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?

```
}

void *producer(void *arg) {
    int id = (long int) arg;
    // make sure each producer produces unique values
    int base = id * loops;
    int i;
    for (i = 0; i < loops; i++) {   p0;
        Mutex_lock(&m);             p1;
        while (num_full == max) {   p2;
            Cond_wait(&empty, &m);  p3;
        }
        do_fill(base + i);          p4;
        Cond_signal(&fill);         p5;
        Mutex_unlock(&m);           p6;
    }
    return NULL;
}

void *consumer(void *arg) {
    int id = (long int) arg;
    int tmp = 0;
    int consumed_count = 0;
    while (tmp != END_OF_STREAM) { c0;
        Mutex_lock(&m);            c1;
        while (num_full == 0) {    c2;
            Cond_wait(&fill, &m);  c3;
        }
        tmp = do_get();            c4;
        Cond_signal(&empty);       c5;
        Mutex_unlock(&m);          c6;
        consumed_count++;
    }
```

main-two-cvs-while.c의 코드는 다음과 같다.

만약 producer는 buffer를 채우고 consumer는 buffer를 비운다. 만약 buffer가 꽉 차게 된다면 producer는 더 이상 buffer를 채울 수 없다. 만약 buffer가 텅 비워진다면 consumer는 더 이상 buffer를 비울 수 없다. Buffer에 producer와 consumer가 동시에 접근하는 것을 막기 위해 lock을 사용한다. 다른 스레드가 lock을 사용하고 있는 동안 lock을 얻기 위해 계속 while문을 도는 것은 비효율적이므로 cv를 사용한다. 여기서 cv는 &empty와 &fill이다. 만약 buffer가 꽉 차 있다면 wait(&empty)에 의해 producer는 잠들게 된다. Consumer가 buffer를 채운 후 signal(&empty)를 통해 producer를 깨워준다. 만약 buffer가 비워져 있다면 wait(&fill)에 의해 consumer는 잠들게 된다. producer가 buffer를 채운 후 signal(&fill)를 통해 consumer를 깨워준다.

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?

