

CS 574
Final Exam
December 10, 2003

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

1. Suppose we wish to have a system supporting both distributed shared memory and process migration

(a) How could we use DSM to simplify copying a process's code and data? Which "style" of virtual memory transfer (monolithic, precopying, or lazy copying, using the terms as defined in the Sprite paper) would this most closely resemble?

Just share the memory between the old and the new hosts, and start executing on the new one. Since the memory is transferred when it's accessed, it's lazy copying.

(b) Would it be most reasonable in this case to use a DSM scheme with fixed homes or one without fixed homes?

Without. Fixed homes will leave a residual dependency behind forever.

(c) How could message channels be migrated in a way analogous to the virtual memory?

Transfer the essential information about the port (port number, partner on the other end, etc) to the new host. When the new host sends a packet, there's no problem; if the partner sends a packet to the old host, just forward the packet to the new host. When the partner gets its first packet from the new host, it can quit sending packets to the old host.

2. (a) What is the strictest memory consistency model covered in class for which the following is a valid execution history? Why is it not valid in the next stricter model? Insert synchronization operations which will guarantee P3 and P4 "see" the same final values for x and y .

It's valid in weak consistency; it isn't valid in processor consistency because P4 reads x and gets two, then reads y and gets 2, and later still reads y and gets 1. This means P2's write of x has reached it before P2's write of y .

P1	w(x)1	w(y)2					S
P2	w(y)1	w(x)2					S
P3			r(x)1	r(y)1	S	r(x)2	r(y)2
P4			r(x)2	r(y)2	S	r(x)2	r(y)2

This answer isn't unique, but it is one that results in both P3 and P4 getting the same final results. Notice that it also fixes the typo in the exam (recall that this typo was also fixed on the board during the exam).

(a) What is the strictest memory consistency model for which the following is a valid execution history? Why could this history never actually happen?

P1	w(x)1		
P2	r(x)1		

It meets the definition of sequential consistency, but it violates causality

3. Using the values $p=3$ and $q=5$, find the smallest possible RSA keys (e, n) and (d, n) such that $d < e$. Use your decryption key to encrypt the message 3. *note: yes, I said “use the decryption key to encrypt” - if you got the keys I’m expecting, this problem will be easy with the decryption key but I wouldn’t want to try it by hand with the encryption key*
- (a) n is $(3-1)(5-1) = 8$
 - (b) The smallest number relatively prime to 8 is 3, so $d = 3$.
 - (c) Since the numbers we’re looking at are so small, we can find e by searching for some number such that $(d \times e)_{\text{mod } 8} = 1$. 3 would work, but the problem specified that $d < e$; the next “lucky winner” is 11. So $e=11$.
 - (d) To encrypt, $3^3 = 27$; $27_{\text{mod } 15} = 12$.

I actually only specified $d < e$ in the problem because I thought that if people got the same number as both the encryption and decryption key they’d think they had to have made a mistake, and would waste time trying to find it. I’m not going to take off any points for using $e=3$. The reason I picked d as the smaller one was that if you worked from the definitions in the paper you’d get the answer faster that way. I was stunned to find out that some people, instead of working from the definitions (which made things really easy) used the example techniques in the appendix. I don’t think I’ll take any points off for doing this if you were able to carry it to an answer; unfortunately, it will have cost a fatal amount of time.... Also, several students noticed that 1 is relatively prime to everything, so they used a decryption key of d . Not what I had in mind since a key that results in ciphertext=plaintext is pretty odd, but perfectly valid given my statement of the problem. One very common way to make the problem hard was to misread a statement from the paper, “It is very easy to chose a number d which is relatively prime to $\phi(n)$. For example, any prime number greater than $\max(p, q)$ will do” as saying d had to be a prime greater than $\max(p, q)$. This results in $d=7$, which involves bigger arithmetic than I wanted you to calculate. I was also pretty surprised that one of the students who thought I really wanted them to calculate 3^7 by hand actually got 2187 - and there was another who successfully got $3^9 = 19,683!$

4. There are four basic memory operations in release consistency: data read, data write, acquire, and release. Which of these should be considered as events in the sense of Lamport’s clock paper? Should some of them be considered as events occurring within a process, and some as events occurring between processes? Do any of them have to be combined to be considered as an event?
- Data read and data write are intra-process events. Release-acquire together are an inter-process event.*
5. Suppose you have a computer system with a 32 bit virtual address, a 32 bit physical address, and a 4K page size. Also assume the machine has a 64K direct-mapped cache (the line size isn’t relevant here), and only 64K of physical memory.

- (a) If this system uses page coloring, which address bits are “bin” bits?
*It takes 16 bits to index 64K, so the line number and byte offset together have to be 16 bits. It doesn’t matter whether there are 64K, 1 byte lines (that would be 16 bits for the line number and 0 bits for the byte offset), one 64K line (that would be 0 line number bits and 16 byte offset bits), or 16K lines of 4 bytes each (that would be 14 bits for the line number and 2 bits for the offset).
 So, we have 12 bits of byte offset in the page, and 16 bits to index the cache. That means we’ve got four bits (bits 12-15) as bin bits.*
- (b) Develop a virtual to physical mapping for the following addresses which will take advantage of page coloring: 1234a123, 17321234, 1234c333, 1234a756, 00122000, ffff0000. Be sure that in your mapping you map pages, not individual addresses.

This turns out to have a unique answer: just set the four bits in the translated address to match the bin bits in the virtual address. Since the physical memory is the same size as the cache, the leading bits are all just zeroes. So we get a mapping of:

VPN	PPN
00122	00002
1234a	0000a
1234c	0000c
17321	00001