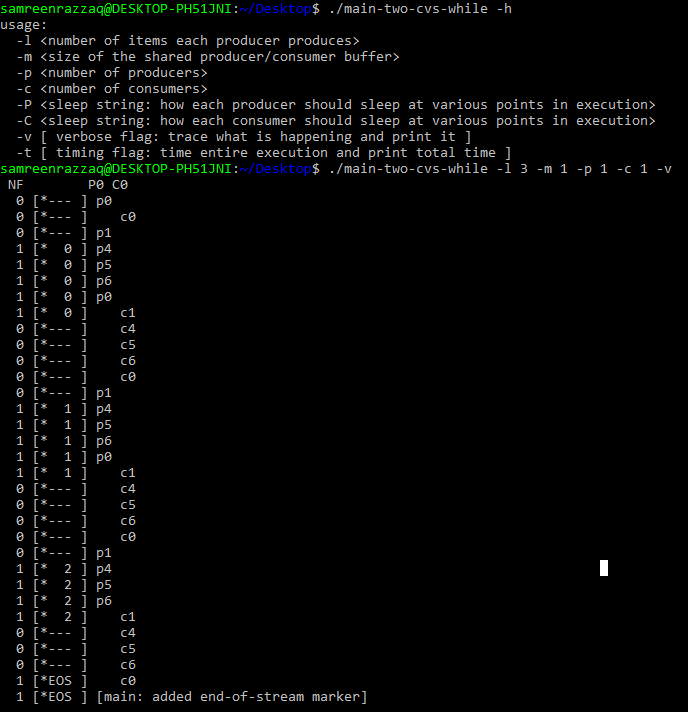


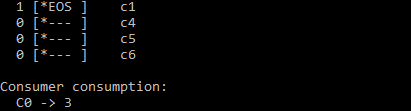
**Lab Task:**

This lab lets you explore some real code that uses locks and condition variables to implement various forms of the producer/consumer queue discussed in the chapter. You’ll look at the real code, run it in various configurations, and use it to learn about what works and what doesn’t, as well as other intricacies. Read the README for details.

**Tasks:**

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?

