[REPORT]

■과 목 명: 운영체제 (SWE3004-42)

■담 당 교 수: 신동군 교수님

■제 출 일: 2022년 4월 29일

■학　　과: 수학과

■학　　번: 2016314786

■성　　명: 김호진

Chap 30. Condition Variables

수학과 김호진 (2016314786)

1. Our first question focuses on main-two-cvs-while.c **(the working solution). First, study the code. Do you think you have an understanding of what should happen when you run the program?**

main-two-cvs-while.c을 살펴보면 condition variable인 empty와 fill을 사용하여 while loop를 동작하기 때문에 working solution이라고 할 수 있다. 그러므로 각 producer는 항목 수(loop option)를 구성할 수 있고, consumer는 추가된 항목이 모두 처리될 때까지 실행된다.

2. 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**?**

버퍼의 크기가 1인 경우, 코드는 다음과 같이 동작한다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch30$ ./main-two-cvs-while -p 1 -c 1 -m 1 -v
 NF        P0 C0
  0 [*--- ] p0
  0 [*--- ]    c0
  0 [*--- ] p1
  1 [*  0 ] p4
  1 [*  0 ] p5
  1 [*  0 ] p6
  1 [*  0 ]    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 -> 1
```

버퍼의 크기가 커지더라도, 코드의 동작에는 변화가 없다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch30$ ./main-two-cvs-while -p 1 -c 1 -m 10 -v
 NF                                                  P0 C0
  0 [*---  ---  ---  ---  ---  ---  ---  ---  ---  --- ] p0
  0 [*---  ---  ---  ---  ---  ---  ---  ---  ---  --- ]    c0
  0 [*---  ---  ---  ---  ---  ---  ---  ---  ---  --- ] p1
  1 [u  0 f---  ---  ---  ---  ---  ---  ---  ---  --- ] p4
  1 [u  0 f---  ---  ---  ---  ---  ---  ---  ---  --- ] p5
  1 [u  0 f---  ---  ---  ---  ---  ---  ---  ---  --- ] p6
  1 [u  0 f---  ---  ---  ---  ---  ---  ---  ---  --- ]    c1
  0 [ --- *---  ---  ---  ---  ---  ---  ---  ---  --- ]    c4
  0 [ --- *---  ---  ---  ---  ---  ---  ---  ---  --- ]    c5
  0 [ --- *---  ---  ---  ---  ---  ---  ---  ---  --- ]    c6
  1 [ --- uEOS f---  ---  ---  ---  ---  ---  ---  --- ] [main: added end-of-stream marker]
  1 [ --- uEOS f---  ---  ---  ---  ---  ---  ---  --- ]    c0
  1 [ --- uEOS f---  ---  ---  ---  ---  ---  ---  --- ]    c1
  0 [ ---  --- *---  ---  ---  ---  ---  ---  ---  --- ]    c4
  0 [ ---  --- *---  ---  ---  ---  ---  ---  ---  --- ]    c5
  0 [ ---  --- *---  ---  ---  ---  ---  ---  ---  --- ]    c6

Consumer consumption:
  C0 -> 1
```

Consumer sleep string을 기본값에서 -C 0,0,0,0,0,0,1로 변경하면 코드는 다음과 같이 동작한다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch30$ ./main-two-cvs-while -p 1 -c 1 -m 1 -C 0,0,0,0,0,0,1 -v
 NF        P0 C0
  0 [*--- ] p0
  0 [*--- ]    c0
  0 [*--- ] p1
  1 [*  0 ] p4
  1 [*  0 ] p5
  1 [*  0 ] p6
  1 [*  0 ]    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 -> 1
```

해당 consumer sleep string에 대해 버퍼 크기(-m 10)와 생산 품목 수(-l 100)가 다른 경우, 다음의 실행 결과를 통해 num_full은 0부터 최대값인 10까지 증가한 뒤 9와 10 사이에서 계속해서 변동하는 것을 확인할 수 있다.