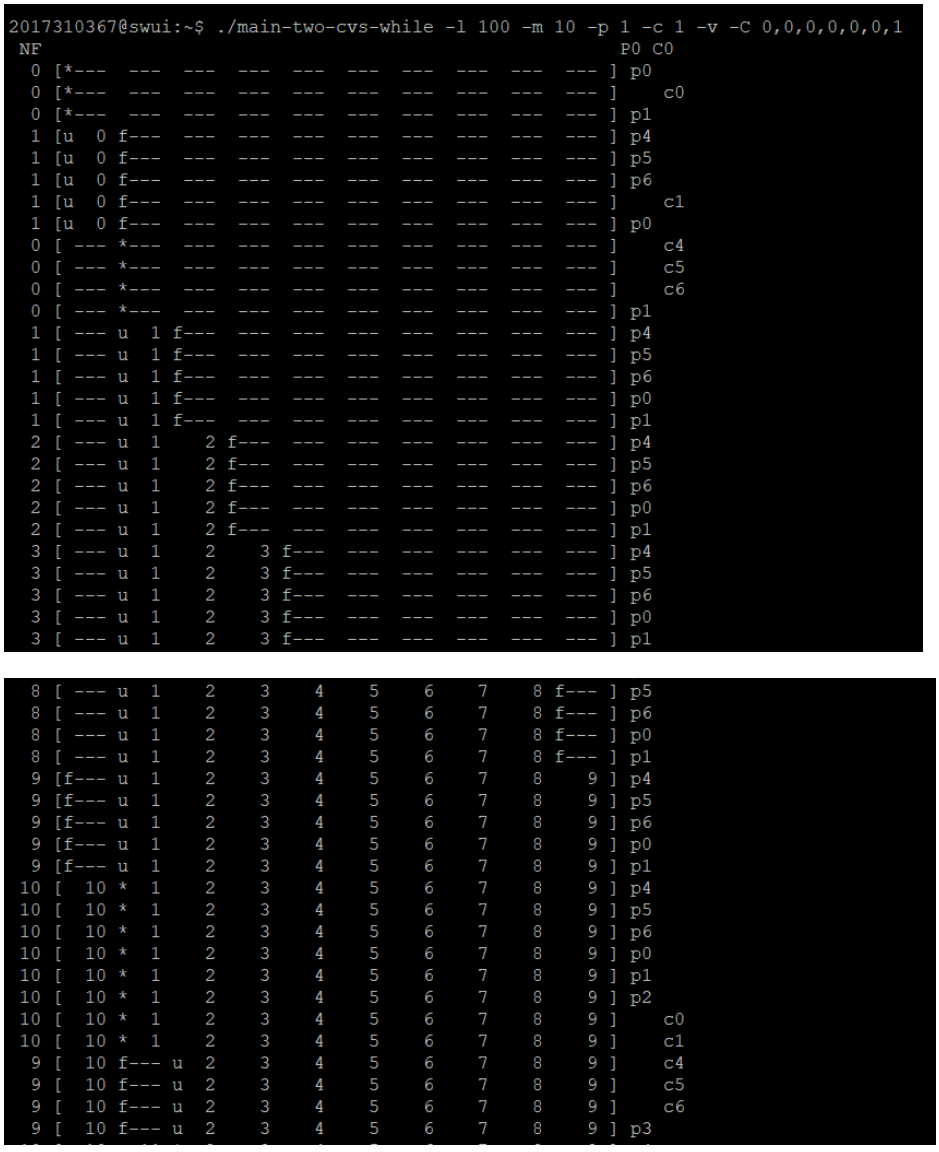
```
2017310367@swui:~$ ./main-two-cvs-while -l 3 -m 2 -p 1 -c 1 -v
 NF             P0 C0
  0 [*---  --- ] p0
  0 [*---  --- ]    c0
  0 [*---  --- ] p1
  1 [u  0 f--- ] p4
  1 [u  0 f--- ] p5
  1 [u  0 f--- ] p6
  1 [u  0 f--- ]    c1
  1 [u  0 f--- ] p0
  0 [ --- *--- ]    c4
  0 [ --- *--- ]    c5
  0 [ --- *--- ]    c6
  0 [ --- *--- ] p1
  0 [ --- *--- ]    c0
  1 [f--- u  1 ] p4
  1 [f--- u  1 ] p5
  1 [f--- u  1 ] p6
  1 [f--- u  1 ]    c1
  1 [f--- u  1 ] p0
  0 [*---  --- ]    c4
  0 [*---  --- ]    c5
  0 [*---  --- ]    c6
  0 [*---  --- ] p1
  0 [*---  --- ]    c0
  1 [u  2 f--- ] p4
  1 [u  2 f--- ] p5
  1 [u  2 f--- ] p6
  1 [u  2 f--- ]    c1
  0 [ --- *--- ]    c4
  0 [ --- *--- ]    c5
  0 [ --- *--- ]    c6
  1 [f--- uEOS ] [main: added end-of-stream marker]
  1 [f--- uEOS ]    c0
  1 [f--- uEOS ]    c1
  0 [*---  --- ]    c4
  0 [*---  --- ]    c5
  0 [*---  --- ]    c6

Consumer consumption:
  C0 -> 3
```

```
2017310367@swui:~$ ./main-two-cvs-while -l 3 -m 3 -p 1 -c 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
  0 [ --- *---  --- ]    c0
  1 [ --- u  1 f--- ] p4
  1 [ --- u  1 f--- ] p5
  1 [ --- u  1 f--- ] p6
  1 [ --- u  1 f--- ]    c1
  1 [ --- u  1 f--- ] p0
  0 [ ---  --- *--- ]    c4
  0 [ ---  --- *--- ]    c5
  0 [ ---  --- *--- ]    c6
  0 [ ---  --- *--- ] p1
  0 [ ---  --- *--- ]    c0
  1 [f---  --- u  2 ] p4
  1 [f---  --- u  2 ] p5
  1 [f---  --- u  2 ] p6
  1 [f---  --- u  2 ]    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 -> 3
```

```
2017310367@swui:~$ ./main-two-cvs-while -l 3 -m 4 -p 1 -c 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
  0 [ --- *---  ---  --- ]    c0
  1 [ --- u  1 f---  --- ] p4
  1 [ --- u  1 f---  --- ] p5
  1 [ --- u  1 f---  --- ] p6
  1 [ --- u  1 f---  --- ]    c1
  1 [ --- u  1 f---  --- ] p0
  0 [ ---  --- *---  --- ]    c4
  0 [ ---  --- *---  --- ]    c5
  0 [ ---  --- *---  --- ]    c6
  0 [ ---  --- *---  --- ] p1
  0 [ ---  --- *---  --- ]    c0
  1 [ ---  --- u  2 f--- ] p4
  1 [ ---  --- u  2 f--- ] p5
  1 [ ---  --- u  2 f--- ] p6
  1 [ ---  --- u  2 f--- ]    c1
  0 [ ---  ---  --- *--- ]    c4
  0 [ ---  ---  --- *--- ]    c5
  0 [ ---  ---  --- *--- ]    c6
  1 [f---  ---  --- uEOS ] [main: added end-of-stream marker]
  1 [f---  ---  --- uEOS ]    c0
  1 [f---  ---  --- uEOS ]    c1
  0 [*---  ---  ---  --- ]    c4
  0 [*---  ---  ---  --- ]    c5
  0 [*---  ---  ---  --- ]    c6

Consumer consumption:
  C0 -> 3
```

