Tutorial 5 Answer Sheet

CS3103 Operating Systems

Student Name: LIU Hengche
Student No. : 5 7 8 5 4 3 2 9

Day: ☐Tuesday ☑Wednesday ☐Friday
Time: ☐ 10:00 - 10:50 ☐ 11:00 - 11:50 ☐ 16:00 - 16:50 ☑ 18:00 - 18:50

Condition Variables

Submission:

- Deadline: Sunday, March 17, 2024, 23:59 pm HKT.
- Answers are allowed in text only. Any form of image/snapshot is not allowed.
- Submit this answer sheet via Canvas->Files->Tutorials->Tutorial 5.

Questions

Question 1: Our first question focuses on `main-two-cvs-while.c` (the working solution). Run with one producer and one consumer, and have the producer produce a few values. Start with a buffer (size 1), and then increase it. How does the behavior of the code change with larger buffers? (or does it?) What would you predict `num_full` to be with different buffer sizes (e.g., `-m 10`) and different numbers of produced items (e.g., `-l 100`), when you change the consumer sleep string from default (no sleep) to `-C 0,0,0,0,0,0,1`?

Answer:

The first command is ./main-two-cvs-while -l 3 -m 1 -p 1 -c 1 -v. To increase the buffer size, we increase the number after '-m'. As the buffer size increases, the producer could put more values to the buffer before it returns the mutex lock and the consumer could have more values to consume from the buffer before returning the lock.

Num_full is to record the the total number of values stored in the buffer. When we use "-C 0,0,0,0,0,0,1", this means that the consumer sleeps at line c6 (Mutex_unlock(&m);   c6;) which means that after the consumer grabs a value and returns the mutex lock, it immediately goes to sleep for one unit of time. Then the producer would produce. So in this case, producer produces a lot faster than the consumer consumes, which suggests that num_full would be almost the same as the buffer size for the most of the time.

Question 2: Let's look at some timings. How long do you think the following execution, with one producer, three consumers, a single-entry shared buffer, and each consumer pausing at point c3 for a second, will take?

```
./main-two-cvs-while -p 1 -c 3 -m 1 -C 0,0,0,1,0,0,0:0,0,0,1,0,0,0:0,0,0,1,0,0,0
-l 10 -v -t
```

Answer:

I think it's around 10 seconds. Although that the consumer is more than the producer, but since the consumers sleep at c3, which means that we do context switch if num_full = 0 and wait for the producer to produce. Then this suggests that the actual speed of producer producing and consumer consuming is pretty much the same. Since there are 10 loops, and each time a consumer consumes it sleeps for a second, so it's around 10 seconds.

