| CS 4760 | **Operating Systems** | Test 1 |
|---|---|---|
| Name: | Spring 2009 | Max Pts: 54 |

**Important**: This is an open book test. You can use any books, notes, or paper, but not exchange anything with other students. You are not allowed to use any electronic/communication devices, including a calculator. *Do not log into the computer during the test. Switch off your cell phones.* Any calculations and rough work can be done on the back side of the test pages. You will lose five points for not writing your name.

1. [6 pt] The current trend in CPU development is towards multicore chips. From an OS point of view, will they be classified as being suitable for SIMD, MIMD, or SMP? Can they be used for all three types? Explain your answer.

2. [6 pt] Name any two advantages of cloud computing. Name any two disadvantages as well.

3. [6 pt] How is the execution context of a process used by the OS?

4. [6 pt] Why do we need two modes (user and kernel) for process execution in an OS?

5. [10 pt] Consider the following program:

```
extern bool   lock[2] = { false, false };  // In shared memory
extern int    turn = 0;                    // In shared memory

void process ( int me )     // me can be 0 or 1; process id
{
    int other = 1 - me;
    while ( 1 )
    {
        lock[me] = true;
        while ( turn != me )
        {
            while ( lock[other] );
            turn = me;
        }

        critical_section();

        lock[me] = false;

        remainder_section();
    }
}
```

This is proposed as a software solution for two concurrent processes. Explain whether it is a valid solution or not?

6. [10 pt] Rewrite the following using `fork`, `join`, and `quit` primitives. Make sure that it exploits maximum parallelism but *always* produces the same result as the sequential execution.

```
W = X1 * X2;
V = X3 * X4;
Y = V * X5;
Z = V * X6;
Y *= W;
Z *= Y;
ANS = Y + Z;
```

7. [10 pt] Assume that each of the five philosophers, $i$, in the dining philosophers problem executes the following segment of code:

```
mutex.P();
fork[i].P();
fork[(i+1) % 5].P();
mutex.V();
    eat();
fork[i].V();
fork[(i+1) % 5].V();
```

Does this code satisfy all requirements of the dining philosophers problem?