CSL373 Minor 1

Answer all 7 questions. 04/02/2013 Max. Marks: 15

OS Abstractions

1. Consider the C function 'printf()' on UNIX. Is printf() implemented by the OS, or by an application-level library? What system call does printf() make internally? [0.5]

2. Consider the following code:

```
for (i = 0; i < 4; i++) {   fork();   }
```

If we start with one process, what is total number of processes spawned by this loop (excluding the first process). Explain. [2]

3. How does the shell implement “&”, backgrounding? e.g., $ “./compute &” [1.5]

4. Because threads can access shared state concurrently, a bad thread interleaving could potentially result in incorrect program behavior (if the program is not written carefully). Such a situation is called a race condition. Assume that you are developing a new OS, let's call this YOS (your-own OS). Assume that YOS runs only on uniprocessor machines.
    a. Is it possible for a multi-threaded application to have a race-condition when running on YOS (on a uniprocessor system)? Why/why not? Clearly state the assumptions you are making regarding your OS design to justify your answer. [1]

b. One way of disallowing race conditions is to use “locks”. Your lab partner suggests that a simple way of implementing locks in YOS is to implement two system calls called “disable_interrupts()” and “enable_interrupts()”. He suggests that because YOS runs only on uniprocessor systems, an application programmer can use these system calls to ensure atomicity of its critical sections. If you agree with him, implement functions “lock(L)” and “unlock(L)” using the system calls “disable_interrupts()” and “enable_interrupts()”. [1.5]

c. Is it a good/bad idea to provide such system calls (enable/disable interrupts) to the application developer? Why/why not? [1.5]

Function Calls

5. Consider the following function:

```
int foo(int a,int b) {
    int c;
    c = a * b;
    int d;
    d = a + b;
    return c + d;
}
```

Assume all optimizations are disabled and the assembly code for a C statement (including declarations) is generated in the same order as the original source code.

a. Write the assembly pseudo-code generated for the function `foo()' on 32-bit x86. We will not focus on the syntax of the assembly written by you, we are only looking to test your understanding of how a compiler generates code for a function. [3]

b. Assume that the compiler decides not to use a frame pointer register (ebp on x86). Is it still possible to generate correct code for function foo()? Which assembly instructions will you change in your previous answer, and to what? [2]

Virtual Memory

6. List the primary advantages of segmentation over paging. [1]

7. List the primary advantages of paging over segmentation.[1]