The actual time it takes after I run the command is 12.02s. The output is as follows:

```
NF          P0 C0 C1 C2

  0 [*--- ] p0

  0 [*--- ]    c0

  0 [*--- ] p1

  0 [*  0 ]       c0

  1 [*  0 ] p4

  1 [*  0 ]          c0

  1 [*  0 ] p5

  1 [*  0 ] p6

  1 [*  0 ]    c1

  1 [*  0 ] p0

  0 [*--- ]    c4

  0 [*--- ]    c5

  0 [*--- ]    c6

  0 [*--- ]       c1

  0 [*--- ]    c0

  0 [*--- ]       c2

  0 [*--- ]          c1

  0 [*--- ]          c2
```

```
0 [*--- ] p1
1 [*  1 ] p4
1 [*  1 ] p5
1 [*  1 ] p6
1 [*  1 ]    c1
1 [*  1 ] p0
0 [*--- ]    c4
0 [*--- ]    c5
0 [*--- ]    c6
0 [*--- ]       c3
0 [*--- ]    c0
0 [*--- ]       c2
0 [*--- ] p1
1 [*  2 ] p4
1 [*  2 ] p5
1 [*  2 ] p6
1 [*  2 ]    c1
1 [*  2 ] p0
0 [*--- ]    c4
0 [*--- ]    c5
0 [*--- ]    c6
0 [*--- ]          c3
0 [*--- ]    c0
0 [*--- ]          c2
0 [*--- ] p1
1 [*  3 ] p4
1 [*  3 ] p5
1 [*  3 ] p6
```

```
1 [*  3 ]    c1
1 [*  3 ] p0
0 [*--- ]    c4
0 [*--- ]    c5
0 [*--- ]    c6
0 [*--- ]       c3
0 [*--- ]    c0
0 [*--- ]       c2
0 [*--- ] p1
1 [*  4 ] p4
1 [*  4 ] p5
1 [*  4 ] p6
1 [*  4 ]    c1
1 [*  4 ] p0
0 [*--- ]    c4
0 [*--- ]    c5
0 [*--- ]    c6
0 [*--- ]          c3
0 [*--- ]    c0
0 [*--- ]          c2
0 [*--- ] p1
1 [*  5 ] p4
1 [*  5 ] p5
1 [*  5 ] p6
1 [*  5 ]    c1
1 [*  5 ] p0
0 [*--- ]    c4
0 [*--- ]    c5
```

```
0 [*--- ]    c6
0 [*--- ]      c3
0 [*--- ]    c0
0 [*--- ]      c2
0 [*--- ] p1
1 [*  6 ] p4
1 [*  6 ] p5
1 [*  6 ] p6
1 [*  6 ]    c1
1 [*  6 ] p0
0 [*--- ]    c4
0 [*--- ]    c5
0 [*--- ]    c6
0 [*--- ]         c3
0 [*--- ]    c0
0 [*--- ]         c2
0 [*--- ] p1
1 [*  7 ] p4
1 [*  7 ] p5
1 [*  7 ] p6
1 [*  7 ]    c1
1 [*  7 ] p0
0 [*--- ]    c4
0 [*--- ]    c5
0 [*--- ]    c6
0 [*--- ]      c3
0 [*--- ]    c0
0 [*--- ]      c2
```

```
0 [*--- ] p1
1 [*  8 ] p4
1 [*  8 ] p5
1 [*  8 ] p6
1 [*  8 ]   c1
1 [*  8 ] p0
0 [*--- ]   c4
0 [*--- ]   c5
0 [*--- ]   c6
0 [*--- ]         c3
0 [*--- ]   c0
0 [*--- ]         c2
0 [*--- ] p1
1 [*  9 ] p4
1 [*  9 ] p5
1 [*  9 ] p6
1 [*  9 ]   c1
0 [*--- ]   c4
0 [*--- ]   c5
0 [*--- ]   c6
0 [*--- ]      c3
0 [*--- ]   c0
0 [*--- ]      c2
1 [*EOS ] [main: added end-of-stream marker]
1 [*EOS ]         c3
0 [*--- ]         c4
0 [*--- ]         c5
0 [*--- ]         c6
```

```
  0 [*--- ]    c1
  0 [*--- ]    c2
  1 [*EOS ] [main: added end-of-stream marker]
  1 [*EOS ]       c3
  0 [*--- ]       c4
  0 [*--- ]       c5
  0 [*--- ]       c6
  1 [*EOS ] [main: added end-of-stream marker]
  1 [*EOS ]    c3
  0 [*--- ]    c4
  0 [*--- ]    c5
  0 [*--- ]    c6
Consumer consumption:
  C0 -> 10
  C1 -> 0
  C2 -> 0
Total time: 12.02 seconds
```

Question 3: Now change the size of the shared buffer to 3 (`-m 3`). Will this make any difference in the total time?

Answer:

Since the speed of the producers and the consumers is about the same, then this suggests that the size of buffer does not really matter since consumers consume as fast as producer produces. So the total time would be about the same.

After execution, the total time is 11.02s.

Question 4: Now change the location of the sleep to c6 (this models a consumer taking something off the queue and then doing something with it), again using a single-entry buffer. What time do you predict in this case?

```
./main-two-cvs-while -p 1 -c 3 -m 1 -C 0,0,0,0,0,0,1:0,0,0,0,0,0,1:0,0,0,0,0,0,1
-l 10 -v -t
```

Answer:

