CS 4760	**Operating Systems**	Test 1
Name:	Fall 2011	Max Pts: 40

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] Unix provides a set of library functions that correspond to a system call. For example, the library function `fopen(3C)` is used to open a file, and so is the system call `open(2)`. What is the point of creating two separate functions for the same operation?

2. [6 pt] Consider a machine with two levels of memory (cache and main). The access time for cache memory is 50ns while the main memory is accessed in 1μs. Let us say that the cache hit ratio is 0.9. What is the average memory access time for this machine?

3. [6 pt] Memory management has two conflicting goals: process isolation and allowing processes to cooperate through shared memory. How does an OS reconcile between the two?

4. [6 pt] What is the difference between the trace of a sequential process and a multi-threaded concurrent process?

5. [6 pt] Bakery algorithm allows two processes to get the same turn number. How does it resolve the conflict when two processes get the same turn number? Is the resolution guaranteed to work? Explain your answer.

6. [10 pt] Consider the following implementation of producer-consumer processes. The `P` and `V` functions implement the standard operations for semaphore wait and signal.

```
// The extern variables are allocated in shared memory.

extern semaphore mutex;               // To get exclusive access to buffers
extern semaphore empty ( n );         // Number of available buffers
extern semaphore full ( 0 );          // Initialized to 0

void producer()
{
    do
    {
        produce ( item );
        mutex.P();
        empty.P();
        put ( item );
        mutex.V()
        full.V()
    } while ( 1 );
}

void consumer()
{
    do
    {
        full.P();
        mutex.P();
        remove ( item );
        mutex.V();
        empty.V();
        consume ( item );
    } while ( 1 );
}
```

Use the critical section solution protocol to determine if the solution is valid.