```
borusser@DESKTOP-L834KLC:~/ostep_ch30$ ./main-two-cvs-while -p 1 -c 1 -m 10 -l 100 -C 0,0,0,0,0,0,1 -v
 NF                                                                  P0 C0
  0 [*---  ---  ---  ---  ---  ---  ---  ---  ---  --- ] p0
  0 [*---  ---  ---  ---  ---  ---  ---  ---  ---  --- ]    c0
  0 [*---  ---  ---  ---  ---  ---  ---  ---  ---  --- ] p1
  1 [u  0 f---  ---  ---  ---  ---  ---  ---  ---  --- ] p4
  1 [u  0 f---  ---  ---  ---  ---  ---  ---  ---  --- ] p5
  1 [u  0 f---  ---  ---  ---  ---  ---  ---  ---  --- ] p6
  1 [u  0 f---  ---  ---  ---  ---  ---  ---  ---  --- ]    c1
  1 [u  0 f---  ---  ---  ---  ---  ---  ---  ---  --- ] p0
  0 [ --- *---  ---  ---  ---  ---  ---  ---  ---  --- ]    c4
  0 [ --- *---  ---  ---  ---  ---  ---  ---  ---  --- ]    c5
  0 [ --- *---  ---  ---  ---  ---  ---  ---  ---  --- ]    c6
  0 [ --- *---  ---  ---  ---  ---  ---  ---  ---  --- ] p1
  1 [ --- u  1 f---  ---  ---  ---  ---  ---  ---  --- ] p4
  1 [ --- u  1 f---  ---  ---  ---  ---  ---  ---  --- ] p5
  1 [ --- u  1 f---  ---  ---  ---  ---  ---  ---  --- ] p6
  1 [ --- u  1 f---  ---  ---  ---  ---  ---  ---  --- ] p0
  1 [ --- u  1 f---  ---  ---  ---  ---  ---  ---  --- ] p1
  2 [ --- u  1    2 f---  ---  ---  ---  ---  ---  --- ] p4
  2 [ --- u  1    2 f---  ---  ---  ---  ---  ---  --- ] p5
  2 [ --- u  1    2 f---  ---  ---  ---  ---  ---  --- ] p6
  2 [ --- u  1    2 f---  ---  ---  ---  ---  ---  --- ] p0
  2 [ --- u  1    2 f---  ---  ---  ---  ---  ---  --- ] p1
  3 [ --- u  1    2    3 f---  ---  ---  ---  ---  --- ] p4
  3 [ --- u  1    2    3 f---  ---  ---  ---  ---  --- ] p5
  3 [ --- u  1    2    3 f---  ---  ---  ---  ---  --- ] p6
  3 [ --- u  1    2    3 f---  ---  ---  ---  ---  --- ] p0
  3 [ --- u  1    2    3 f---  ---  ---  ---  ---  --- ] p1
  4 [ --- u  1    2    3    4 f---  ---  ---  ---  --- ] p4
  4 [ --- u  1    2    3    4 f---  ---  ---  ---  --- ] p5
  4 [ --- u  1    2    3    4 f---  ---  ---  ---  --- ] p6
  4 [ --- u  1    2    3    4 f---  ---  ---  ---  --- ] p0
  4 [ --- u  1    2    3    4 f---  ---  ---  ---  --- ] p1
  5 [ --- u  1    2    3    4    5 f---  ---  ---  --- ] p4
  5 [ --- u  1    2    3    4    5 f---  ---  ---  --- ] p5
  5 [ --- u  1    2    3    4    5 f---  ---  ---  --- ] p6
  5 [ --- u  1    2    3    4    5 f---  ---  ---  --- ] p0
  5 [ --- u  1    2    3    4    5 f---  ---  ---  --- ] p1
  6 [ --- u  1    2    3    4    5    6 f---  ---  --- ] p4
  6 [ --- u  1    2    3    4    5    6 f---  ---  --- ] p5
  6 [ --- u  1    2    3    4    5    6 f---  ---  --- ] p6
  6 [ --- u  1    2    3    4    5    6 f---  ---  --- ] p0
  6 [ --- u  1    2    3    4    5    6 f---  ---  --- ] p1
  7 [ --- u  1    2    3    4    5    6    7 f---  --- ] p4
  7 [ --- u  1    2    3    4    5    6    7 f---  --- ] p5