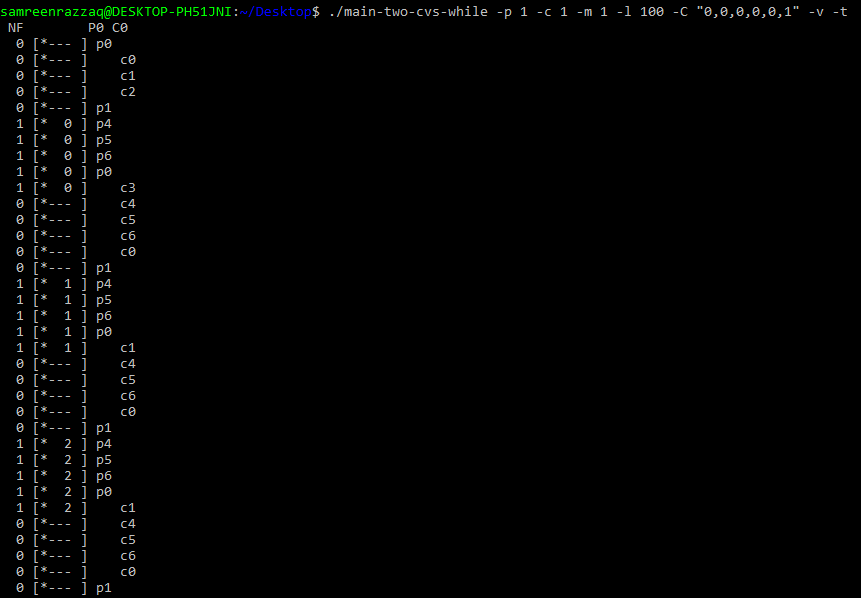


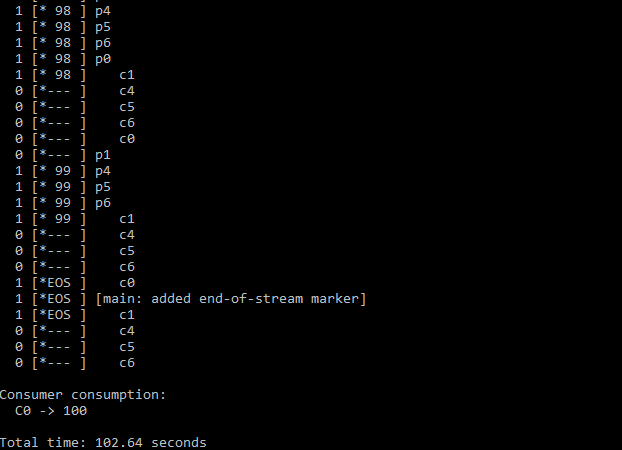


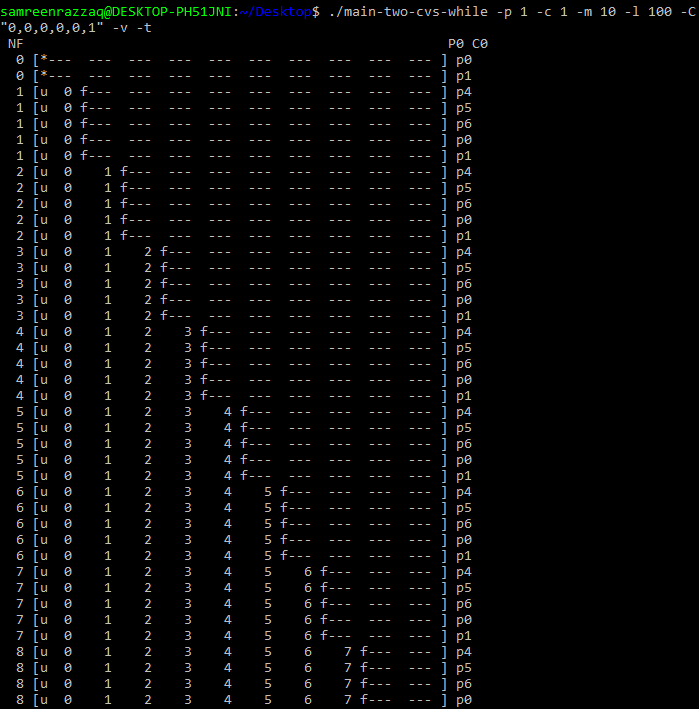
**Explanation:**

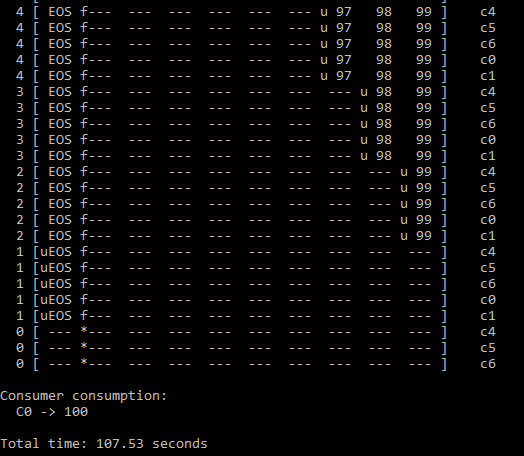
The provided output suggests a C program ("main-two-cvs-while.c") involving multiple producers and consumers. Producers generate items, consumers consume them. Each thread is denoted by "p" or "c" followed by a number. The output displays the state of these threads over time, with indicators like "[\*--- ]" and "[\* 0 ]". The program seems to operate in a continuous loop, and the end-of-stream marker indicates the conclusion of data processing. Consumer 0 (C0) consumes at a rate of 3 items.

1. 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?



