Homework Wan Huzaifah bin Wan Azhar

Answer:



* First, the program will create threads for producer and consumer.
* Then, producer or consumer threads will run.
  + The order does not matter here if there is no argument specified, it will run at random.
  + The producer will get the lock, wait until buffer is empty before filling it with a value. Then it releases lock.
  + The consumer thread is the same except it waits until buffer is filled.
  + Both threads wait by using while-loop and condition variable.
* There are two condition variables in this program.
  + One CV is to signal or wait for an empty buffer.
  + Another CV is to signal or wait for full buffer.
  + These two CV can make producer thread signal to consumer thread and vice versa.
* The main program will wait until all producer threads is finished.
* Finally, it will wait until all consumer threads is finished before exiting gracefully.



* The behaviour pattern of producer produces and then consumer consume does not change even if buffer size increase.
  + This is because when the producer finish producing, it will signal the consumer thread to run next, which will signal producer after it finish consuming.
  + As such the pattern of producer -> consumer -> producer -> consumer will keep repeating even if buffer is increasing.
* I predict the final num\_full to be 0 even if buffer is set to 10, produced items to 100 and 7th line of consumer code to sleep to 1.
  + This is because even if consumer code sleep to 1, it does not change that the next thread running will be producer thread.
  + As established earlier, changing buffer size and produced item does not change behaviour pattern of the program.
  + The only things changing is that the program will take some time to finish as there is 1s of sleep in every consumer thread.
  + Turns out I’m correct after running the code.

1. Skipped because I don’t have Mac.

./main-two-cvs-while   
-p 1 #1 producer  
-c 3 #3 consumer  
-m 1 #buffer size of 1  
-C #Consumer sleep at every 4th line of code  
0,0,0,1,0,0,0:   
0,0,0,1,0,0,0:   
0,0,0,1,0,0,0   
-l 10 #producing 10 item  
-v -t

Simulation:

P1 produce 1 <1s

C1/C2/C3 consume 1s

P1 produce 2 <1s

C1/C2/C3 consume 1s

P1 produce 3 <1s

C1/C2/C3 consume 1s

P1 produce 4 <1s

C1/C2/C3 consume 1s

P1 produce 5 <1s

C1/C2/C3 consume 1s

P1 produce 6 <1s

C1/C2/C3 consume 1s

P1 produce 7 <1s

C1/C2/C3 consume 1s

P1 produce 8 <1s

C1/C2/C3 consume 1s

P1 produce 9 <1s

C1/C2/C3 consume 1s

P1 produce 10 <1s

C1/C2/C3 consume 1s

Total time, t > 10s

* The simulation describes the general flow of the program.
* Although running the program different time produce different output, it generally follows the same pattern as the simulation.
  + This is because, in the beginning, it doesn’t matter which threads start first, but the Producer must produce something first before any consumer can consume anything.
  + When producer is finished producing, it will wake up one consumer thread, which will then wake up another producer thread, which will wake up consumer thread and so on.
* At the end, time taken is higher than 10 seconds.

1. Changing shared buffer to 3 will not change much in time taken.

* This is because there is only one producer.
* One producer can only produce one items for one consumer at a time.



* The total time taken to finish the program should be less than previous time, which is t < 10s
* This is because after consumer signals to producer that the buffer is empty, it releases the lock and then sleeps for 1 seconds.
* Therefore, other thread can run without waiting for the consumer thread to finish.
* After all consumer thread is finished, the consumer threads restarts waiting.

1. The time taken does not change much from previous time.

* This is because although more can be put into the buffer, there is still only one producer.
* Therefore, consumer cannot consume as fast if the producer is slow.

1. There is no way to cause problem with this code because it is correct with one producer and consumer.

* Argument -P 0,0,0,0,0,0,10 -C 0:0 cause consumer threads to wake up another consumer thread.
* But as, there is no thing to consume, all threads are sleeping.
* The program will not finish.



* There is no problem in this program with one producer and one consumer as the consumer will always consume a value.
* There will be a problem with more than one consumer.
  + If another consumer consumes a value while another consumer is sleeping, the consumer that is sleeping will continue and trigger exception.
  + This is because using ‘if-statement’, the thread will only check once; that is before it is sleeping.
  + If the threads are awakened, it will continue to get the value from the buffer.
  + But the buffer is empty, thus it will lead to error.
  + As such, the thread needs to check the buffer before and after the threads is sleeping.
* Using ‘while’ will cause the awaken thread to check the buffer first.
  + If buffer is empty, it continues to sleep.
  + Otherwise, consume the value.



* Running using argument -p -c -m 10 -l 10 -v -C 0,0,0,0,0,1,0:0 can cause severe imbalance of consumption between consumer thread.
* This is because when the first consumer thread wants to consume, it will sleep for one second before other threads get the lock.
* And thus, in the end, it can only consume one value.