

**CS 574**  
**Final Exam**  
**December 10, 2007**

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 instructions. There will be a ten point deduction for failure to comply with them:

- start each problem on a new sheet of paper. It's likely that the easiest way to turn in Question 7 is to fill it in on the exam paper and turn that in.
- write your Banner ID number, but not your name, on each sheet of paper you turn in.

Also, 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.

You have until 12:30 to finish the exam. The questions are equally weighted.

1. Both Minix and Xen attempt to limit the damage that can be done by a device driver containing a bug.

(a) What is the mechanism used by each to limit the damage?

*Minix: spin off device drivers into userland; use kernel to mediate low-level IO port reads and writes.*

*Xen: provides its own device abstractions to guest OSes (hmm, there's room for some misunderstanding here: I meant guest OS device drivers; a student might read the question as referring to Xen's device drivers, though of course there is no limit to the damage a Xen bug can cause. If I see an answer like this, I'll have to give it credit...), guest OS can only ask Xen to do IO for it – can't do it directly.*

(b) Suppose the mouse driver has a bug. How much damage can it cause (e.g. is there a set of processes you can identify that would crash, could it bring down the whole system, etc), for each?

*Minix: at worst, any applications using the mouse might crash (if they're carefully written, they'll even survive since they can observe an IO failure and retry).*

*Xen: it can bring down a whole domain.*

*Neither one could bring down the whole system, though.*

5 points for each subpart

<i>Points</i>	<i>Error</i>
-2	<i>Minix: don't mention kernel mediation of IO access</i>
-3	<i>Xen: don't mention low-level drivers providing device abstraction to domain</i>
-2	<i>Minix: don't mention impact on processes using mouse driver</i>
-3	<i>Xen: don't mention you can bring down a domain</i>

2. CPU schedulers for operating systems tend to use very complex algorithms, with lots of heuristics to try to enhance response to interactive inputs. CPU schedulers for virtual machines tend to be very simple, perhaps even to the point of simply alternating time slices between the virtual machines being hosted. Why are these very different choices both appropriate?

*The important difference here is in the granularity of the object being scheduled: you will typically have many processes, but comparatively few VMs. Consequently, simple time-slicing between VMs makes them all just look like slower machines, while the OS running on each VM uses as complicated a scheduling algorithm as one running on bare hardware.*

3. Let's consider a few of the differences between D-Bus and RPC.

(a) How is D-Bus's execution model better suited to parallel execution than RPC's?

*The most pertinent difference here is that D-Bus uses an asynchronous message-passing model, while RPC uses a synchronous procedure call model.*

(b) How is RPC better suited to a heterogeneous network environment than D-Bus?

*D-Bus is explicit about using the sender's native byte-ordering; normally, RPC uses a host-independent representation (though it turns out that the Birrell paper doesn't specify whether Cedar RPC does so...). I'll be grading this part very generously...*

(c) How does D-Bus differ from RPC in locating a service to communicate with?

*RPC uses a well-defined database. D-Bus's actual technique is buried down on the details where they don't tell us about it. It appears to be what amounts to string-matching in the D-Bus server.*

*Three part question that needs to add up to 20. I'll give seven points each for the first two parts, and six points for the third.*

4. One of the differences between Alewife and DASH is that when a cache line is dirty and a new processor reads it, the cache line ends up in different states in the two systems.

(a) What states does it end up in, for each of them? And which processors have valid copies? Assume the processor that previously had it is named processor  $P_{old}$  and the new processor is named processor  $P_{new}$ .

*LimitLESS:  $P_{old}$  loses its copy, and  $P_{new}$  gets it in state read-only.*

*DASH: both  $P_{old}$  and  $P_{new}$  get the data in state shared-remote.*

(b) What is the expected program behavior (i.e., what do you think the programs running on the processors are going to do in the near future) that makes the DASH solution look like a good one? What is the expected behavior that makes the Alewife solution look like a good one?

*DASH: the programs running on both processors will continue to read the data. They've both got it in a readable state, so they can just continue.*

*LimitLESS: the program that just read the data will change it in the near future; at any rate, the old program doesn't intend to look at it soon. Changing to read-write state will require communication with the home node, but won't require communication with  $P_{old}$ .*

5. Suppose you wanted to add process migration to a system that had distributed shared memory. What extra difficulties do you anticipate this might cause the memory consistency algorithm?