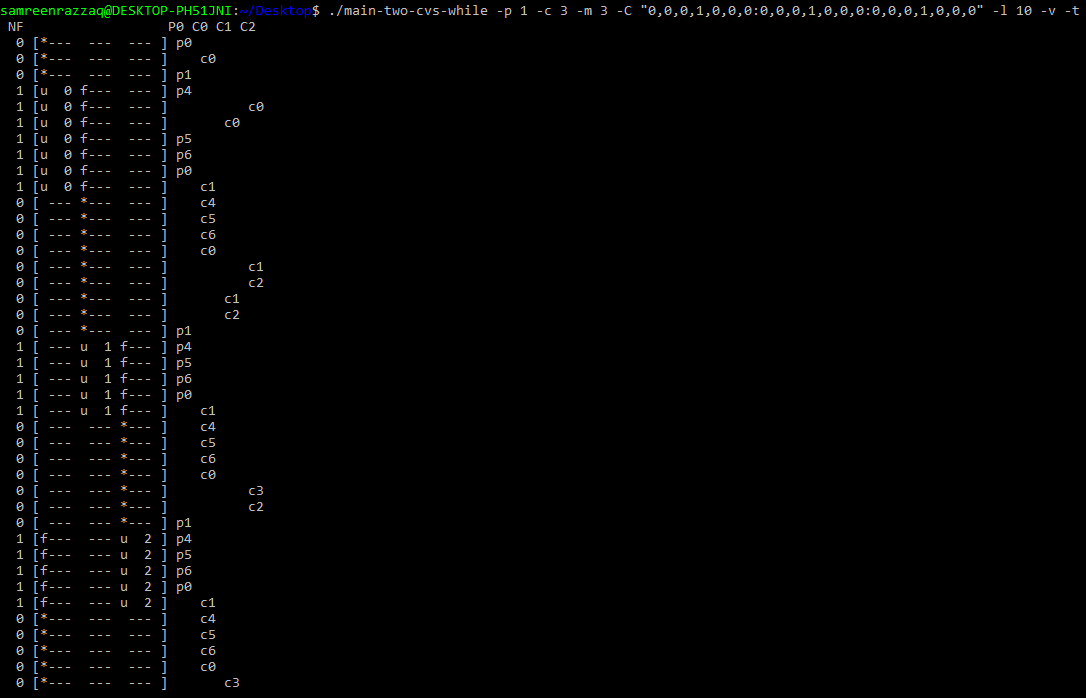


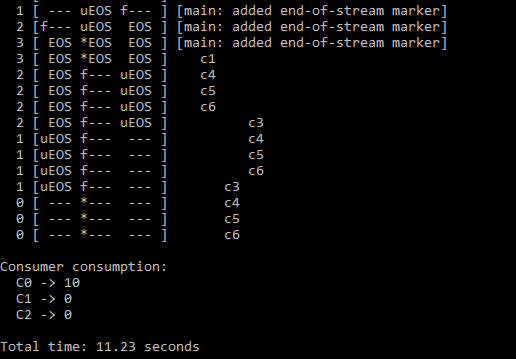
****

****

**Explanation:**

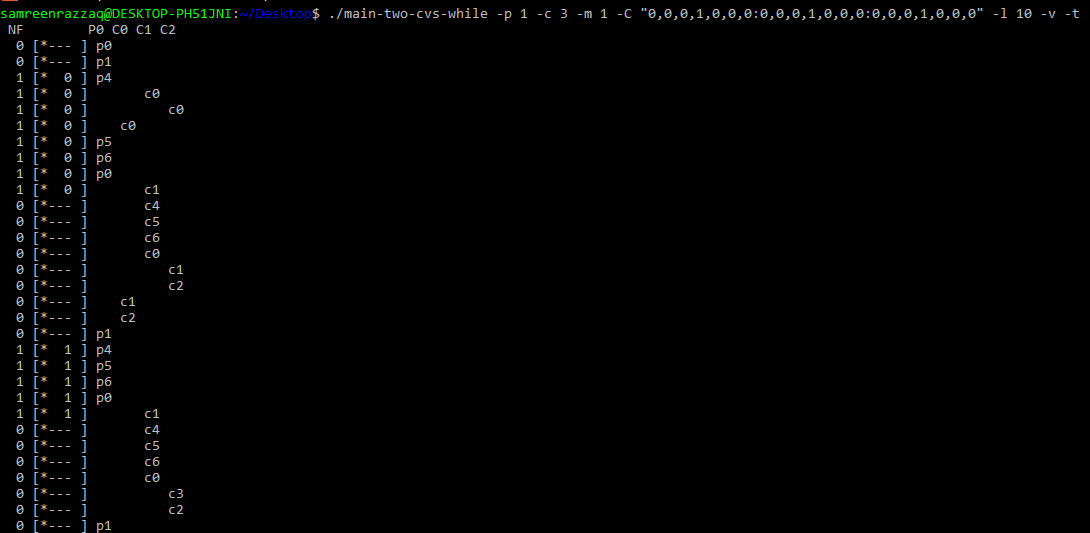
Running the program with one producer and one consumer initially operates smoothly, but adjusting the buffer size influences synchronization and the rate at which items are produced and consumed. Increasing the buffer size allows more items to be stored, potentially affecting the pattern of buffer fill and consumption. Predicting the variable `num\_full` involves considering different buffer sizes and produced item counts. Altering the consumer sleep string can impact the consumption rate, potentially leading to desynchronization between the producer and consumer threads. In summary, varying buffer sizes and adjusting the consumer sleep string influence the efficiency and synchronization of the producer-consumer interaction in the program.

