CMPUT 379 Midterm Exam [Harms]
February 28, 2001
Closed Book

Comments:

- This exam is worth 22% of your final grade. There are 3 questions and 4 pages. The mark distribution is given beside the questions. The total number of marks is 30.
- This is closed book exam. Calculators are not necessary nor allowed.
- If you are concerned about an interpretation of an exam question,ask me or state your assumptions and then answer the question.
- Be sure to show your work! Good Luck!

(6 marks) Question 1.

Multilevel Feedback Queues have been used in many operating systems for CPU scheduling. The variation that we discussed in class uses round robin within each queue and priority scheduling between the queues. Queue i's quantum is 2^i ($i = 1, ..., n$) where n is the number of queues. The scheduling is pre-emptive.

a) Explain the feedback mechanism (that is, explain the ways in which a process can move from one queue to another).

b) Discuss the advantages and disadvantages of this method of setting the quantum for each queue (that is, doubling the quantum for each subsequent queue).

c) Explain how starvation can occur in this scheme. Suggest a solution to the starvation problem.

(10 marks) Question 2.

a) What are the 4 necessary conditions for deadlock to occur?

b) Suppose we have a system with 3 resources (R1, R2, R3) and 3 processes (P1, P2, P3).

- There are 3 instances of R1, 2 instances of R2 and 4 instances of R3 (these totals include both allocated and non-allocated resource instances).

At time **T1**, the following allocations have already been made:

- P1 holds 2 instances of R1 and 1 instance of R2
- P2 holds 1 instance of R1 and 1 instance of R3
- P3 holds 1 instance of R2 and 2 instances of R3

At time **T2**, P2 makes a request for 1 instance of R1 and 1 instance of R2
At time **T3**, P3 makes a request for 1 instance of R1.

Now we are at time **T4**

(i) Draw the resource allocation graph representing the system at time T4.

(ii) Is there deadlock at time T4? Explain why or why not.

(iii) Suppose that the **maximum** total requirements for the processes are:

- P1: 2 instances of R1, 1 instance of R2, and 1 instance of R3
- P2: 2 instances of R1, 1 instance of R2, and 3 instances of R3
- P1: 3 instances of R1, 1 instance of R2, and 2 instances of R3

Using the Banker's algorithm, would the system be considered **safe** at time T4 or not? Show your work.

(14 marks) Question 3.

In class we discussed the Readers-Writers (R-W) synchronisation problem and studied a solution to the First R-W problem that used semaphores for synchronisation and mutual exclusion.

The ***First Readers-Writers (1st R-W)*** Problem can be described by the following 3 points:

Reader Requirement : *a* reader can share access with other readers

Writer Requirement : a writer needs exclusive access (no other writers or readers)

Priority Requirement : no reader will be kept waiting unless a writer has already obtained access permission

a) This problem (by its very definition) can suffer from starvation. Explain how?

b) Suppose that the following modification is made to the priority requirement in the problem definition above. The other requirements remain the same.

Priority Requirement : no reader will be kept waiting unless a writer has already obtained access permission **or unless > 4 writers are waiting**

(in other words, a new reader must wait in two cases: a writer is writing or >4 writers are waiting)

This will be called the ***Modified First Readers-Writers*** (***Mod 1st R-W***) Problem. Does this solve the starvation problem? Explain why or why not?

c) On the next page is an attempt to provide a solution to the **Mod 1st R-W** problem. You will also find the semaphore functions of wait() and signal() defined (almost) as in class. The difference is that the semaphore queues are assumed to be FIFO.

One of the problems of this solution is that deadlock may occur. Explain how (be specific).

d) Provide a correct solution (in pseudocode) of the **Mod 1st R-W** problem by modifying the incorrect solution or providing a completely new solution. Clearly deadlock should not occur but also be certain to meet the reader, writer and modified priority requirements.

```
Writer ()
{

(w1)     while (1) {
(w2)           wait (wmutex);
(w3)           writerwait++;
(w4)           signal (wmutex);
(w5)           wait (wrt);
(w6)           wait (wmutex);
(w7)           writerwait --;
(w8)           signal (wmutex);
               WRITE
(w9)           signal (wrt);
               . . .           }
}
```

```
Reader ()
{

(r1)     while (1) {
(r2)           wait (wmutex);
(r3)           if (writerwait > 4)
(r4)               wait (wrt);
(r5)           signal (wmutex);
(r6)           wait (rmutex);
(r7)           readcnt ++;
(r8)           if (readcnt == 1)
(r9)               wait (wrt);
(r10)          signal (rmutex);
               READ
(r11)          wait(rmutex);
(r12)          readcnt --;
(r13)          if (readcnt == 0)
(r14)              signal (wrt);
(r15)          signal (rmutex);
               . . .            }
}
```

<u>Initialisations</u>

Semaphores: wmutex, rmutex, wrt
(all initialised to 1)

Global variables: writewait = 0, readcnt = 0

```
wait (s)
{
    s.value = s.value - 1;
    if (s.value < 0) {
        place process at end of FIFO queue
        block process
    }
}
```

```
signal (s)
{
    s.value = s.value + 1;
    if (s.value <= 0) {
        remove 1st process on FIFO queue
        wakeup process
    }
}
```