Buffer 사이즈가 커지면 꽉 차거나 비워질 확률이 줄어든다. 즉, buffer가 커질수록 buffer가 꽉 찰

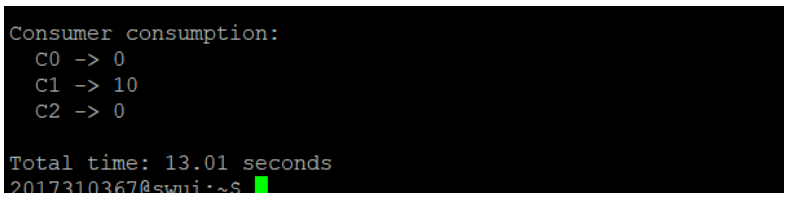
때 불려지는 wait(&empty)나 buffer가 비워져 있을 때 불러지는 wait(&fill)이 발생하지 않게 된다. 현 코드에서는 producer와 consumer가 하나씩 있고 같은 순서로 buffer를 사용하였기 때문에 일정한 코드를 보인다.



Buffer 사이즈가 10이고 produced item이 100이며 c6에서 consumer를 잠재웠을 때의 결과이다. C6에서 consumer가 잠들었기 때문에 producer는 buffer가 꽉 찰 때까지, 즉 wait(&empty)를 부를 때까지 버퍼를 채우게 된다. 만약 buffer가 꽉 차면 consumer를 깨운다. 그러나 consumer는 c6에서 다시 잠들고 비워진 buffer는 producer가 또다시 채우게 된다. 따라서 buffer 사이즈가 10이더라도 9개의 buffer는 채워진 채로 이 과정을 반복한다.

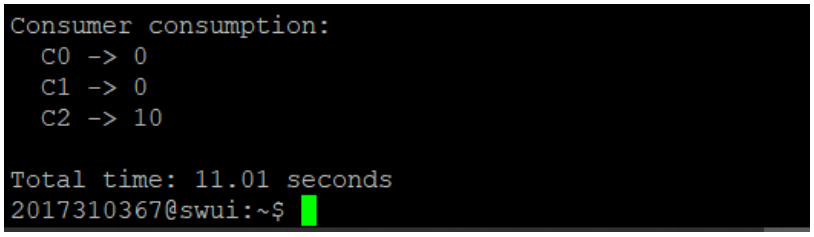
3. If possible, run the code on different systems (e.g., a Mac and Linux). Do you see different behavior across these systems?

