Duration: 2 hour 30 min

Examiner: A. Goel

Please fill your student number and name below and then read the instructions below carefully.

Student Number: _____

First Name: _____

Last Name: _____

Instructions

Examination Aids: No examination aids are allowed.

Do not turn this page until you have received the signal to start.

Do not remove any sheets from this test book. Answer all questions in the space provided. No additional sheets are permitted. Use the last blank page as scratch space. This page will not be marked.

This exam consists of 8 questions on 21 pages (including this page). The value of each part of each question is indicated. The total value of all questions is 125.

For the written answers, explain your reasoning clearly. Be as brief and specific as possible. Clear, concise answers will be given higher marks than vague, wordy answers. Marks will be deducted for incorrect statements in an answer. Please write legibly!

Work independently.

MARKING GUIDE

Q1:	_____	(16)
Q2:	_____	(13)
Q3:	_____	(20)
Q4:	_____	(16)
Q5:	_____	(18)
Q6:	_____	(18)
Q7:	_____	(14)
Q8:	_____	(10)
BONUS MARKS:	_____	2
TOTAL:	_____	(125)

Question 1. **Short Questions** [16 MARKS]

Part (a) [2 MARKS] After the `fork()` system call, either the parent or the child process may run first. If you change this system call to always run the parent first, explain whether it could be beneficial in any way or not.

Part (b) [2 MARKS] You decide to use Unix-style feedback scheduling for scheduling the network packets of different sockets. To do so, you measure network usage by counting the number of packets sent or received on each socket. Describe two benefits of using this scheme.

Part (c) [4 MARKS] Your friend tells you that using locks, semaphores, condition variables are a thing of the past! They are not required to correctly implement multi-threaded application code if the application is run using a non-preemptive, user-level thread scheduler (e.g., think about Lab 2). Is your friend correct or not (or perhaps both!)? Clearly explain your answer.

Part (d) [2 MARKS] In a multiprocessor operating system, describe one benefit of using per-core data structures (e.g., a per-core run queue), and one benefit of using a global data structure (e.g., a global run queue).

Part (e) [2 MARKS] Does a traditional Unix file system use access control lists or capabilities or both? Briefly explain your answer.

Part (f) [2 MARKS] In the map-reduce framework, the map and reduce functions need to be deterministic. What are deterministic functions and why are they needed?

Part (g) [2 MARKS] Which ECE344 lab did you like the most and why? If you had to replace one of the labs, which one would you replace and why?

Question 2. Synchronization [13 MARKS]

The Niagara Rainbow bridge is used to cross the Canada-US border. Unfortunately, it is undergoing repairs and only one lane is open to traffic. To prevent accidents, traffic lights have been installed at either end of the bridge to synchronize the traffic going in different directions. A car can only cross the bridge if there are no cars going in the opposite direction on the bridge. Sensors at either end of the bridge detect when cars arrive and depart from the bridge, and these sensors control the traffic lights.

```
1
2
3
4  CA_car(int num)                          US_car(int num)
5  {                                        {
6
7       print(CAnum);                        print(USnum);
8
9       lock(ca_lock);                            lock(us_lock);
10
11      if (CA == 0) {                            if (US == 0) {
12
13          lock(bridge_lock);                        lock(bridge_lock);
14
15      }                                         }
16      CA++;                                     US++;
17
18      unlock(ca_lock);                          unlock(us_lock);
19
20      // multiple waiting Canadian         // multiple waiting US
21      // cars can cross the bridge         // cars can cross the bridge
22
23      print(CAnum crosses);             print(USnum crosses);
24
25      lock(ca_lock);                            lock(us_lock);
26
27      CA--;                                     US--;
28      if (CA == 0) {                            if (US == 0) {
29
30          unlock(bridge_lock);                      unlock(bridge_lock);
31
32      }                                         }
33
34      unlock(ca_lock);                          unlock(us_lock);
35
36 }                                        }
```

Part (a) [10 MARKS] The traffic light code shown in the previous page is used to synchronize the cars. Each Canadian car is a separate thread that runs the `CA_car` function, and similarly, each US car runs the `US_car` function. Assume that the `print()` function runs atomically. You find that the code works correctly, so there are no car crashes. However, sometimes all the US cars keep crossing the bridge, while the Canadian cars keep waiting for a long time. You are tired of this behavior and want to fix this problem.