The total time is now 5 seconds. The output is as follows:

```
NF          P0 C0 C1 C2

  0 [*--- ] p0

  0 [*--- ]    c0

  0 [*--- ] p1

  0 [*--- ]       c0

  1 [*  0 ] p4

  1 [*  0 ]          c0

  1 [*  0 ] p5

  1 [*  0 ] p6

  1 [*  0 ]    c1

  1 [*  0 ] p0

  0 [*--- ]    c4

  0 [*--- ]    c5

  0 [*--- ]    c6
```

```
0 [*--- ]      c1
0 [*--- ]      c2
0 [*--- ]         c1
0 [*--- ]         c2
0 [*--- ] p1
1 [*  1 ] p4
1 [*  1 ] p5
1 [*  1 ] p6
1 [*  1 ]      c3
1 [*  1 ] p0
0 [*--- ]      c4
0 [*--- ]      c5
0 [*--- ]      c6
0 [*--- ] p1
1 [*  2 ] p4
1 [*  2 ] p5
1 [*  2 ] p6
1 [*  2 ]         c3
1 [*  2 ] p0
0 [*--- ]         c4
0 [*--- ]         c5
0 [*--- ]         c6
0 [*--- ] p1
1 [*  3 ] p4
1 [*  3 ] p5
1 [*  3 ] p6
1 [*  3 ] p0
1 [*  3 ] p1
```

```
1 [*  3 ] p2
1 [*  3 ]    c0
1 [*  3 ]    c1
0 [*--- ]    c4
0 [*--- ]    c5
0 [*--- ]    c6
0 [*--- ] p3
1 [*  4 ] p4
1 [*  4 ] p5
1 [*  4 ] p6
1 [*  4 ] p0
1 [*  4 ] p1
1 [*  4 ] p2
1 [*  4 ]       c0
1 [*  4 ]       c1
0 [*--- ]       c4
0 [*--- ]       c5
0 [*--- ]       c6
0 [*--- ] p3
1 [*  5 ] p4
1 [*  5 ]          c0
1 [*  5 ] p5
1 [*  5 ] p6
1 [*  5 ]          c1
1 [*  5 ] p0
0 [*--- ]          c4
0 [*--- ]          c5
0 [*--- ]          c6
```

```
0 [*--- ] p1
1 [*  6 ] p4
1 [*  6 ] p5
1 [*  6 ] p6
1 [*  6 ] p0
1 [*  6 ] p1
1 [*  6 ] p2
1 [*  6 ]    c0
1 [*  6 ]    c1
0 [*--- ]    c4
0 [*--- ]    c5
0 [*--- ]    c6
0 [*--- ] p3
1 [*  7 ] p4
1 [*  7 ] p5
1 [*  7 ] p6
1 [*  7 ] p0
1 [*  7 ] p1
1 [*  7 ] p2
1 [*  7 ]       c0
1 [*  7 ]       c1
0 [*--- ]       c4
0 [*--- ]       c5
0 [*--- ]       c6
0 [*--- ] p3
1 [*  8 ] p4
1 [*  8 ] p5
1 [*  8 ]          c0
```

```
1 [*  8 ] p6

1 [*  8 ]         c1

1 [*  8 ] p0

0 [*--- ]         c4

0 [*--- ]         c5

0 [*--- ]         c6

0 [*--- ] p1

1 [*  9 ] p4

1 [*  9 ] p5

1 [*  9 ] p6

1 [*  9 ]    c0

1 [*  9 ]    c1

0 [*--- ]    c4

0 [*--- ]    c5

0 [*--- ]    c6

1 [*EOS ] [main: added end-of-stream marker]

1 [*EOS ]      c0

1 [*EOS ]      c1

0 [*--- ]      c4

0 [*--- ]      c5

0 [*--- ]      c6

1 [*EOS ] [main: added end-of-stream marker]

1 [*EOS ]         c0

1 [*EOS ]         c1

0 [*--- ]         c4

0 [*--- ]         c5

0 [*--- ]         c6

1 [*EOS ] [main: added end-of-stream marker]
```

```
  1 [*EOS ]    c0

  1 [*EOS ]    c1

  0 [*--- ]    c4

  0 [*--- ]    c5

  0 [*--- ]    c6


Consumer consumption:

  C0 -> 4

  C1 -> 3

  C2 -> 3


Total time: 5.00 seconds
```

Question 5: Finally, change the buffer size to 3 again (`-m 3`). What time do you predict now?

Answer:

The time is still 5 seconds. When the buffer is enlarged, the producer could produce faster, but since the main overhead is caused by the consumer, then the time is not heavily affected if we only change the buffer size.

After execution, the total time is 5.00 seconds.

Question 6: Now let's look at `main-one-cv-while.c`. Can you configure a sleep string, assuming a single producer, one consumer, and a buffer of size 1, to cause a problem with this code?

Answer:

I don't think that there's any problem if we have only one conditional variable for the single producer consumer problem. This is because we can consider consumer also a kind of producer that produces empty slots while the producer consumes the empty slot. In this case, one conditional variable is sufficient to indicate for both consumer and producer whether they enter the critical section or not.

Question 7: Now change the number of consumers to two. Can you construct sleep strings for the producer and the consumers so as to cause a problem in the code?

Answer:

```
./main-one-cv-while -l 1 -p 1 -c 2 -m 1 -P 0,0,0,0,0,0,1 -v
```

Since there's only one conditional variable, then this means that there's a hazard of consumers waking consumers and producers waking producers. So we can create a one-producer and two-consumer scenario where two consumers might wake up each other.

Question 8: Now examine `main-two-cvs-if.c`. Can you cause a problem to happen in this code? Again, consider the case where there is only one consumer, and then the case where there is more than one.

Answer:

If there's only one producer and one consumer, then rechecking before entering critical section would not be required.

However, if not, the hazard could be entering the critical section while the condition is no longer ideal(Mesa semantic), like consumer consuming from an empty buffer and producer produces to a full buffer.

We could use the command: ./main-two-cvs-if -p 1 -c 2 -l 2 -m 1 -P 1 -C 1,0,0,0,0,0,0:0,0,0,1,0,0,0. This causes the scenario where consumers consume empty slots.