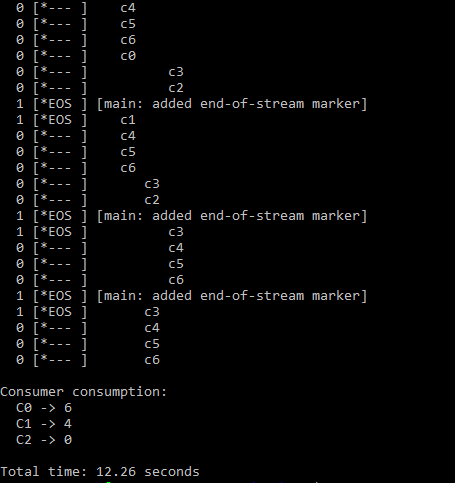
1. Let’s look at some timings. How long do you think the following execution, with one producer, three consumers, a single-entry shared buffer, and each consumer pausing at point c3 for a second, will take? ./main-two-cvs-while -p 1 -c 3 -m 1 -C 0,0,0,1,0,0,0:0,0,0,1,0,0,0:0,0,0,1,0,0,0 -l 10 -v -t



**Explanation:**

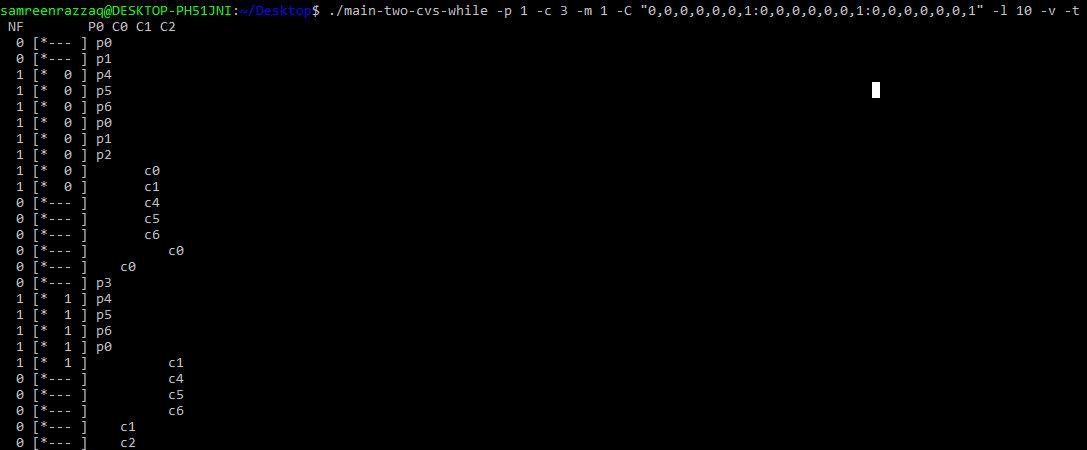
The provided execution scenario involves a C program with one producer, three consumers, a single-entry buffer, and each consumer pausing for a second at point c3. The overall execution time depends on the coordination between the producer and consumers, the specified buffer size, and the one-second pauses introduced. The time taken will be **“11.23 seconds”.**

