2 can be handled by gets();

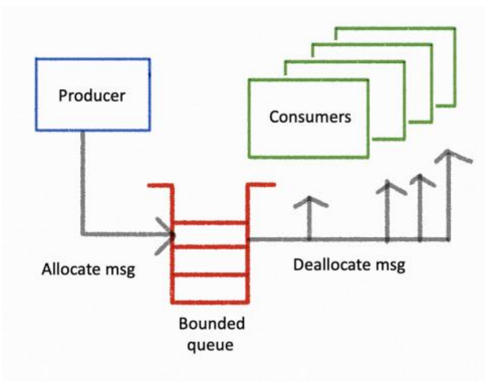
h. 3-3 can be handled by pthread_join();

i. 3-4 can be handled by handler_Ctr_C() or main function.

j. 4-1 can be handled by the pthread_mutex_lock() and pthread_cond_wait().

Table 2 : problem approach

2. Approach



Pic 2 : Overview of the program flow

Pic 1 gives the overview of the program flow. The main function creates Producer, Consumers and Bounded queue. Bounded queue generates the bridge between Producer and Consumers. Producer creates the prefix from the given input and allocate to the Bounded queue. Consumers deallocate bounded queue and calculates the distance from the given prefix.

Here are more specific approaches. To explain more detail of the solution I will follow the Table 2.

a. give_next_prefix() function gives next prefix by looking at the current prefix. (This is same as the hw2) It uses recursive function and generate prefix by decreasing value from right to left. There needs two version of give_next_prefix(). Producer generate prefix from 0 to length-12 and child process length-11 to length -1.

b. If all of the subtasks are given, Consumers should know that there are no more subtask and that they don't have to wait until the buffer is filled. To inform the state we can use the existing communication "bounded buffer". By giving the msg "finished", Consumers can compare the string and find out it is okay to finish the thread. We should repeat the message as much as the number of Consumers so that every Consumer can listen to the message.

d. When thread number are increased we can handle by creating new thread through pthread_create(). When thread numbers are decreased, there needs handler function for reallocating the subtask to the bounded buffer. By giving handler function through the pthread_cleanup_push(), we can provide handler when pthread_cancel() is executed. After main function terminates the thread (that matches with the thread ID), pthread_cancel() automatically pops the handler function and handler reallocates the subtask to the bounded buffer .

e. Because threads shares same memory space, we can use global variables to print out the current information of threads. Also, Cosumers should update the current state through the global variable.

g. Before joining the Consumer threads, main function can listen to the user input by using gets() function.

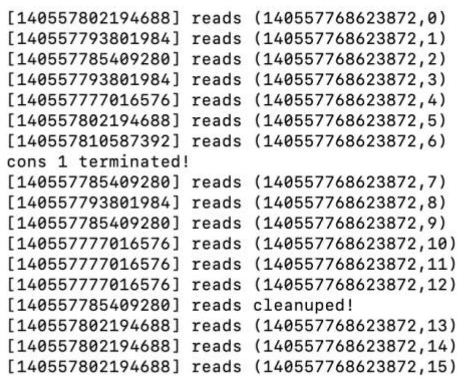
j. Bounded buffer can be implemented by using the conditional waits and mutex lock. When bounded_buffer_queue() or bounded_buffer_dequeue() are execute it acquires the key for accessing buffer by locking the (pthread_mutex_t) lock. Since, the thread should wait until the buffer gets empty or filled I used pthread_cond_wait() to improve the performance of the system.

3. Evaluation

These are some subproblem solution to check the implementation of the approach.

1. checking the behavior of program_clear()

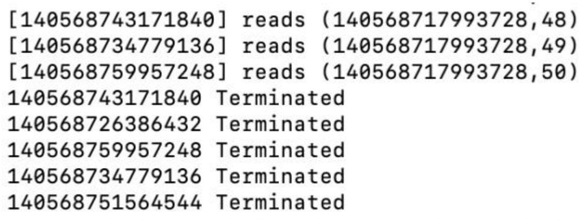
Program is designed to terminate the first consumer thread within a given period. If the program_clear() is well implemented "cons 1 terminated!" should execute through the handler function. And after the other consumer thread reads the reallocated message it should print the message "cleanuped!".



```
[140557802194688] reads (140557768623872,0)
[140557793801984] reads (140557768623872,1)
[140557785409280] reads (140557768623872,2)
[140557793801984] reads (140557768623872,3)
[140557777016576] reads (140557768623872,4)
[140557802194688] reads (140557768623872,5)
[140557810587392] reads (140557768623872,6)
cons 1 terminated!
[140557785409280] reads (140557768623872,7)
[140557793801984] reads (140557768623872,8)
[140557785409280] reads (140557768623872,9)
[140557777016576] reads (140557768623872,10)
[140557777016576] reads (140557768623872,11)
[140557777016576] reads (140557768623872,12)
[140557785409280] reads cleanuped!
[140557802194688] reads (140557768623872,13)
[140557802194688] reads (140557768623872,14)
[140557802194688] reads (140557768623872,15)
```

2. checking if the code is well terminated after receiving all of the producer thread's job.

The example subproblem has 50 subtask and 5 consumer thread. The result shows that all subtasks are finished and 5 threads are terminated.



```
[140568743171840] reads (140568717993728,48)
[140568734779136] reads (140568717993728,49)
[140568759957248] reads (140568717993728,50)
140568743171840 Terminated
140568726386432 Terminated
140568759957248 Terminated
140568734779136 Terminated
140568751564544 Terminated
```

4. Discussion

One thing I struggled hard was getting user inputs. (This made me unable to finish homework on time…) I tried several ways to handle this problem. First, I tried to create new thread that can listen to the user input, however this showed segmentation default. Also, I tried to make user defined handler such as ctr+C. However, I find out user defined handler can only be applied to one character. After thinking more other way I found out that main function can listen to user Input after creating threads and before joing threads. However, I still don't know how to end receiving user inputs when all of the thread execution are ended.

Another problem I struggled is giving Consumer threads that there is no more subtask. My first approach was changing the global variable when Producer finishes its job. However, Producer thread ended up before the Consumer threads end, which makes this approach hard to implement. Another method I though was counting the number of subtask, but I also find out that this method cannot notify the condition to all of the Consumer threads. Finally, I found out giving new message that express the end of process to the Consumer would solve the problem simply.

5. Conclusion

This homework teaches the way of using multiple thread and implementing mutual exclusion. By using the multiple thread, we can efficiently manage the shared memory and also increase the speed of calculation by thread parallelism.