

CS 574
Final Exam
December 5, 2005

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
- write your social security number, but not your name, on each sheet of paper you turn in
- show your work whenever appropriate. There can be no partial credit unless I see how answers were arrived
- be succinct. You may lose points for facts that, while true, are not relevant to the question at hand

You have until 10:00 to finish the exam.

1. In a computer system using Alewife's LimitLESS protocol, support variable x is homed (*ie* has its directory) on processor A, and variable y is homed on processor B. There are two valid copies of x , on processors C and D, and one valid copy of y , on processor C.

Now suppose processor D executes the following line of code:

$x = y;$

List all the messages that need to pass between the processors to accomplish this. Be sure to give the source processor, destination processor, message type, and name of the variable for each message.

First, D needs to get a read-only copy of y. Looking at my question again, it's ambiguous as to whether the single valid copy of y is a read-only copy and the memory is also valid, or if it's a read-write copy and there is no valid copy in the memory itself (and with my emphasis during the exam on "home" meaning where the directory is, I may have implied the latter). With that in mind, I'll take either (memory has a valid copy):

D -> RREQ_y-> B
B -> RDATA_y-> D

or (memory does not have a valid copy):

D -> RREQ_y-> B
B -> INV_y-> C
C -> UPDATE_y-> B
B -> RDATA_y-> D

Now, D needs to get read-write access to its copy of x.

D -> WREQ_x-> A
A -> INV_x-> C
C -> ACKC_x-> A
A -> WDATA_x-> D

| Points | Description |
|--------|---|
| -2 | <i>D ends by sending A and update for x, which A passes on to C</i> |
| -8 | <i>Confused messages (P={A, C} for x), ends with P={}</i> |

2. One computer wants to send the message 10110101 to another computer using a cyclic redundancy check to assure correctness. Give the resulting message (including both the original message and the frame check sequence) if a CRC generating polynomial of $x^3 + x + 1$ is used.

The polynomial translates to 1011. Dividing, we get

$$\begin{array}{r}
 1011) 10110101000 \\
 \underline{-1011} \\
 01010 \\
 \underline{-1011} \\
 100
 \end{array}$$

So the resulting message is **10110101100**

| Points | Description |
|--------|--|
| -2 | Extra 0 added to message before division; results in remainder of 0011 |
| -3 | Five bit remainder? |
| -3 | Didn't construct message at end |

3. It is much easier to share memory regions between processes in a computer using conventional page tables than one using either inverted or hashed page tables.

(a) Why?

Conventional page tables are lookup tables from virtual addresses to physical. Sharing memory is just a matter of having the same contents in two or more table entries (in different processes). With inverted and hashed page tables, the table maps physical addresses to virtual; there isn't a way to have more than one virtual page number in the entry.

(b) When a computer uses inverted or hashed page tables, there has to be some scheme for resolving hash collisions. Can something similar to a hash collision resolution scheme be used to make it easier to do shared memory?

You could have a linked list of PID/VPN pairs coming from each table mapping.

There was no real way to assign points on the basis of "so many points for such and such an error." In general, a wrong answer that had something to do with the question got half credit. People who did think like tell me about how page tables are set up during the Linux boot process (!) got less.

4. Would it be possible to implement an NFS server that used Google FS internally?

So far as we can determine from the information in the relevant papers, yes: NFS provides a series of RPC calls mapping NFS requests to the server's native file system. Since the google file system supports directory walking, reads, writes (and in general, normal file system operations) it could easily be a backend for NFS.

5. How are block devices different from character devices in ways that (1) make it possible for Linux to use a different structure for block devices than for character devices, and (2) make it a good idea. I'm specifically thinking of the separation between the interaction with user processes from the strategy used to schedule reads and writes in block devices.

The purpose of a character device is more to provide real IO connecting to the outside world; the purpose of a block device is much more along the lines of providing a mechanism for data storage. Consequently, there is no harm in performing reads and writes out of order. Since many block devices (especially disks) don't perform random access, reordering reads and writes can provide much higher performance.

6. In a uniprocessor system, if one process sends a signal to another one, it isn't necessary to make sure the signal is delivered to the receiving process immediately; instead, some signal-related data structures are modified and the sending process is able to resume immediately. Why is this sufficient to ensure the receiving process will receive the signal (assuming it isn't blocked or ignored) before it performs any additional work?

At the time the data structures are modified, the CPU is running in the kernel. The next time the receiving process is scheduled for execution, the pending signals are checked. So, the signal will be delivered before the process ever does anything else.

7. The addresses of Intel page tables, as specified by CR3 (the PDBR) and by the Physical Frame Numbers in the tables themselves, as physical (as opposed to virtual). Since Linux runs with virtual memory enabled, how is it possible for the kernel to manipulate the contents of these tables?

On a small-memory (less than about 1MB) system, all of physical memory is mapped into the upper 1 MB of virtual space. On these systems, it's possible to access any physical address a by just accessing the virtual address $3MB+a$.

Things are a little tougher with large-memory systems; on these, it may be necessary to create a temporary mapping from inside the kernel's virtual space to a page in high memory and then modify the virtual space inside that mapping.