

**CSC 469H1 F
ADVANCED OPERATING SYSTEMS**

**UNIVERSITY OF TORONTO
Fall 2006**

Term Test #2

NO AIDS ALLOWED

Please **PRINT** in answering the following requests for information:

Family Name: _____

Given Names: _____

Student Number: | _ _ _ | | _ _ _ | | _ _ _ |

Login (@cdf): _____

Notes to students:

1. This test lasts for 110 minutes and consists of 89 marks. Budget your time accordingly.
2. This test has 8 questions and 11 pages (including this one); Check that you have all pages.
3. **Write in pen. No pencils. Really, I mean it.**
4. Write your answers on this “question and answer” paper, in the spaces provided. Be concise. In general, the amount of space provided is an upper bound on the “size” of answer that is expected. If necessary, use space where available and provide explicit pointers.
5. State your assumptions and show your intermediate work, where appropriate.
6. Do **not** go beyond here until instructed to do so. Write your student number at the top of each succeeding page once you get going.

Question	Marks
1	/15
2	/10
3	/16
4	/14
5	/10
6	/12
7	/6
8	/6
Total	/89

1. [15 marks, 3 each] Definitions

Define the following terms, in the context of this course:

a) Byzantine failure

Failure model in which faulty process exhibits arbitrary behaviour. [Typically modelled as malicious attackers to capture worst case behaviour; faulty processes may collude with other failed processes but are not more powerful than non-failed ones.]

b) Clustered page table

Page table design in which (a) part of the virtual address is a hash key (as in hashed page tables) but (b) page table entries record mappings for a range of several consecutive pages. [Adv: provides fixed size page tables with low overhead (storage space and lookup time) for sparse and dense address spaces]

c) Distributed Shared Memory

Technique that allows distributed processes to transparently share a global virtual address space, although physical memory may be located on many nodes. [Typically builds on existing virtual address translation hardware, augmented with software support for remote page fetch and consistency control.]

d) RAID Level 1

Mirrored disks.

e) Vulnerability

A software flaw with a security implication.

2. [10 points] Superpages

a) [2 points] Any paged virtual memory system requires both hardware and software support. What changes or additions to hardware are required to support superpages?

The TLB entry format must include a page size field.

b) [4 points] Describe 2 situations in which it may be desirable to *demote* a superpage.

1) On a write to a superpage, we don't want to consider the entire superpage to be dirty, so we demote the superpage and mark only the base page that was actually written as being dirty.

2) Under memory pressure, we may want to demote a superpage so that we can detect if all the base pages in the superpage are actually still being used, allowing us to make better choices about what pages to evict.

c) [4 points] Explain how the use of superpages can lead to fragmentation, and what can be done about it.

Superpages creation/allocation requires multiple contiguous free physical memory pages (equal to the superpage size). With multiple page sizes, the unit of allocation is no longer uniform, and "holes" of different sizes appear in the physical memory space. Eventually, there may be enough free memory for a process, but no contiguous chunks large enough to create the desired superpage size. "Wired" pages which can't be removed from memory and persistent file cache pages contribute to the problem. Solutions include clustering "wired" pages together, freeing inactive file cache pages more aggressively, re-locating allocated pages to create larger free contiguous chunks (coalescing), and biasing page reclamation to prefer pages that restore the largest contiguous chunks. [Only one problem source and one solution needed in answer.]

3.[16 points; 4 each] Page**Placement**

Consider the following snippet of code that must run on a system with a 4kB page size and a 16kB physically-indexed direct-mapped data cache with a 16 byte cache line size.

```

...
page_size = 4096;
array_size = 2*page_size;
char *a = (char *)malloc(array_size);
char *b = (char *)malloc(array_size);

for (int i=0; i < array_size; i++) {
    a[i] = i % 256;
    b[i] = a[i] >> 1;
}

```

Assume the loop is the first access to the virtual memory allocated for arrays **a** and **b**, and that the variable **i** is allocated in a register (so it won't take space in the cache).

a) What is the minimum number of cache misses that will occur during the execution of this loop? What is the maximum number of cache misses?

