University of Regina
Computer Science

Bao Cao
200363431

CS 330 - Assignment 6

Question 3:

a)

p	q	wantp	wantq
*p1	q1	f	f
p2	*q1	f	f
*p2	q2	f	f
p3	*q2	t	f
p3	*q3	t	t
*p3	q4	t	t
p4	*q4	t	t
*p4	q5	t	f
p5	*q5	f	f
*p5	q3	f	t
p3	q3	t	t

b)

p	q	wantp	wantq
*p1	q1	f	f
p2	*q1	f	f
*p2	q2	f	f
p3	*q2	t	f
p3	*q3	t	t
*p3	q4	t	t
p4	*q4	t	t
*p4	q5	t	f
p5	*q5	f	f

*p5	q6	f	t
p3	*q6	t	t
*p3	q7	t	t
p4	*q7	t	t
*p4	q8	t	f
p5	*q8	f	f
*p5	q1	f	f
p3	*q1	t	f
p3	*q2	t	f
p3	q3	t	t

Question 4:

$(p1, q1, f, f, 1) \xrightarrow{p1} (p2, q1, f, f, 1) \xrightarrow{q1} (p2, q2, f, f, 1)$

$\xrightarrow{p2} (p3, q2, t, f, 1) \xrightarrow{q2} (p3, q3, t, t, 1) \xrightarrow{p3} (p4, q3, t, t, 1)$

$\xrightarrow{q3} (p4, q4, t, t, 2) \xrightarrow{q4} (p4, q5, t, t, 2) \xrightarrow{q5} (p4, q6, t, t, 2)$

$\xrightarrow{q6} (p4, q1, t, f, 2) \xrightarrow{q1} (p4, q2, t, f, 2) \xrightarrow{q2} (p4, q3, t, t, 2)$

$\xrightarrow{q3} (p4, q4, t, t, 2)$

Correctness:

Mutual exclusion: since await is controlled by 2 variable, even if there is a state when both waitp and waitq are true, the next statement executed will set the last variable of the other process to put it in await. Therefore, two process cannot go into critical section at the same time.

Freedom from deadlock: holds because before going into critical section, both process set wantp and wantq to true. No matter which statement is executed next, one process will be put into await and the other can go into critical section.

Freedom from starvation: not hold because from the above diagram, after q went into critical section, last will never be set back to 1, which make p stuck in await.

Question 5:

$$\begin{tikzcd}
\text{(p1, q1, 0, 0)} \arrow[r, "p1"] & \text{(p2: if, q1, 0, 0)} \arrow[r, "q1"] & \text{(p2: if, q2: if, 0, 0)} \\
{} \arrow[r, "p2"] & \text{(p2: else, q2, 0, 0)} \arrow[r, "q2"] & \text{(p2: else, q2: else, 0, 0)} \\
{} \arrow[r, "p2"] & \text{(p3, q2: else, -1, 0)} \arrow[r, "q2"] & \text{(p3, q3, -1, -1)}
\end{tikzcd}$$

After the last state from the above diagram, both p and q can go into critical section ——> the algorithm is not correct.

Question 6:

a) if p1 is executed before q1: p q

if p1 is executed after q1: q is blocked. Therefore the output is still p q

b) p q

c) if p1 is executed before q2: p q

if p1 is executed after q2: q p

Question 7:

p is always printed first and then either q or r. Therefore the possible outputs are p q r or p r q

Question 8:

If p1 is executed first, q is blocked until B is set to true and p3 is executed. Therefore nothing will be printed.

If q1 is executed first, the condition of while loop is always true (B is false). Therefore, "*" will be printed infinitely.

Question 9:

Set n < k is the number of processes in the critical section. In order to enter the critical section, all n processes have to execute wait and by the definition of wait, S.V is decremented n times. If all k processes enter the critical section, S.V = 0. Assume there is more than k processes. If any of them want to enter the critical section, they will be added to the set of blocked processes (since S.V = 0). After one of k processes in the critical section finishes executing, it will execute signal, and since S.L is not empty, one of blocked processes will be unblocked and enter the critical section. If one process call signal when S.L is empty, S.V will be incremented to 1, which mean there are k -1 processes are in the critical section. When these k - 1 processes finish in critical section, they will call signal which results S.V incremented back to k. Therefore, there are no more than k processes can enter the critical section at the same time.

Question 10:

Assume notFull.L is empty. The consumer before remove d from buffer has to be blocked and put into notEmpty.L. The producer in order to put d into buffer has to do wait (notFull) which decreases notFull.V by 1 and increase inBuffer by 1. After the call signal(notEmpty) from the producer, the consumer is unblocked and removes d from buffer, which decreases inBuffer by 1. After that, the consumer does signal(notFull) which increases notFull.V by 1. Therefore, notFull.V is invariant.

Question 11:

By the definition of StartRead, when there is a writer who is waiting to write, a reader will be blocked, which gives the first blocked writer precedence over waiting readers. When there is no more readers, signalC(OKtoWrite) is executed to unblocked a writer. Therefore, there is no starvation of writers in the readers-writers algorithm.

Question 12:

a)

```
operation StartRead
    lock <- true
    readers <- readers + 1
    if emptyC (OKtoRead)
        lock <- false
    else
        signalC (OKtoRead)
```

b)

```
operation StartRead
    lock <- true
    if writers != 0 or not emptyC (OKtoWrite)
        waitC (OKtoRead)
    else
        readers <- readers + 1
        if emptyC (OKtoRead)
            lock <- false
        else
            signalC (OKtoRead)
```

c)

```
operation StartRead
    lock <- true
    if not emptyC (OKtoRead)
        if readers != 0 and not emptyC (OKtoWrite)
            waitC (OKtoRead)
        if writers != 0 and writer > 2
            waitC (OKtoWrite)
            signalC (OKtoRead)
    else
        lock <- false
```