  7 [ --- u  1    2    3    4    5    6    7 f---  --- ] p6
  7 [ --- u  1    2    3    4    5    6    7 f---  --- ] p0
  7 [ --- u  1    2    3    4    5    6    7 f---  --- ] p1
  8 [ --- u  1    2    3    4    5    6    7    8 f--- ] p4
  8 [ --- u  1    2    3    4    5    6    7    8 f--- ] p5
  8 [ --- u  1    2    3    4    5    6    7    8 f--- ] p6
  8 [ --- u  1    2    3    4    5    6    7    8 f--- ] p0
  8 [ --- u  1    2    3    4    5    6    7    8 f--- ] p1
  9 [f--- u  1    2    3    4    5    6    7    8    9 ] p4
  9 [f--- u  1    2    3    4    5    6    7    8    9 ] p5
  9 [f--- u  1    2    3    4    5    6    7    8    9 ] p6
  9 [f--- u  1    2    3    4    5    6    7    8    9 ] p0
  9 [f--- u  1    2    3    4    5    6    7    8    9 ] p1
 10 [  10 *  1    2    3    4    5    6    7    8    9 ] p4
 10 [  10 *  1    2    3    4    5    6    7    8    9 ] p5
 10 [  10 *  1    2    3    4    5    6    7    8    9 ] p6
 10 [  10 *  1    2    3    4    5    6    7    8    9 ] p0
 10 [  10 *  1    2    3    4    5    6    7    8    9 ] p1
 10 [  10 *  1    2    3    4    5    6    7    8    9 ] p2
 10 [  10 *  1    2    3    4    5    6    7    8    9 ]    c0
 10 [  10 *  1    2    3    4    5    6    7    8    9 ]    c1
  9 [  10 f--- u  2    3    4    5    6    7    8    9 ]    c4
  9 [  10 f--- u  2    3    4    5    6    7    8    9 ]    c5
  9 [  10 f--- u  2    3    4    5    6    7    8    9 ]    c6
  9 [  10 f--- u  2    3    4    5    6    7    8    9 ] p3
 10 [  10   11 *  2    3    4    5    6    7    8    9 ] p4
 10 [  10   11 *  2    3    4    5    6    7    8    9 ] p5
 10 [  10   11 *  2    3    4    5    6    7    8    9 ] p6
 10 [  10   11 *  2    3    4    5    6    7    8    9 ] p0
 10 [  10   11 *  2    3    4    5    6    7    8    9 ] p1
 10 [  10   11 *  2    3    4    5    6    7    8    9 ] p2
```

3. If possible, run the code on different systems (e.g., a Mac and Linux). Do you see different behavior across these systems?

Linux에서 코드를 실행하면, 프로세서가 하나만 있는 것처럼 동작하기 때문에 대부분의 경우에서 single consumer가 모든 value를 갖는다. 하지만, Mac에서는 producer와 consumer thread가 interleave하므로 이러한 현상이 비교적으로 드물게 발생한다.

4. 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

First consumer는 wait 없이 value를 얻으므로 sleep 하지 않는다. Second consumer는 wait을 하지만, producer가 producing을 한 이후 lock을 풀면 first consumer가 다시 lock을 가지게 되므로 second consumer는 wakes up 되었을 때 value를 갖지 못하고 다시 sleep 하게 된다. 이런 식으로 second consumer와 third consumer는 총 9초 동안 sleep 하게 된다. 이후 stream이 종료되면, 세 consumer는 각각 1초씩 sleep 하게 된다. 따라서 프로그램은 다음과 같이 12초 정도가 소요된다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch30$ ./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
```

```
Consumer consumption:
  C0 -> 10
  C1 -> 0
  C2 -> 0

Total time: 12.03 seconds
```