Minimum - 4 pages of data used and 4 pages fit in cache, misses only on first access to each line.

$$== (4 * 4096) / 16$$

$$== 1024$$

Maximum - a and b conflict with each other in the cache, misses for a and b on each iteration of the loop

$$== 2 * 2 * 4096$$

$$== 16384$$

b) Suppose that the OS is using *page coloring* for page placement, and that array **a** starts at address 0x8047000. Give an address for **b** that will minimize cache misses, and an address for **b** that will maximize cache misses. Draw a picture showing how the pages of **a** and **b** are mapped to the cache in each case.

Minimize: address of b = 0x8049000

0x8048000	a[4096] - a[8191]
0x8049000	b[0] - b[4095]
0x804a000	b[4096] - b[8191]
0x8047000	a[0] - a[4095]

Maximize: address of b = 0x804b000

0x8048000 / 0x804c000	a[4096]-a[8191] & b[4096]-b[8191]
	Unused
	Unused
0x8047000 / 0x804b000	a[0]-a[4095] & b[0]-b[4095]

This question continues
on the following page.

c) Assuming we don't mind wasting virtual memory space, show how you would allocate **a** and **b** (using malloc) to guarantee best case behaviour, if the OS uses page coloring. Explain the general idea in English (use a picture if you like), and then give C code. (You only need to handle this specific example - you do not have to generalize to arrays of arbitrary sizes).

Allocate a single chunk of memory large enough to hold both a and b, and align the start of a to the start of a page. Set b to begin 2 pages after a. Since the pages of a and b will all have different colours in the virtual address space, they will also have different colours when physical pages are assigned, so we get best case behaviour.

Code:

```
...
page_size = 4096;
array_size = 2*page_size;
char *mem = (char *)malloc(2*array_size + 2*page_size);
char *a = (mem + page_size) & 0xffff000;
char *b = a + array_size;
```

d) If the OS is using *bin hopping* for page placement, will this result in the best or worst case behaviour? Draw a picture showing how the pages of **a** and **b** will be mapped to the cache in this case.

We will still get best case behaviour, but it will be the result of the order in which pages are allocated.

<i>a[0] - a[4095]</i>
<i>b[0] - b[4095]</i>
<i>a[4096] - a[8191]</i>
<i>b[4096] - b[8191]</i>

The key is that they alternate; a need not begin at the first location in the cache.

4. [14 points] Clocks in Distributed Systems

(a) [4 points] After contacting a time server, the local clock on some machine is discovered to be 4 seconds fast, with a current reading of 10:27:54.0 (hh:mm:ss). Explain why it is undesirable to set the clock back to the correct time immediately, and show how it should be adjusted so that it is correct when it reads 10:28:0.0. Assume that the hardware timer is programmed to interrupt every 10ms.