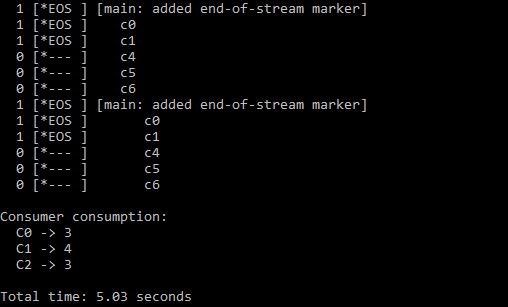
1. Now change the size of the shared buffer to 3 (-m 3). Will this make any difference in the total time?



**Explanation:**

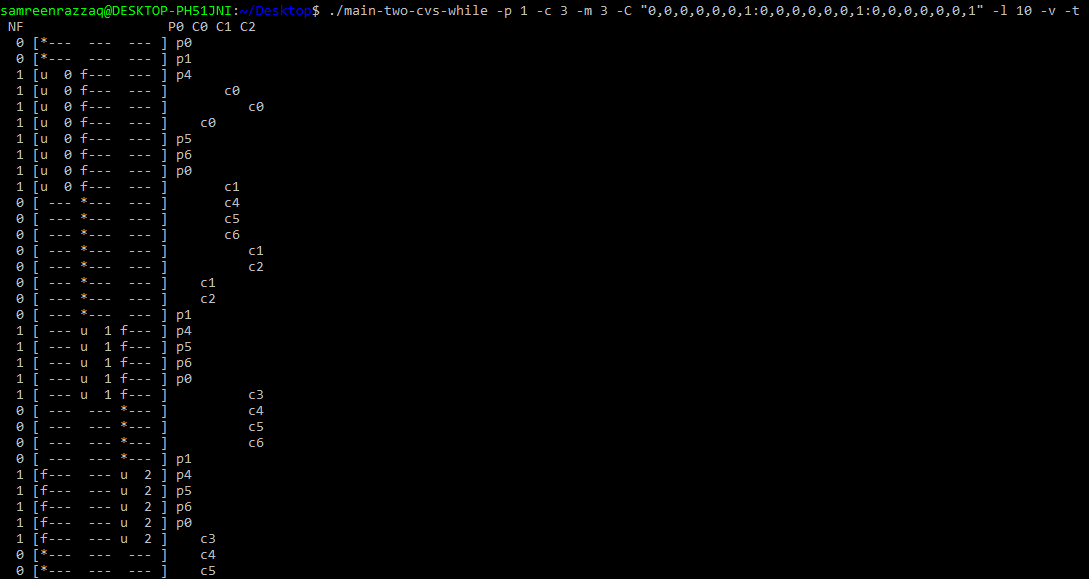
The time will be reduced by increasing the size of shared buffer. Now the time will be **“12.26 seconds”.**

