**OSP-2150-PrgAsg1**

Student ID: s3850825

Name: You Chan Lee

Github project url: https://github.com/s3850825/OSP-PrgAsg1-s3850825

The Producer-Consumer Problem

I will explain how the algorithm for the Producer-Consumer problem works. First, I defined NUM\_THREADS as 5 and NUM\_BUCKETS as 10 because they are given. I also defined MAX\_RANDOM\_NUMBER as 99 to generate random numbers between 1 and 99. There are some global variables, bucketIn variable is declared for Producer to see which bucket Producer is in and initialised to 0 as it starts from the first bucket, whereas bucketOut variable is declared for Consumer to see which bucket Consumer is in and initialised to 0 as it starts from the first bucket. An integer NumOfItems is declared and initialised to 0 to check how many items in the bucket. A size 10 integer array bucket[NUM\_BUCKETS] is declared to contain the produced random numbers. Four timeval variables are declared to check the start and end time. A double variable elapsedTime is declared to calculate the running time. Lastly, pthread\_cond\_t and pthread\_mutex\_t is declared.

After declared all the global variables, the algorithm starts from the main method. Firstly, gettimeofday() is used to check the start time. A size 5 integer array threadNum[NUM\_THREADS] is declared and initialised for Producer’s and Consumer’s numbering. An integer variable result is used to check the result of thread creation. Srand((unsinged) time(NULL)) is used to use of the computer’s internal clock to keep changing the seed for random number generator. Both of pthread\_mutex\_init() and pthread\_cond\_init() are used to initialized. And 5 producers and 5 consumers are declared by each array producers[NUM\_THREADS] and consumers[NUM\_THREADS]. After declared producers and consumers, using for loop 5 producer threads and 5 consumer threads are created by using pthread\_create(). Own numbering from threadNum[] array is passed to each producer and consumer thread when they are created.

The function producer is used to state what producers need to do and the function consumer is used to state what consumers need to do. Firstly, in producer function all the producers have infinite while loop to keep producing items, but while loop will be broken when the running time is greater than or equal to 10 seconds. Therefore, producers only work for 10 seconds. A producer produces one item which is a random number between 1 and 99. pthread\_mutex\_lock() is used to lock this thread so that no other thread can execute the same region until this thread is unlocked which is pthread\_mutex\_unlock(). While inside of pthread\_mutex\_lock() a producer checks if buckets are full which means NumOfItems is 10, then this producer thread wait until a consumer thread consumes an item. If buckets are not full, then this producer thread produces an item and put it in the bucket and increments NumOfItems by 1. Once the production is done, then pthread\_mutex\_unlock() is used to unlock this thread so that other threads can execute and also pthread\_cond\_signal() is used to send a signal to another thread who is on waiting. This can avoid deadlock because sending a signal to other threads prevents making all threads to wait. After send the signal, this thread sleeps 10000 microseconds which are same as 0.01 seconds. This can avoid starvation because while this tread is sleeping, other treads can start working, so this ends up dividing the work fairly.

Now, either a producer or a consumer starts working. Secondly, in consumer function all the consumers also have infinite while loop, so consumers also work for 10 seconds. A consumer consumes one item which is a producer produced in a bucket before. Likewise, pthread\_mutex\_lock() is used to lock this consumer thread to prevent those other threads execute the same region. While this consumer thread is locked, this consumer checks if buckets are empty which means NumOfItmes is 0, then this consumer thread wait until a producer thread produces an item. If buckets are not empty, then this consumer thread consumes an item and make this bucket empty, and decrements NumOfItems by 1. Once the consumption is done, then pthread\_mutex\_unlock() is used to unlock this thread so that other threads can start working and pthread\_cond\_signal() is also used to send a signal to another thread who is on waiting. After send the signal, this thread sleeps 10000 microseconds which are same as 0.01 seconds. This can avoid starvation because while this tread is sleeping, other treads can start working, so this ends up dividing the work fairly. After production and consumption run for 10 seconds, pthread\_join() is used to wait all threads to be terminated. Once all the threads are terminated, then checks the total running time and print it. Lastly, pthread\_mutex\_destroy() is used to destroy the mutex object and pthread\_exit(NULL) is used to terminate calling thread.

We can see the producer and consumer problem in real-world industrial. For example, in post office if customers send parcels, then these parcels are kept in post office. While these parcels are waiting to be sent to the recipient, a delivery truck comes to post office and workers load parcels into the truck. Therefore, other workers need to unload these parcels from the full delivery truck to deliver them to the recipient and workers in post office need to wait until next empty delivery truck comes in to load parcels. Here we can clearly items are parcels, producers are workers who need to load parcels into an empty delivery truck, consumers are workers who need to unload parcels from a full delivery truck to deliver them to the recipient and buckets are delivery trucks.

The Dining Philosophers’ Problem

Cooperating processes that need to share limited resources

• Set of processes that need to lock multiple resources – Disk and tape (backup),

• Travel reservation: hotel, airline, car rental databases