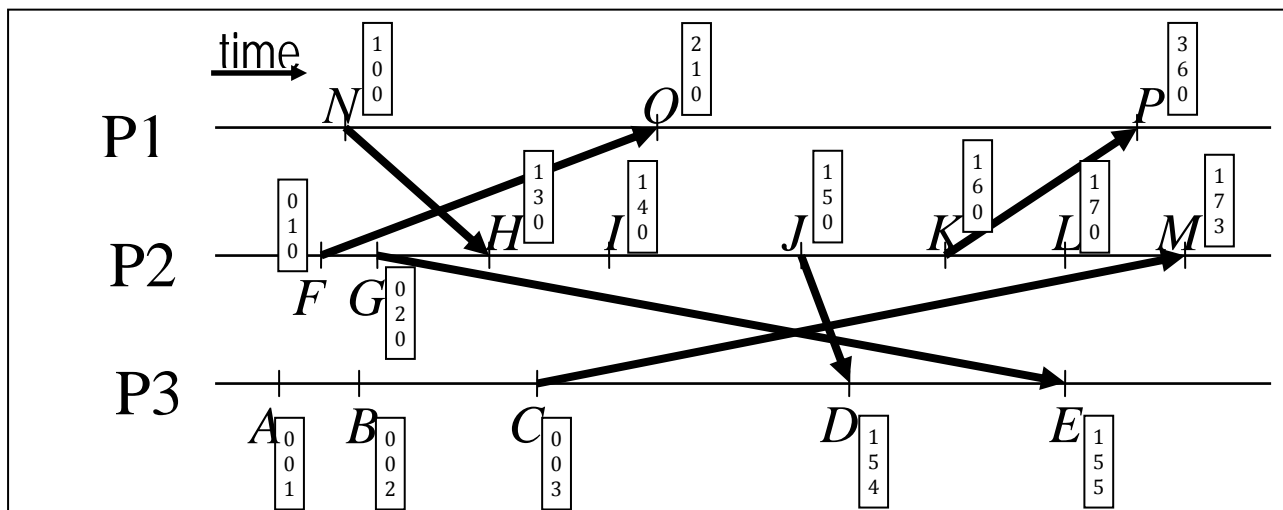
*Making time go backwards is likely to cause confusion for local processes (e.g. make). To adjust: We need to recoup 4s = 400ms. If we did it "all at once", this would be like letting the system run for 10 "local" seconds to 10:28:04.0 and then setting the clock back to 10:28:00. In this time, there will be 100 interrupts / sec * 10 sec = 1000 timer interrupts. Instead of doing it all at once, we will adjust the clock gradually by adding $x < 10ms$ to the time on each interrupt. Over 1000 interrupts we want time to increase by only 6 sec \rightarrow 6ms per interrupt.*

(b) [6 points] Under Lamport's "happens-before" relation, indicate whether the following statements are true or false for the events shown on the diagram below:

i) G happens before Q *False. [F \rightarrow Q, but we know nothing about G and Q]*

ii) N happens before E *True [N \rightarrow H and H \rightarrow J and J \rightarrow D and D \rightarrow E]*

iii) D happens before P *False [No causal relation from P3 to P1 at all.]*



(c) [4 points] On the diagram above, show the vector timestamps for each event.

[Doing c first should have made b easier!]

5. [10 points] Distributed

Agreement & Group Communication

(a) [6 points] Consider the Byzantine Generals Problem. Show that 2 lieutenants can agree on an action (attack or retreat) based on the order given by a commanding general, even if one of the participants is faulty, *if all messages are digitally signed by the sender.*

Note that loyal lieutenants need not obey a faulty/traitorous general. Assume default action is "retreat".

Case 1: Loyal lieutenants, traitorous general. Same order sent to both lieutenants.

→ *Lieutenants exchange orders from general, and obey, whatever it was.*

Case 2: Loyal lieutenants, traitorous general. Different orders sent to each lieutenant.

Round 1: L1 receives "attack" signed by Gen'l, L2 receives "retreat" signed by Gen'l

Round 2: L1, L2 send order they received to each other, signed by themselves.

→ *L1 receives ("retreat" signed by Gen'l) signed by L2*

→ *L2 receives ("attack" signed by Gen'l) signed by L1*

L1, L2 can now independently see that General sent conflicting orders, and so cannot be trusted. Therefore L1, L2 each take the default action and retreat.

Case 3: Loyal Gen'l, L1 disloyal (case where L2 is disloyal is symmetric)

Round 1: General sends "attack" to L1 and L2 with signature

Round 2: L1 sends "retreat" to L2 and signs it, but can't forge General's signature

→ *L2 sees that message from L1 was not signed by general, therefore L1 is lying and L2 obeys the order received from the General.*

(b) [4 points] The processes below are participating in a group communication protocol that provides *FIFO Atomic Broadcast* (meaning we combine FIFO Order and Total Order constraints). Show two possible legal orders for message delivery in P1 and P2:

P0	P1		P2		P3		
	Order 1	Order 2	Order 1	Order 2			
Broadcast m1	m1	a) m1	<div style="border: 1px solid black; padding: 5px; width: fit-content; margin: auto;"> Total order (atomic) means that P1 and P2 must deliver in the same order. So whatever order is used for P1, the same thing should appear in the boxes for P2. </div>		Broadcast m3		
		m3					
Broadcast m2		m2			b) m3		Broadcast m4
		m3			m1		
	m4	m2					
		m4					
		c) m3					
		m4					
		m1					
		m2					

6. [12 points; 4 each] Fault**Tolerance**

a) Explain the basic idea behind replicated state machines, and why they require atomic broadcast to provide a fault tolerant service.

