Worksheet 5.3: Readers-Writers Problem

Consider the following solution studied in class for the **Readers-Writers Problem:**

sem_1 = 1; sem_2 = 1; counter = 0;

```
        Writer() {
          printf("\n Writer arrived");
1         wait(sem_1);
          printf("\n Performing writing");
2         perform_writing ();
3         signal(sem_1);
        }

        Reader() {
1         wait(sem_2);
          printf("\n Reader arrived");
2         counter++;
3         if (counter == 1)
                    printf("\n Checking semaphore");
4                   wait(sem_1);
5         signal(sem_2);

          printf("\n Started reading");
6         perform_ reading ();
          printf("\n Done reading");

7         wait(sem_2);
8         counter--;
9         if (counter == 0)
                    printf("\n Releasing semaphore");
10                  signal(sem_1);
11        signal(sem_2);
        }
```

1. Considering the print statements added to the above code, show the output printed by the code if the following sequence of events takes place (order is very important). Assume for simplicity that the print statements for each event execute completely before the next event occurs, that is, events are not interleaved with prints. Note that indentation in the above code is significant. So, all the indented lines after an **if** statement get executed if the condition is true.

   Reader 1 arrives
   Reader 2 arrives
   Writer arrives
   Reader 1 done reading
   Reader 3 arrives

   Reader arrived
   Checking semaphore
   Started reading

   Reader arrived
   Started reading

   Writer arrived

   Done reading

   Reader arrived
   Started reading

2. The above solution may cause starvation. Will it cause the starvation of readers or the starvation of writers? Give a specific scenario (sequence of events) that will cause starvation. **(Limit: 3 Lines)**

   It may cause the starvation of writers. A possible scenario is
   Reader arrives and locks the buffer for reading. Then writer arrives
   Then an unlimited number of readers arrive and access the buffer. Writer will starve

3. Trace the consequences of replacing signal(sem_2) on Line 11 of the **Reader** with signal(sem_1). Cover **all** the consequences and indicate whether the code will work correctly or not. If certain readers/writers get stuck, **specify the line** at which they will get stuck. You must cover two cases:

   Case 1: There is one reader in the buffer
   Impact on Readers: (**Limit: 2 Lines)**

   The reader in the buff will exit the buff but won't release the reader semaphore (sem_2). So, all subsequent readers will get blocked on Line 1 of Reader.

   Impact on Writers: (**Limit 2 Lines)**

   The reader in the buff will signal the reader/writer sem (sem_1) twice, thus allowing multiple writers to access the buff at the same time.

   Case 2: There are multiple readers in the buffer
   Impact on Readers: (**Limit: 3 Lines)**

   The first reader to complete reading will exit the buff but without releasing the reader sem (sem_2). So, all other readers in the buff will get blocked on Line 7 in the reader when they try to exit.
   Readers that arrive after the first reader exits will get stuck on Line 1 in the reader.

   Impact on Writers: (**Limit 2 Lines)**

   The first reader to exit will signal the reader/writer buff, thus making it possible for a writer to access the buffer while there are readers in the buff.

4. Modify the above code to prevent starvation with minimal changes. Show the changes on the given code.

   Sem fairness = 1;

```
   Writer() {
     wait (fairness);
1    wait(sem_1);
     signal(fairness);
2    perform_writing ();
3    signal(sem_1);
   }

   Reader() {
     wait(fairness);
1    wait(sem_2);
2    counter++;
3    if (counter == 1)
4            wait(sem_1);
5    signal(sem_2);
     signal(fairness);

6    perform_ reading ();
```

```
7         wait(sem_2);
8         counter--;
9         if (counter == 0)
10                  signal(sem_1);
11         signal(sem_2);
       }
```