*The big thing is that all the DSM algorithms maintain a directory showing what nodes contain what memory. If the process moves out from the node, that information is all wrong now. We might try to patch things up using something like the socket forwarding MOSIX uses; the easiest fix might be to flush everything from our cache before moving...*

*Another potential solution would be to just leave the data in whatever state it was in, and have all of cache be invalid when the process reaches the new machine. That would cause a huge number of messages as memory got migrated as it was touched (much worse than in Sprite, since with Sprite at least whole pages get migrated at once).*

6. Compute the CRC of the message string 100111101101

using the polynomial  $1 + X + X^3$

The input string is presented using the modern convention of most significant bit first, which is backwards from that used in the paper.

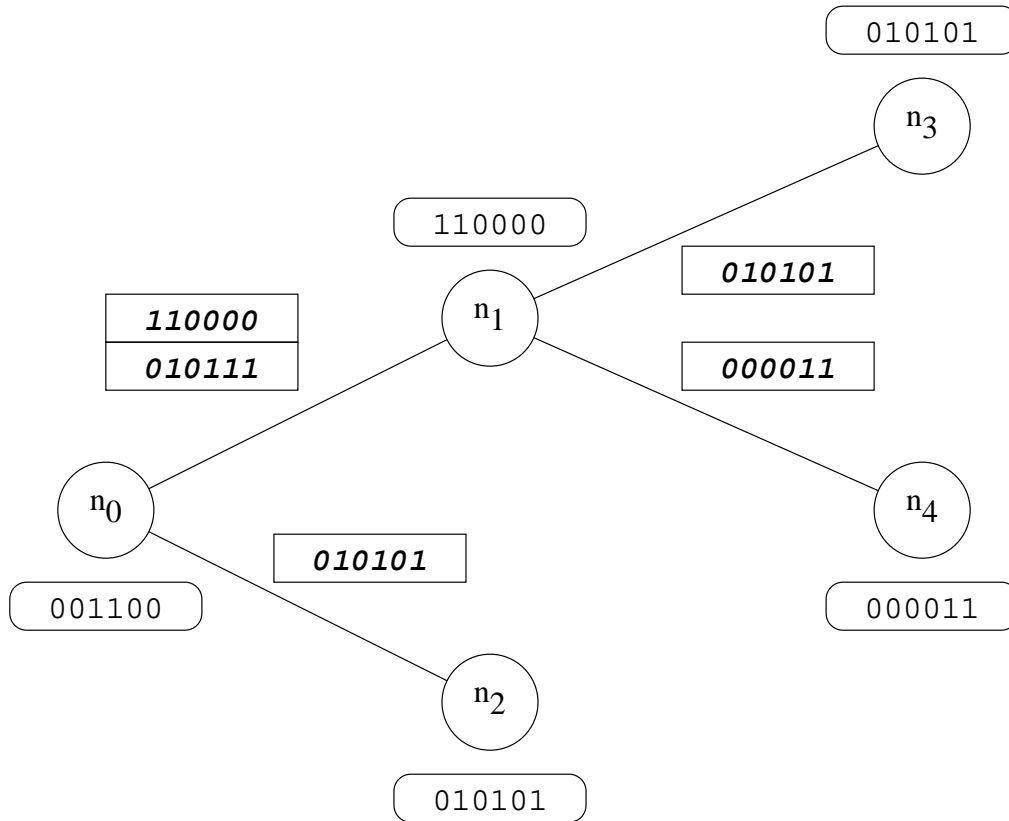
*This is, of course, the first problem from the homework I didn't assign. It seemed reasonable that people who did it ought to get some benefit!*

```
100111101101000
 1011
001011101101000
 1011
0000101101000
 1011
000001000
 1011
 011
```

As usual, I tried to guess whether the errors I saw were conceptual or arithmetic. So, not adding any 0's before starting the division cost points, reversing bits didn't cost points, adding a fewer-than-three-bit CRC at the end cost points.

7. The following figure shows a portion of an OceanStore filesystem, using the same conventions as in Figure 2 of the OceanStore paper (the fact that two of the rounded boxes contain the same mask is deliberate).

(a) Fill in the rectangular boxes.



-2 for a box with a wrong mask

(b) Describe how a query for a file that hashes to 010101 is routed through the network.

*Of course, there are two nodes with the file:  $n_2$  and  $n_3$ . The system starts at  $n_0$ , which does not have file. Examining the links, it finds that the file might be at a distance of 2 along the link to  $n_1$  or at a distance of 1 along the link to  $n_2$ . So it follows the shorter path, and finds the file at  $n_2$ .*

-5 for going up to  $n_1$  and not  $n_2$

-2 for going to  $n_1$  as well as  $n_2$