*A state machine consists of state variables and commands that modify state variables (causing state changes) and/or produce output. They are deterministic, so the same sequences of commands applied to the same state leads to the same output. Fault tolerant services are provided by having multiple replicas of the service all handle the same set of requests. As long as **any** instance is alive, the client can get a response (assuming fail-stop failures).*

Atomic broadcast is required because we must guarantee that all replicas process the same commands in exactly the same order; otherwise they may not produce the same output. [This is a form of distributed consensus.]

b) In A3, the chatserver maintains state for each client, which is lost when the server fails and restarts. Although your clients could attempt to rebuild part of the state automatically, some problems can't be solved by the client (e.g. a new user contacting the new instance of the server and registering the user name you had with the old server before your client re-establishes a connection with the new server). Assuming you could change the server code, describe how you would improve failure recovery. Give as much detail as you can.

Have the server keep more detailed logs (on stable storage) about each client that has registered, and on recovery, rebuild the server state using the logs. No new clients would be accepted until the state was recovered. Additional checks would be needed to detect an existing client re-registering with the new instance of the server (the new server would be using different port numbers and possibly run on a different host, so the clients would still have to detect server failure and re-establish the communication channels).

c) Suppose you were not allowed to change the server code, but you were able to convince the maintainer of the service to run multiple instances of the chat server, and the location service was modified such that the web-hosted text file contained parameters (e.g. hostname and port numbers) for *all* currently-running instances of the chat server. (The servers themselves will not be aware that there are multiple servers running). Explain what you could do in the client code to improve fault tolerance and failure recovery.

Clients can contact and register with multiple servers; each chat message from a client is sent to all servers. Clients must make sure they keep track of the member_id assigned by each server, and use it in communication with that server. If we assume the server failure model is fail-stop (as in the assignment), then the client can display a chat message as soon as it arrives from any server. Additional care will be needed to detect and discard duplicate copies of chat messages arriving from the additional servers. One simple strategy is for clients to include a sequence number at the beginning of each chat message. The pair <member_name, sequence #> uniquely identifies each message from each client so that duplicates can be detected. Confusion will result if all users are not running clients that employ the same strategy.

Clients will continue to detect server failure, as before, and will check with the location server for any new servers that they can add to the set as a replacement.

7. [6 points] Reliable Storage

Describe how LFS handles crash recovery and ensures file system consistency.

LFS keeps two special "checkpoint regions" that record (i) the locations of the inode map blocks, (ii) the segment usage table, (iii) the address of the last block written in the log, and (iv) a timestamp. On crash recovery, the checkpoint region with the most recent timestamp is used. The file system state (inode map and end of log) is consistent when the checkpoint is written, so we could start from that point, but modifications made between checkpoints would be lost. Instead, LFS rolls forward, reading the log from the end location recorded in the checkpoint and applying changes according to the operation log stored with each segment, thereby recovering a more up-to-date version of the file system while maintaining consistency.

8. [6 points] Security

Discuss why buggy programs so often lead to compromised systems, and give one example of an OS security mechanism intended to address this problem.

Certain classes of bugs (such as buffer overflows or format string vulnerabilities) result from improper handling of user input, and allow an attacker to gain control over the process running the buggy program. By itself, this would not lead to a compromised system, just a compromised process, but many OSs do not provide proper separation of authorization or privilege domains. (For example, setuid programs in Unix, or allowing any user to write the registry in Windows) In many cases, this means that a compromised process can in fact be used to compromise the system.

FreeBSD jails are an example of an OS security mechanism that addresses this problem. The idea is to give a process (and all its descendants) an isolated view of the system, including a limited view of the file system and its own superuser account, which can only affect things contained within the same jail.

Extra space. Please indicate clearly which question(s) you are answering here, if any.

Total marks = (89)

End of test