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

버퍼의 크기를 3(-m 3)으로 변경하면서 충분한 버퍼 슬롯을 가지게 되고, 그로 인해 Threads 중 하나의 EOS를 기다릴 필요가 없게 되면서 total time은 11초 정도가 소요된다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch30$ ./main-two-cvs-while -p 1 -c 3 -m 3 -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
```

```
Consumer consumption:
  C0 -> 1
  C1 -> 0
  C2 -> 9

Total time: 11.03 seconds
```

6. 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

해당 상황에서 consumer는 Lock을 가지고 있는 동안 sleep 하지 않으면서, 3가지 항목을 동시에 처리할 수 있게 된다. 따라서 10개의 항목과 3개의 EOS를 처리하는 데 (10+3)/3+contention_time =5초 정도가 소요된다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch30$ ./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
Consumer consumption:
  C0 -> 3
  C1 -> 4
  C2 -> 3

Total time: 5.01 seconds
```

7. Finally, change the buffer size to 3 again (-m 3**). What time do you predict now?**

동시에 처리할 수 있는 threads 수를 증가시키기는 것은 Consumer의 수이기 때문에 buffer의 크기는 시간에 영향을 거의 미치지 않는다. 따라서 이전과 마찬가지로 5초 정도가 소요된다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch30$ ./main-two-cvs-while -p 1 -c 3 -m 3 -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
Consumer consumption:
  C0 -> 3
  C1 -> 4
  C2 -> 3

Total time: 5.01 seconds
```

8. 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?**

아니다. Single producer와 one consumer만 존재하므로 하나의 condition variable로도 충분하다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch30$ ./main-one-cv-while -p 1 -c 1 -m 1 -C 1,1,1,1,1,1 -v
 NF        P0 C0
  0 [*--- ] p0
  0 [*--- ]    c0
  0 [*--- ] p1
  1 [*  0 ] p4
  1 [*  0 ] p5
  1 [*  0 ] p6
  1 [*  0 ]    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 -> 1
```

9. 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?

다음의 sleep string에서는 모든 threads가 sleeping 되면서 코드에 문제가 생긴다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch30$ ./main-one-cv-while -p 1  -c 2 -m 1 -v -P 0,0,0,0,0,0,1
 NF        P0 C0 C1
  0 [*--- ] p0
  0 [*--- ]       c0
  0 [*--- ]    c0
  0 [*--- ] p1
  1 [*  0 ] p4
  1 [*  0 ] p5
  1 [*  0 ] p6
  1 [*  0 ]       c1
  0 [*--- ]       c4
  0 [*--- ]       c5
  0 [*--- ]       c6
  0 [*--- ]    c1
  0 [*--- ]       c0
  0 [*--- ]    c2
  0 [*--- ]       c1
  0 [*--- ]       c2
  1 [*EOS ] [main: added end-of-stream marker]
  1 [*EOS ]    c3
  0 [*--- ]    c4
  0 [*--- ]    c5
  0 [*--- ]    c6
  0 [*--- ]       c3
  0 [*--- ]       c2
```

10. 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.**

One consumer의 경우, main-two-cvs-if.c에 문제를 발생시키기 힘들다.

그러나 consumer가 둘 이상이면, 다음처럼 empty buffer error로 인한 문제가 발생할 수 있다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch30$ ./main-two-cvs-if -p 1 -c 2 -l 10 -m 1 -C ,0:0,2,0,0
error: tried to get an empty buffer
```

11. Finally, examine main-two-cvs-while-extra-unlock.c**. What problem arises when you release the lock before doing a put or a get? Can you reliably cause such a problem to happen, given the sleep strings? What bad thing can happen?**

put이나 get을 하기 전 lock을 해제하면 put과 get 내의 critical section에 대한 mutual exclusion 이 제거된다. 이로 인해 두 producers가 동시에 do_fill을 하거나 두 consumers가 동시에 do_get 을 하려고 할 때, shared memory에 대한 race condition이 생기게 되어 문제가 발생한다.

다음의 sleep string은 First consumer가 하나의 Value만을 consume하는 단점을 가진다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch30$ ./main-two-cvs-while-extra-unlock -p 1 -c 2 -m 10 -l 10 -v -C 0,0,0,0,1,0:0
```

```
Consumer consumption:
  C0 -> 1
  C1 -> 9
```