To understand this problem further, you observe the cars as they arrive. Say the cars arrive in the following order, as output by the `print()` function (from left to right):

US1 US2 US3 CA1 CA2 US4 US5 CA3

You find that the US4 car can cross before the CA1 or CA2 cars. To ensure fairness, you would like CA1 and CA2 to cross before US4, and similarly US4 and US5 should cross before CA3.

Modify the code shown in the previous page to ensure such fairness (you do not need to remove any existing lines of code). Make sure that multiple cars (going one way) can cross the bridge together. For example, US1, US2 and US3 (or CA1 and CA2) should be able to be on the bridge at the same time, and can cross in any order.

Hint: You need one semaphore, and just calls to `sem_down()` and `sem_up()` operations to implement this behavior. Show the semaphore operations, and make sure to initialize the semaphore.

Part (b) [1 MARK] Does your code make any assumptions? If so, state them.

Part (c) [2 MARKS] Does your solution increase or decrease the bridge throughput, i.e., the total number of cars crossing the bridge per unit time? Circle your answer and explain why.

Increases throughput Decreases throughput

Question 3. **Paging** [20 MARKS]

A processor uses 48 bit virtual addresses. Its memory management unit uses a four-level page table scheme with a page size of 4 KB. The page table entries are 32 bits wide, and all page tables (except the top-level page table) require exactly a single page frame. The memory management unit is designed to be able to address a maximum of 1TB of physical memory. The frame number is stored in the lowest order bits in the page table entry as shown below:

Page table entry:

Control Bits	Frame number

Part (a) [2 MARKS] How many page table entries are stored at each level of the page table?

Top level:

Second level:

Third level:

Fourth level:

Part (b) [1 MARK] The processor manufacturer is interested in updating the paging MMU and has hired you to consider alternative paging designs. You know that programs are getting larger and so you propose to change the paging MMU to use a three-level page table scheme with 16 KB pages. You do not make any other changes to the processor or the paging scheme. How many page table entries would be stored at each level of the new page table scheme?

Top level:

Second level:

Third level:

Part (c) [8 MARKS] The manufacturer is concerned that your three-level scheme may have significant memory overheads. To make a reasoned argument about the benefits or drawbacks of the three-level scheme, you do some back of the envelope calculations of memory overheads of paging for the program stack. Assume that the program stack starts at the top of the 48-bit virtual address space of the program and grows downwards. In the table below, show the memory requirements (in KB) for all the page tables that are needed to access the stack in the four and the three-level paging scheme as the stack grows over time. You may provide your answer as a formula or as a final value. If you make any assumptions, write them down.

Stack Size	Four-level Page Tables (in KB)	Three-level Page Tables (in KB)
4KB		
16KB		
4MB		
64MB		

Part (d) [2 MARKS] Based on the above analysis, is there a crossover point (in terms of stack size) below which one scheme requires less memory for the page tables than the other, but above which it requires more memory than the other? If so, what is the crossover point? Write your answer in KB or MB. If you make any assumptions, write them down.

Part (e) [2 MARKS] Consider a program that has larger memory requirements over time. Assume that it can be larger because its regions (segments) are larger, or because it has more regions (e.g., mmapped regions), but not both. What would you suggest to the manufacturer in the two cases shown below. Briefly explain your answer.

	Use Four or Three-Level Page Table
Larger regions	
More regions	

Part (f) [4 MARKS] A partial last level page table, with page table entries on each row, is shown below. The start of the table is shown at the top and memory addresses grow downwards, as shown in the first column.

Page Offset				
00	e1	7a	d9	12
04	00	03	80	00
08	89	2d	c6	d5
0c	b2	d3	53	58
10	8a	be	9b	09
14	80	05	00	09
18	22	9c	18	67
1c	b5	37	23	8f
20	b6	94	c6	d8
24	cc	f7	3f	e8
28	ab	13	22	40
2c	b3	81	11	00
30	80	d6	7b	e0
34	0f	33	b4	56
38	d9	3b	78	98
3c	9c	31	76	b2

The MMU will use this table to translate the virtual address 0xf23d0c00974c. What is the physical address corresponding to this virtual address using the four and the three level paging schemes? Show your calculations.