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



1개의 producer와 1개의 consumer, 그리고 buffer size가 1이고 각 consumer들이 c3에서 잠들었을 경우, 13.01초가 걸렸다. Buffer가 하나뿐이라 wait(&empty)와 wait(&fill)이 자주 발생하여 스레드 교환이 잦았으며 이로 인해 오랜 시간이 걸린 것으로 보인다.

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



1개의 producer와 1개의 consumer, 그리고 buffer size가 3이고 각 consumer들이 c3에서 잠들었을 경우, 11.01초가 걸렸다. Buffer size가 3이기 때문에 wait(&empty)와 wait(&fill)이 4번 문제보다 덜 발생하여 이로 인해 buffer size가 1인 경우보다 짧은 시간이 걸린 것으로 보인다.

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

```
Consumer consumption:
  C0 -> 3
  C1 -> 4
  C2 -> 3

Total time: 5.00 seconds
2017310367@swui:~$
```

1개의 producer와 1개의 consumer, 그리고 buffer size가 1이고 각 consumer들이 c6에서 잠들었을 경우, 5초가 걸렸다. C6에서 이미 모든 활동을 끝내고 스레드를 넘기는 것이기 때문에 계속해서 buffer의 상태를 확인해야 하는 4번 문제와 5번 문제보다 더 적은 시간이 걸렸다.

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

```
Consumer consumption:
  C0 -> 3
  C1 -> 3
  C2 -> 4

Total time: 5.00 seconds
2017310367@swui:~$
```

1개의 producer와 1개의 consumer, 그리고 buffer size가 3이고 각 consumer들이 c6에서 잠들었을 경우, 5초가 걸렸다. C6에서 이미 모든 활동을 끝내고 스레드를 넘기는 것이기 때문에 계속해서 buffer의 상태를 확인해야 하는 4번 문제와 5번 문제보다 더 적은 시간이 걸렸으며 buffer의 크기가 영향을 주지 않았다.

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?

1개의 producer와 1개의 consumer, 그리고 buffer size가 1일 경우 main-one-cv-while.c에는

아무 문제없다.

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?

만약 2개의 consumer 일 경우에는 문제가 발생한다. P3에 sleep을 주면 첫번째 consumer는 if절을 통과했으므로 buffer에서 아이템을 가져간 후 다시 가져갈 아이템이 없으므로 wait하게 된다. 그 후 두번째 consumer가 깨어나면 역시 가져갈 아이템이 없으므로 wait하게 된다. 따라서 Producer를 깨워줄 consumer가 모두 잠든 상태가 되어버린다.

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.

역시 1개의 producer와 1개의 consumer, 그리고 buffer size가 1일 경우 main-one-cv-while.c에는 아무 문제없다. 그러나 만약 2개의 consumer 일 경우에는 문제가 발생한다.

```
2017310367@swui:~$ ./main-two-cvs-if -p 1 -c 2 -m 1 -C 0,0,0,3,4,0,0:0,0,0,3,3,0,0 -P 7,0,0,0,0,0,0 -l 3 -v -t
 NF        P0 C0 C1
  0 [*--- ] p0
  0 [*--- ]    c0
  0 [*--- ]       c0
  0 [*--- ]    c1
  0 [*--- ]    c2
  0 [*--- ]       c1
  0 [*--- ]       c2
  0 [*--- ] p1
  1 [*  0 ] p4
  1 [*  0 ] p5
  1 [*  0 ] p6
  1 [*  0 ]    c3
  1 [*  0 ] p0
  0 [*--- ]    c4
  0 [*--- ]    c5
  0 [*--- ]    c6
  0 [*--- ] p1
  0 [*--- ]    c0
  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
error: tried to get an empty buffer
2017310367@swui:~$
```

위 코드와 같이 c3에서 sleep를 주는 경우 첫번째 consumer가 이미 buffer에서 가져갔음에도 두번째 consumer도 이미 while문을 통과했기 때문에 buffer에서 아이템을 가져가고자 한다. 그렇기에 empty buffer라고 오류가 나게 된다.

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?