CS 414 Fall 2004: Homework 2 (due Sept. 23)

Please submit solutions using the CMS system. Grading for each question will be on a simple "check-/check/check+" basis (later this will be translated into points: 0 if you don't do the problem, 3 for check-, 4 for check, 5 for check+). You can discuss problems with other people, but your actual solution must be your own work, not joint work. All of these questions have short answers, a sentence or two at most. If you think a question is asking for a half a page… you probably don't understand the problem!

1. Suppose that you are writing software for the Cornell Autonomous Underwater Vehicle (a self-directed submarine). You have responsibility for a program that will run on a single-processor CPU. The architecture of the system you are developing can be understood in terms of a set of sensors and a set of actuators. The sensors detect conditions in the environment around the UAV, and when "events" occur trigger interrupts. The software computes the desired course for the UAV, then prepares a schedule of actions that need to be taken to follow that course (e.g. changing motor speed, adjusting the angles for the vanes that control the UAV's orientation and bearing, adjusting the fill level in buoyancy tanks, etc). As time elapses, the program then takes the actions indicated in the schedule.
    a. How might this system be benefit from a multi-threaded architecture?
    b. Does the software you'll write have "true parallelism" or does it only have "concurrency"?
    c. Describe a hypothetical situation in which your software might benefit from using a busy-wait.
    d. Describe a hypothetical situation in which a busy-wait might prevent the UAV from correctly implementing its schedule of actions.
    e. Is this type of single-processor architecture at risk of race conditions? Explain very briefly why, or why not.
2. Consider the "bouncing boxes" application you developed in homework 1. Recall that in homework 1, the boxes bounce off the edge of the frame but not off each other.
    a. If a thread is adjusting the variables inside a HasAThread object and its associated button, would that code be part of a critical section?
    b. Does the problem change when boxes bounce off each other, too? Why or why not?

3. In class, we saw that an atomic **`test_and_set`** operation can be used to implement critical sections on a machine supporting parallelism (with multiple physical CPUs). But suppose that you were given a different atomic instruction called **`decrement.`** This instruction decrements a variable and leaves the initial value in register 0. Could **`decrement`** be used to implement critical sections? Show us how or explain why not.

4. In the Linux operating system, some system calls can "block". For example, if a process calls the read() system call on an input device like a keyboard, the process

will block until the user types a full sentence. How will a multi-threaded application behave if one of its threads calls read()?

5. Suppose you are designing a Windows gaming application. When the user presses the "Start" button, your application calls a procedure called "run_game()" that starts simulating monsters and other objects. But now you find that the application is unresponsive to other kinds of input -- you can't click on the screen, or press buttons. What can cause this sort of behavior? Explain how it can be corrected.

6. Consider the n-process Bakery Algorithm:

```
CSEnter(i):
    chosing[i] = true;
    number[i] = max(number[0], ..., number[N-1])+1;
    chosing[i] = false;

    for(j = 0; j < N; j++)
    {
        while(chosing[j]) continue;
        while(number[j] > 0 && (number[j],j) < (number[i],i)) continue;
    }

CSExit(i):
    number[i] = 0;
```

a. Suppose that process 2 is trying to enter the critical section and has already picked a number value and set chosing[2] to false.. In a "worst case scenario", how many times could process 0 "sneak in" before process 2 gets its turn? Explain.

b. Same question, but now answer for the case where process 2 hasn't yet set chosing[2] to false.