Operating Systems (CS422)

Spring 2002

Yale University

May 7, 2002

Name: ______________________________ **Netid:** _______________

- Write your answers directly on the examination paper, including any work that you wish to be considered for partial credit. Use the back side of each sheet if you require more space to answer a question.
- Each question is marked with the number of points assigned to that problem. The total number of points is 30.
- The duration of this exam is two hours.
- The examination is open-notes and open-book.

Problem	Topic	Points
1	File Systems	
2	Security	
3	Concurrent Queues	
4	Dynamic Linking	
5	Networking	
6	Concurrent Heaps	
TOTAL		

Problem 1 *File Systems [5 points]*

A. Consider the traditional Unix file system (as implemented in BSD). Consider the following code.

```
FILE *fd = fopen("/log", "a");  // opens the file in append mode
                                // new writes go to the end of the file
char buf[4096];

fwrite(buf, 4096, 1, fd);       // writes 4K to end of the file

fclose(fd);
```

How many disk reads and disk writes are issued when you execute the above code? Let the size of file "/log" be 8K to begin with. Assume that block size is 4K. Assume that the buffer cache for the file system is empty and that all writes are forced all the way to the disk. State any other assumptions that you might make.

B. Flash memory is a storage medium used in embedded devices. It is a form of Electrically Erasable Read Only Memory (EEPROM) and is both programmable (data stored on the device can be changed) and persistent (data stored on the device survives reboots). Data is organized as blocks and writes are performed in block granularity. The lifetime of a flash chip is limited. It is measured in erase cycles (rewrites of a block), with the typical lifetime being 100,000 erases per block. Its primary advantage is very low power consumption.

Flash memory is increasingly being used in lieu of disks in PDAs/handhelds. In fact, the OS can access the device as if it is a traditional hard disk and can create a file system on the device. What problems would one face if one were to execute Unix file system code (such as BSD FFS) on top of a flash card? How would one modify the file system to overcome these difficulties?

Problem 2 *Security [5 points]*

Consider the following two variations of the Needham-Schroeder secret key protocol. Do they have any security flaws? If not, are they equivalent to the original protocol? Or are they superior to the original protocol?
Let A be a client that wants to communicate with B, and let S be the server. K_a is A's secret key, and K_b is B's secret key. K_{ab} is generated by the server to serve as a session key. Let x and y be nonces. Notation: *text*^{k} means that *text* is encrypted using key k.

Version 1:

A sends to S: $(A\ B\ x)$
S sends to A: $(x\ K_{ab}\ (A\ K_{ab})^{K_b})^{K_a}$
A sends to B: $(A\ K_{ab})^{K_b}$
B sends to A: $(y)^{K_{ab}}$
A sends to B: $(2 * y)^{K_{ab}}$

Version 2:

A sends to S: $(A\ B\ x)$
S sends to A: $(x\ B\ K_{ab})^{K_a}$
S sends to B: $(x\ A\ K_{ab})^{K_b}$
B sends to A: $(x + 1)^{K_{ab}}$
A sends to B: $(x + 2)^{K_{ab}}$

Problem 3 *Concurrent Queues [4 points]*

For the following implementation of atomic transfer, say whether it (i) works, (ii) doesn't work, or (iii) is dangerous — that is it sometimes works and sometimes does not. If the implementation does not work or is dangerous, explain why and show how to fix it so it does work.

```
void AtomicTransfer (Queue *queue1, *queue2) {
   Item thing;    / * thing being transferred */

   queue1->lock.Acquire();
   thing = queue1->Dequeue();
   if (thing != NULL) {
      queue2->lock.Acquire();
      queue2->Enqueue(thing);
      queue2->lock.Release();
   }
   queue1->lock.Release();
}
```

The atomic-transfer routine is supposed to dequeue an item from one queue and enqueue it on another. The transfer must appear to occur atomically: there should be no interval of time during which an external thread can determine that an item has been removed from one queue but not yet placed on another. In addition, the implementation must be highly concurrent — it must allow multiple transfers between unrelated queues to happen in parallel. You may assume that queue1 and queue2 never refer to the same queue.

Problem 4 *Dynamic Linking [5 points]*

Assume that you are using the Multics VM system (or a system similar to it) and that you are executing a process instantiated from the following program. (The process is created by invoking Prog:main.)

```
[Module: Prog.c]
void main()
{
    int x = Math.pi;         // access the pi variable from the math module
    int y = Math.sin(x);     // invoke the sin function
    Helper.help();           // invoke the help function located in Helper module
}
```

The Helper module looks like:

```
[Module: Helper.c]
void help()
{
    int z = Math.cos(0);     // invoke the cos function
    /* Program point p */
    if (z > 0.5)
        Xwin.display(z);     // invoke a display method defined in Xwin module
}
```

The Math module looks like:

```
[Module: Math.c]
double pi = 3.14;
double sin(int ignore)  { return 0.0; }
double cos(int ignore)  { return 1.0; }
double tan(int ignore)  { return 0.0; }
```

Assume that the program is in the middle of executing the help() method and is currently at program point p. Answer the following questions regarding the virtual memory image of this process.

- How many segments are currently associated with the process?
- How many linkage sections have been instantiated for the process?
- How many entries are contained in each one of the linkage sections?
- Describe the contents of each linkage section. Be specific. State any assumptions that you need to make to answer the question.

Problem 5 *Networking [5 points]*

You are developing a network protocol for the reliable delivery of fixed-sized messages over unreliable networks. You are using a sequence number in each message to allow the receiver to eliminate duplicates, but you still have three design alternatives to consider.

The design alternatives are: a) sender must receiver an acknowledgement for the previously sent message before it can send the next message in the sequence, b) sender can transmit up to n unacknowledged messages, but the receiver will discard any messages that are received out of sequence, and c) the sender can transmit up to n unacknowledged messages, and the receiver will acknowledge each on receipt, even if they arrive out of order.

- What state must the receiver keep around for the different alternatives?
- Order the alternatives in terms of performance if: a) the sender and receiver are connected through a direct network link, and b) the sender and the receiver are separated by multiple network links with a router at each hop.

Problem 6 *Concurrent Heaps [6 points]*

Consider the following array-based implementation of the "heap" data structure, which is often used in priority queue implementations. The heap has the property that the value stored at a node is less than the values stored at its two children. As you might recall from previous encounters with the array-based heap, the root is stored at index 0, and the children of a node stored at index p are located at $2 * p + 1$ and $2 * p + 2$. An insert operation creates the new element as the rightmost leaf (which is at the end of the array) and propagates the value to the top by doing a sequence of pairwise swaps between the value and its parent. Make the following code thread-safe so that multiple inserts can be performed concurrently. Maximize the amount of concurrency. A solution that holds a single lock for the duration of an entire insert operation is not an acceptable solution.

```
int heapElems[MAX];
int numElems = 0;

void Insert(int val)
{
    assert(numElems < MAX);
    int pos = numElems++;
    heapElems[pos] = val;
    while (pos != 0) {
        int parent = (pos-1)/2;
        if (heapElems[parent] < val)
            break;
        heapElems[pos] = heapElems[parent];
        heapElems[parent] = val;
        pos = parent;
    }
}
```