CSE 410 Assignment 7

Spring 2008

Due: Midnight, June 2, Monday

Synchronization is an important topic and is somewhat tricky. Since these questions involve no actual program construction, be sure to take your time and clearly explain your answers.

1. One semaphore implementation we covered in class was a spinlock – what is it? What are other techniques to accomplish waiting without "spinning"? Should spinlocks be completely avoided? Give a brief explanation for your answers.

2. Consider the following Acquire/Release implementation by busy-waiting. Explain why such implementations with busy waiting are usually not appropriate for single-processor systems, yet is often used in multiprocessor systems.

```
Acquire() {                              Release() {
   while(TestAndSet(value)) {               value = 0;
      // noop                             }
   }
}
```

3. Consider the following Acquire/Release implementation using primitives provided by the kernel for disabling/enabling interrupts. Implementing synchronization primitives by disabling interrupts is not always appropriate in a single-processor system. In such a system, what could happen if the following synchronization primitives are available in user-level programs and interrupts are disabled?

```
Acquire() {                              Release() {
    kernel_disable_interrupts();             kernel_enable_interrupts();
}                                        }
```

4. Explain indefinite postponement in your own words.

5. (Silberschatz 6.11)

6. (Silberschatz 6.22)

See the next page →

7. [Dining Philosophers, Dijkstra] Consider five philosophers who spend their lives thinking and eating. The philosophers share a circular table surrounded by five chairs, each belonging to one of the philosophers. In the center of the table is a bowl of rice, and the table is laid out with five single chopsticks, one between each of the philosopher's chairs. When a philosopher thinks, she does not interact with her colleagues. When a philosopher is hungry, she may eat if she can pick up the two chopsticks immediately to her left and to her right. However, a philosopher may only pick up one chopstick at a time and obviously cannot pick up a chopstick that is already being held by a neighbor. When a hungry philosopher has both chopsticks at the same time, she eats without releasing her chopsticks. When she is finished eating, she puts down both of her chopsticks and starts thinking again.
(a) Describe how a deadlock could occur in this situation, leading to starvation of one or more philosophers because she (they) cannot acquire both of their chopsticks in order to eat.
(b) Discuss how the four necessary conditions for deadlock indeed hold in this setting. Discuss how deadlocks could be avoided by eliminating any one of the four conditions.

8. **[Extra credit]** Construct a multithreaded example (diagram or timeline) to demonstrate that if `wait()` and `signal()` operations are not executed atomically, then mutual exclusion may be violated.

See the next page →

9. **[Extra credit]** Suppose that you are provided with a SPIM pseudoinstruction "cas" that takes three arguments, one memory address through a register, and two other registers. Simply, it is used to atomically set the value of a word in memory to a new value if its current value is equivalent to a given one and returns the result of the operation (1 on success and 0 on failure) in $v0. For example, cas 0($t1), $t2, $t3 does the following: If the word in the address 0($t1) is equal to what is in $t2, it writes what is in $t3 to 0($t1) and sets $v0 to 1; otherwise it sets $v0 to 0 and has no side effects on the memory (it changes nothing in the memory). (Note: Codes related to this question are not expected to run in SPIM, just reason about their execution on paper.)

   a) Using the "cas" pseudoinstruction, implement two functions Acquire and Release in the MIPS assembly language.

   b) Suppose that the following MIPS codes labeled with T1 and T2 are executed by two distinct threads T1 and T2. Suppose a single processor and it is possible that a thread is preempted after execution of any instruction. When a thread is preempted, all the registers are saved and they are restored when it is restarted later. What are the possible values in the memory word labeled "x" at the end, after both T1 and T2 terminate? Give the execution scenarios to justify your answer.

```
. data
x   .word  0

.text
T0:                              T1:
la  $t0,  x                      la  $t0,  x
lw  $t1,  0($t0)                 lw  $t1,  0($t0)
addi  $t1, $t1, 1                addi  $t1, $t1, 1
sw  $t1,  0($t0)                 sw  $t1,  0($t0)
# assume thread ends here        # assume thread ends here
```

   c) Insert calls to Acquire and Release you implemented in part a) at appropriate locations in the above code so that when both T1 and T2 terminate the memory word labeled "x" can contain only "2" at the end.