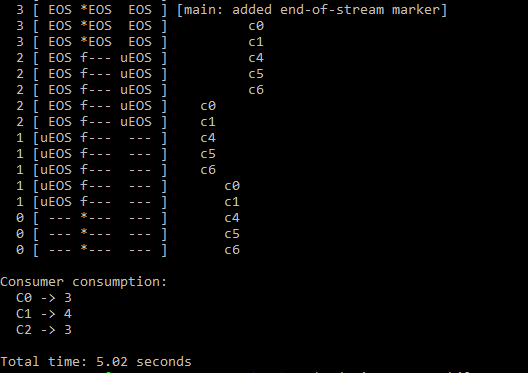
1.  Now change the location of the sleep to c6 (this models a consumer taking something off the queue and then doing something with it), again using a single-entry buffer. What time do you predict in this case? ./main-two-cvs-while -p 1 -c 3 -m 1 -C 0,0,0,0,0,0,1:0,0,0,0,0,0,1:0,0,0,0,0,0,1 -l 10 -v -t



**Explanation:**

The time taken in this case will reduce again to a great extent. It will be **“5.03 seconds”.**

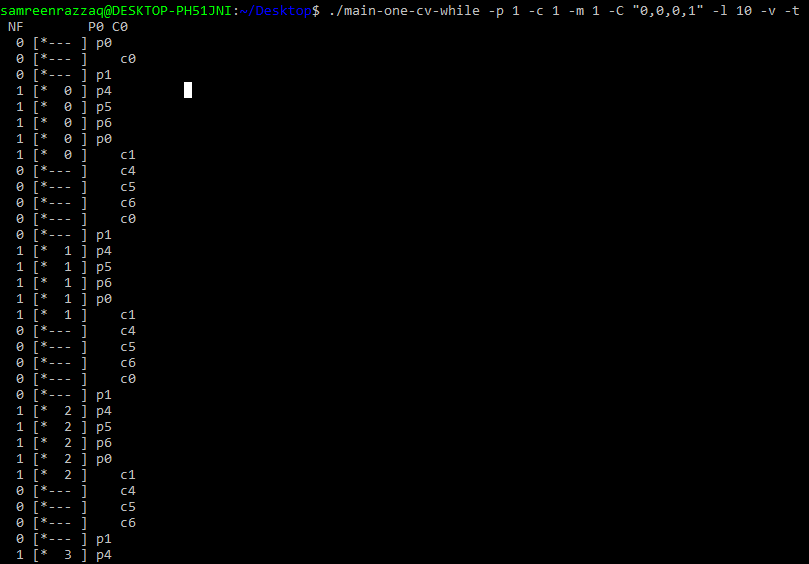
1. Finally, change the buffer size to 3 again (-m 3). What time do you predict now?

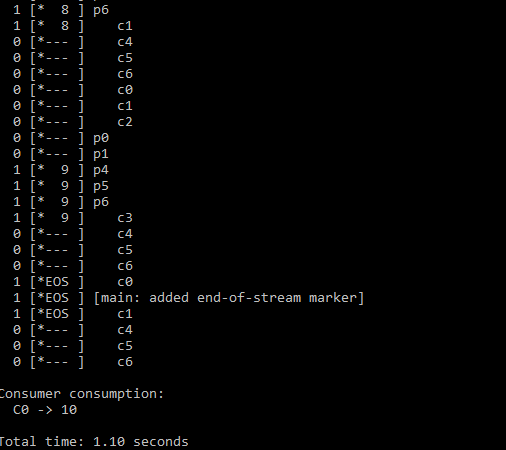


**Explanation:**

This time changing the size of shared buffer will have no effect on the time. It will be the same **“5.02 seconds”.**

1. 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?

