CS471: Operating System Concepts
Fall 2006
(Lecture: TR 11:25-12:40 PM)
Homework #3
Points: 20
Due: September 19, 2006

Question 1 [Points 10] Exercise 6.1: Provide an argument to show that the algorithm satisfies all three requirements for the critical section problem.

Answer: To prove property (a) we note that each pi enters the critical section only if the *flag*[j] = *false.* Since only pi can update *flag*[j], and since pi, inspects *flag*[j] only while *flag*[j] = *true,* the results follows

a. To prove property (b), we first note that the value of variable *turn* is changed only at the end of the critical section. Suppose that only process pi wants to enter the critical section. In this case, it will find *flag*[j] = *false* and immediately enter the critical section, independent of the value of the *turn*. Suppose that both processes want to enter the critical section and the value of *turn* = 0. Two cases are now possible. If pi finds *flag*[0] = *false* then it will enter the critical section. If pi finds the *flag*[0] = *true* then it will enter the critical section. If pi finds *flag*[0] = *true* then we claim that p0 will enter critical section before p1. Moreover, it will do so within a finite amount of time.

b. To demonstrate this fact, first consider process p0. Since *turn* = 0, it will only be waiting for *flag*[1] to be set to false; more precisely, it will not be executing in the *begin* block associated with the *if* statement. Furthermore, it will not change the value of *flag*[0]. Meanwhile, process p1 will find *flag*[0] = *true* and turn = 0. It will set *flag*[1] = *false* and wait until *turn* = 1. At this point, p0 will enter the critical section. A symmetric argument can be made if *turn* =1.

Question 2 [Points 5] Exercise 6.9: Show that, if the wait and signal operations are not executed atomically, then mutual exclusion may be violated.

Answer: Consider the wait(S) semaphore operation defined in page 202 of the textbook. Suppose wait(S) is not executed atomically. Then there will be several problems. As an example consider the statement S→value--. This statement actually consists of 3 CPU instructions.

LOAD S →value
SUB 1
STO S→ value

If two processes Pi and Pj execute the wait statement and if the S→value is 1 before Pi and Pj execute the wait statement. We know how the final value of

S→value may be either -1 or 0 depending on how the CPU instructions of Pi and Pj are interleaved. Only -1 value is correct. If the value was 0, then there would be 1 process in the list but the value would show that there are now. This creates a problem.

Similarly, if signal(S) is not implemented atomically, due to similar problems (e.g., S→value++), mutual exclusion condition could be violated due to incorrect value.

Question 3 [Points 5] Exercise 6.13: Write a bounded-buffer monitor in which the buffers (portions) are embedded within the monitor itself.

Answer:

```
Monitor boundedbuffer
{
        Const bufsize = 100;
        item buf[100];
        int in, out, count;
        condition Producer, Consumer;

         void produce (item x)  {

            if (count == bufsize) Producer.wait();
            buf[in] = x;
           count++;
           in = (in + 1) % n;
           Consumer.signal();
        }

         void consume (item& x)   {
             if (count == 0) Consumer.wait();
             x = buf[out];
             count --;
             out = (out + 1) % n;
             Producer.signal();
        }

        initialization_code() {
             in:=0;
             out:=0;
             count:=0;
          }
}
```