Department of Computer Science
Operating Systems Qualifying Exam
August 14, 2008

The following exam is open book and open notes. You may feel free to use whatever additional reference material you wish, but **no electronic aids** are allowed. Please note the following:

- show your work whenever appropriate. There can be no partial credit unless you show how you derived your answers
- be succinct. You may lose points for facts that, while true, are not relevant to the question at hand

1. (25 points) Some computer system uses a hashed page table. For the sake of making this question tractable, assume that a virtual address is a ridiculously short eight bits, divided into a four bit VPN and a four bit offset. The physical address is also ridiculously short: it's ten bits. Also, assume the hash function is just the VPN modulo 8. An in-table chain of Next pointers is used to resolve hash collisions. Here's the current state of the HPT:

Entry	VPN	PFN	Next
000			
001	0001	101010	101
010			
011	1011	111100	111
100	1100	010101	
101	1001	001100	110
110	0101	110011	
111	1111	000000	

   In the table, an entry with an empty VPN is empty (and does not supply a VM mapping). The end of a Next chain is denoted by an empty Next field.

   For each of the following virtual addresses, either give the physical address it's mapped to or identify it as a page fault.

   (a) `00010110`

   (b) `10010101`

   (c) `01011100`

   (d) `00000011`

   (e) `00111010`

2. (25 points) When scheduling processes on a multiprocessor computer:

   (a) What considerations might lead you to migrate a process from one CPU to another?

   (b) What might lead you to decide not to migrate it?

3. (25 points) Assume the following code is to be executed on three nodes P1, P2, and P3 of a multiprocessor computer.

P1	P2	P3
`x = 1;`	`y = x+1;`	`y = y+2;`

   (a) Using the standard memory operation notation, describe the memory operations that occur in this code (for convenience, the memory transaction notation is attached to this exam). For purposes of this part of the question, assume P1's statement is executed first, then P2's, and finally P3's.

   (b) Assuming

      i. the computer uses the LimitLESS (*i.e.* Alewife) protocol,

      ii. that all the variables are homed on P1, and

      iii. that the processor sets of all the variables is initially empty (so the processor caches are all empty)

   give all the messages that need to be passed among the processors when this code is executed using the transactions you found in part (a). Note: be sure to give all relevant information for each message: the sending processor, the receiving processor, the message type, etc.

   (c) Find a different possible set of memory operations for this code which will result in `y` having a final value of 1.

4. (25 points) Suppose you're using a sorted binary tree to maintain information. The code to insert a node in the tree is as follows:

```
typedef struct _node *Node;
struct _node {
    Semaphore sem;
    int key;
    Node left, right;
};
void insert(Node treenode, Node newnode)
{
    if (newnode->key < treenode->key)
        if (treenode->left == NULL)
            treenode->left = newnode;
        else
            insert(treenode->left, newnode);
    else
        if (treenode->right == NULL)
            treenode->right = newnode;
        else
            insert(treenode->right, newnode);
}
```

   (this code is deliberately simplified: it assumes that a new `Node` to be inserted will have its `left` and `right` pointers set to `NULL`, and doesn't consider the case of an empty tree)

   (a) The given code is correct in a single-process environment. However, show that if an attempt is made to insert two nodes with the same key in a multiprocess shared-memory environment, it's possible for it to only be inserted once (note that in the uniprocessor case this would result in both nodes being inserted, which is regarded as the correct behavior).

   (b) The `Semaphore` type is a binary semaphore; its operations are named `lock(s)` and `unlock(s)`.

   Insert `lock()` and `unlock()` statements in the `insert()` function so it will work correctly, while holding semaphores for as short a time as possible.

A Memory Operation Notation

This appendix provides a brief review of the memory operation notation being used in this exam.

A.1 Operations

In this notation, a memory operation is represented with the characters `op(var)val` where

- *op* is the operation: `R` is used for a read operation, and `W` is used for a write operation.
- *var* is the variable to be read or written.
- *val* is the value that is read or written.

So, using this notation, `W(x)1` means "write the value 1 to the variable `x`", and `R(y)2` means "read the value of `y` from memory. The value memory returned is 2.

A.2 Sequences of Operations

A sequence of operations performed by a single processor is written horizontally on a line:

```
R(x)0  W(x)1
```

states that a processor first reads the value of `x`, and obtains a 0. It then writes `x` to memory again; the value it writes is a 1.

An example of a line of C code which might result in this sequence of memory operations is

```
x = x + 1;
```

As we can see, the addition performed by the processor has been abstracted away: all that is shown in the memory operation notation is the actual memory operations performed.

A.3 Multiprocessor Memory Operations

Finally, if we have several processors performing memory operations, we represent each processor's operations on a line, like this:

P1	`R(x)0  W(x)1`
P2	`R(x)0  W(x)2`

In this example, P1 and P2 simultaneously read `x`, and both obtain a value of 0. In the second time unit P1 writes a new value to `x` (a 1), and on the third time unit P2 writes a new value to `x` (a 2). The final value of `x` in this example is 2.

It's important to keep in mind that there is nothing saying that this is the only possible sequence of operations – from the information provided, P2 could equally well have written its result first, or it might not have even read `x` until after P1 had written it. The notation merely specifies what did happen, it *doesn't* say there's any reason it needed to happen that way.

A.4 Extensions

There are a variety of extensions to this notation, to either insert synchronization or to explicitly represent communication. Those are beyond the scope of what we need for this!