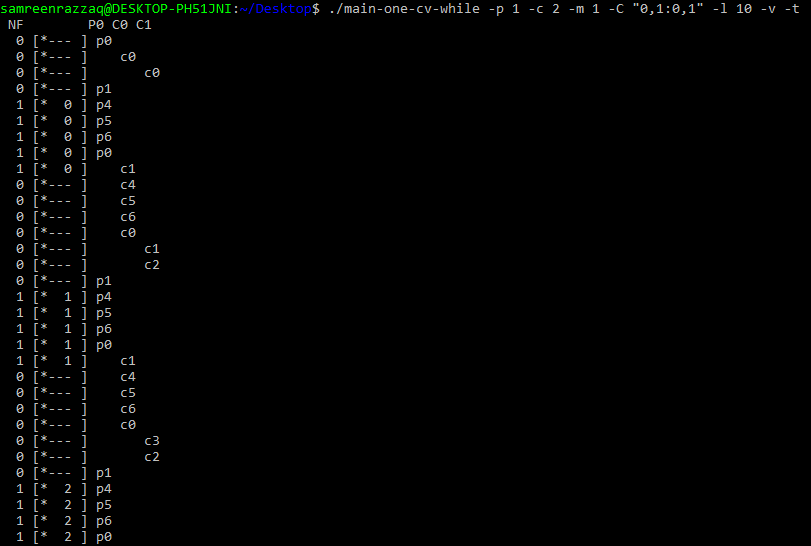


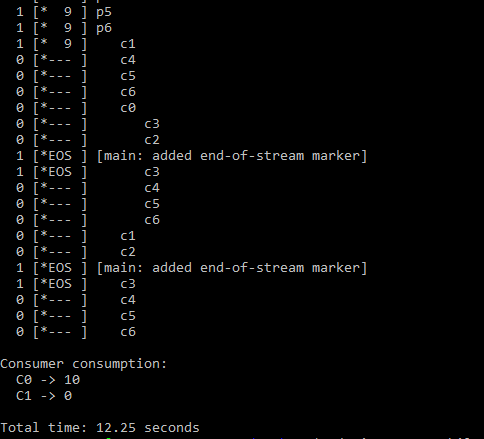


**Explanation:**

The sleep string "-C "0,0,0,1"" indicates a 1-unit pause for the consumer. In this scenario, with a small buffer, this sleep configuration may cause synchronization issues. If the consumer pauses, and the producer tries to add an item during that pause, it could lead to problems like slowdown or deadlock due to insufficient buffer space.

1. 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?

