Examination 2 Operating Systems, Computer Science 66421
Spring 1999 (April 28, 1999)

1 Rules and Hints for the Exam

This examination is open book and open note (your notes only please). Calculators are permitted. Networked devices are strictly prohibited. The questions are marked as to their relative value, the exam will be scored out of 100% but is worth 25 points towards your course grade. Relax and try to do what you can.

Here are some hints for doing well:

* You have about 100 minutes, so using one minute per percentage point of work is a good rate.
* You can do the problems out of order, just be sure that the grader can find the order.
* Please be neat (illegible answers cannot get credit).
* Please put in the upper right hand corner of each page your name, the page number and the total number of pages your grader should expect (in case the pages become detached). So for example if your name was James Taylor, then you would write: James Taylor 1 of 5.
* Don’t forget to staple your pages together at the end.

2 The Problem Set

1. Deadlock (10 %) — Consider a system with 2 processes, $P_1$ and $P_2$, and 2 units of a particular resource, $R_1$ and $R_2$. Requests and grants are for a single unit of resource, although each process may require 2 units of resource during its execution. Draw a state diagram where each state describes the amount of resource granted and the amount of resource requested by each process. Mark deadlocked states by drawing them using 2 circles (much like accepting states in finite automata). (Hint: draw the system for a single process first and then try it with 2).

2. Networking Systems Programming (15 %) —
   
   (a) Network Systems Programming (5 %) — In BSD sockets, what does the listen system call do and what is the significance of the second parameter.
   
   (b) Data Communication Technology and Performance (10 %) — Traditional kernel technology for input and output (I/O) and data communication relied on buffering of data in the kernel prior to transferring the data into the address space of the user process. In contrast, zero copy systems calls and libraries are frequently used for I/O devices and in data communications devices, and transfer data directly from the device into the user’s data space. Compare and contrast the two approaches, under which circumstances is one better or worse than the other.
3. Threads and Synchronization (30%) —

(a) Recall that Java uses the key word synchronized to guard a region of code (possibly a whole method or a smaller subregion). Lazy creation of objects is a way of deferring allocation and initialization until an object is actually needed (and can help avoid wasting systems resources). In a single threaded program, the logic might look something like this:

class SomeClass {
    private Resource resource = null;

    public Resource getResource() {
        if (resource == null) {  // Already allocated?
            resource = new Resource();  // does expensive allocation here
        }
        return resource;        // Now we have the resource
    }
}

However, suppose that you have a multi-threaded program AND you can turn off optimizations that reorder the generated instructions. If you looked in a text book and saw code that looked like:

class SomeClass {
    private Resource resource = null;  // reference to shared resource

    public Resource getResource() {
        if (resource == null) {  // already allocated?
            synchronized {
                resource = new Resource();
            }
        }
        return resource;
    }
}

would this work? Why or why not? If not, can you provide a fix that does not involve making the entire getResource method synchronous (15%)?

(b) Using C like pseudo-code, and an atomic hardware test-and-set operator design a semaphore based synchronization routines for both sem_wait and sem_signal (15%).

4. Networking Systems Performance (30%) — Consider a 1000 \times 10^6 \text{bits/sec} Ethernet (Gigabit). Suppose that data travels at about 2 \times 10^5 meters per second over a copper wire.

(a) Suppose that you have a 100 meter cable, how many bits can be in transit on that cable simultaneously? (10%).

(b) Suppose that the network requires a 100 byte gap between packets to support a distributed file system.

   i. If the file system used traditional IP packets with 18 byte headers and up to 1500 bytes of payload data, which contain UDP packets with 8 byte headers. What is the maximum data transfer rate over the wire (assuming that the probability of packet loss, corruption and reordering is negligible). (10%)

   ii. Suppose that IP’s Jumbo Frames option is used, allowing for 9000 bytes of payload in an IP packet. What is the maximum rate of data transfer over the cable. (10%)
5. Naming and Persistence and Systems programming (15 %): A Bloom Filter is a heuristic algorithm for
detecting membership in a set that may (with low probability) give false positives (that is a membership
test falsely indicates that a particular element is a member of the set when in fact it is not actually a
member of the set). Bloom filters are frequently used in information retrieval and in distributed file
systems (to identify which machine in a distributed system a file resides on). A Bloom Filter models
a set using an array of \( N \) bit-mapped sets with \( K \) bits (we will call this the filter). The insertion and
membership test operators employ a unique hash function for each bitmap. To insert an element into
the set represented by the filter, for each bitmap, the bitmap’s unique hash function is applied to the
element and the bit selected is turned on. To test for membership, for each bitmap, the same unique
hash function is evaluated and the corresponding bit is checked in the bitmap, with a report of the
item being in the filter if the corresponding bit is on in each bitmap. Write efficient C like pseudo-code
to implement a Bloom filter:

```c
#define K .... /* Number of bits in a bitmap */
#define N .... /* Number of bitmaps in the Bloom Filter */
/* assume K is a multiple of the number of bits in a long integer */
typedef unsigned long BitMap[K/(8 * sizeof(unsigned long))];
typedef BitMap Filter[N]; /* the Filter itself */

/* inserts the Filename into the set of files represented by the Filter */
void BF_insert(const char *FileName, const Filter *bf_ptr);

/* tests to see if the Filename is represented by the set of files
 in the Filter */
bool BF_ismember(const char *FileName, const Filter *bf_ptr);

You can assume that the following functions are already written:

/* Computes the hash function selected by WhichFunction on a given
 FileName returning a value in that is between 0 and NumValues - 1 */
int BF_hash(const char *FileName, int WhichFunction, int NumValues);

Give the computational complexity of your BF_insert and your BF_ismember operations.
```