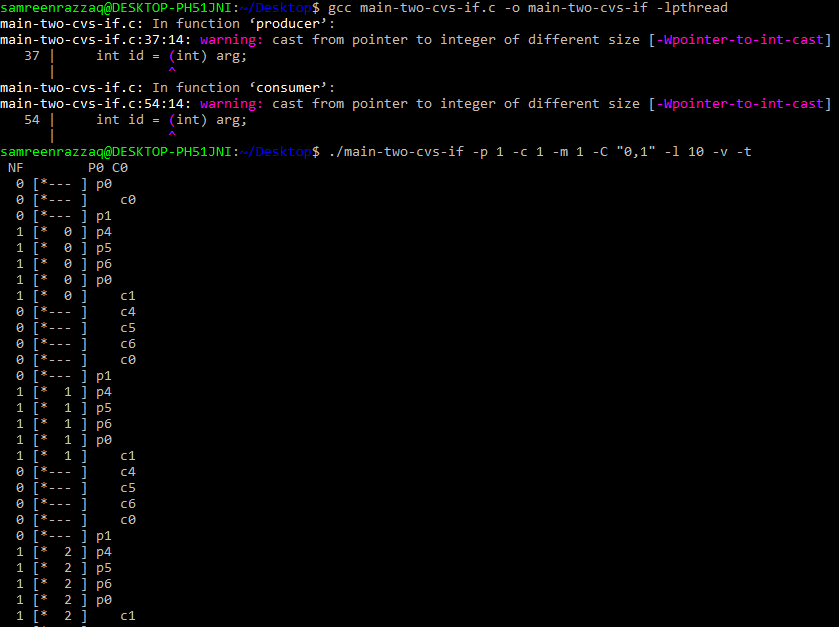


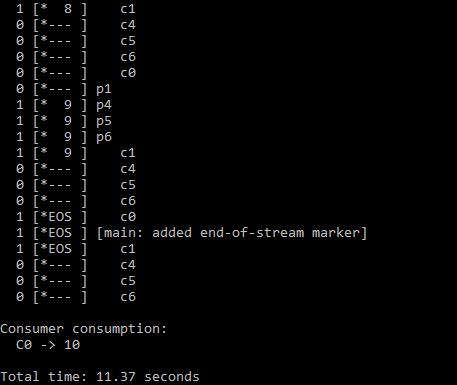


**Explanation:**

The sleep string is adjusted as **“0,1:0,1”**. By adjusting sleep times strategically, the goal is to disrupt the intended coordination between the producer and consumers, leading to issues like deadlock or race conditions in the execution of the code.

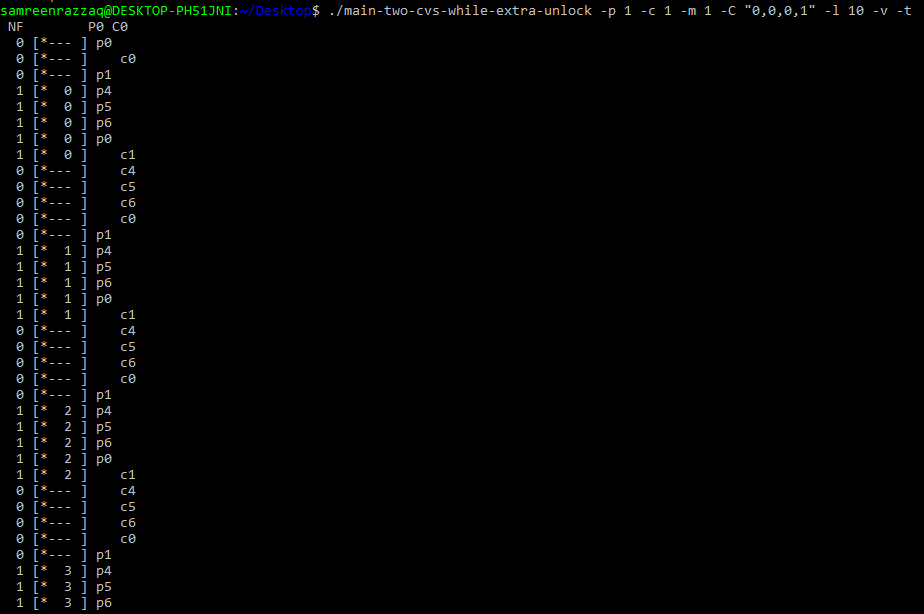
1. Now examine main-two-cvs-if.c. Can you cause a problem to happen in this code? Again consider the case where there is only one consumer, and then the case where there is more than one.

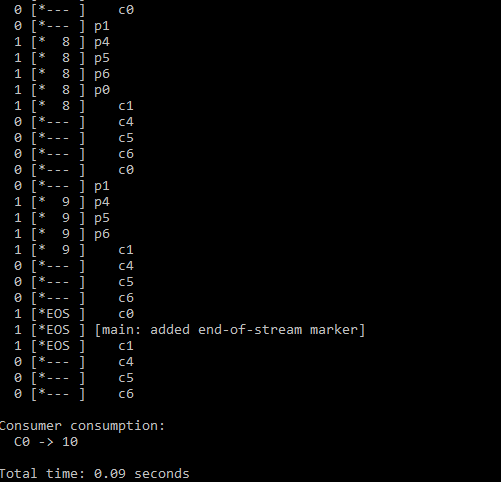




**Explanation:**

The goal of this problem is to explore sleep configurations that may disrupt intended synchronization in "main-two-cvs-if.c."

1. 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?



**Explanation:**

The goal of this question is to reliably identify and cause problems, considering the provided sleep strings. Releasing the lock too early can lead to race conditions, potentially causing data corruption or other concurrency-related issues.