	Physical Address
Four-Level Page Table	
Three-Level Page Table	

Part (g) [1 MARK] Based on the page table design, which paging scheme (four or three level) provides more flexibility for the OS to implement virtual memory management? Briefly explain why.

Question 4. Page Replacement [16 MARKS]

Two processes, P1 and P2 are executing on a system with 8 pages of physical memory. The OS runs the page replacement algorithm on demand, i.e., when a frame needs to be allocated and no free frames are available. At that time, the replacement algorithm uses the global clock algorithm to make one frame available. The clock algorithm tracks the status of pages and frames using the per-process page tables and the coremap shown below.

Initial State

P1 Page Table

Page	Valid	Reference	Frame
0	1	1	5
1	1	0	6
2	0	0	-
3	0	0	-
4	1	1	2

P2 Page Table

Page	Valid	Reference	Frame
0	0	0	-
1	1	0	1
2	1	1	7
3	1	1	3
4	1	1	4

Coremap

Frame	Allocated	ID	Page
0	0	-	-
1	1		
2	1		
3	1		
4	1		
5	1		
6	1		
7	1		

Step 1

V	R	F

A	I	P

Step 2

V	R	F

A	I	P

Step 3

V	R	F

A	I	P

Step 4

V	R	F

A	I	P

Part (a) [2 MARKS] Fill in the initial state of the coremap using the page tables for P1 and P2. The Allocated column (already filled), is the allocation status of each page. The ID is the process ID.

Part (b) [14 MARKS] Suppose the two processes run and access 4 pages as shown below.

Step 1: P1: Page 1
Step 2: P1: Page 2
Step 3: P1: Page 3

Context switch

Step 4: P2: Page 0

Show the changes to the page tables and the coremap after each step on the previous page. Assume that after each step, the process can access the corresponding page (e.g., P1 can access Page 1 after Step 1). When a page table or coremap entry changes, update the entire entry. Assume that when the clock replacement algorithm runs for the first time, the clock hand is pointing at Frame 0.

Question 5. TLB and Memory Management [18 MARKS]

Your are designing a high-performance, CPU-bound application called FAST that processes lots of data in (DRAM) memory. When you measure the performance of FAST using a testing framework, you find that FAST behaves erratically, performing well initially but then it becomes slower when it runs for a long time (have we seen this before?). Using profiling tools, you find that the problem is caused by the heap allocator (i.e., `malloc` library).

To understand this problem further, you look at the code of the `malloc` library, and find that it uses linked list allocation as shown below. On each call to `malloc(size)`, the library allocates a chunk of `8 + size` bytes. The 8 bytes are for the header (shown with a dotted pattern). The header tracks whether a chunk is allocated or free, and it has a pointer to the next chunk.

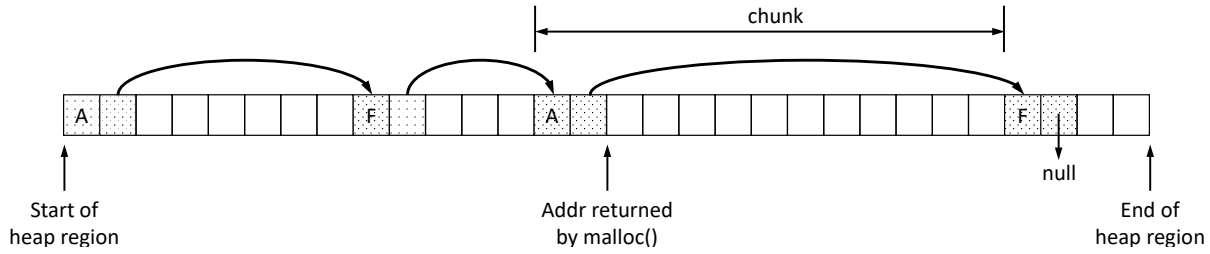


Figure 1: Malloc Linked List Implementation

You instrument FAST with a debugging function of the `malloc` library called `total_heap_size()` that prints the amount of memory currently allocated to the heap region of the program. FAST is implemented as a multi-threaded program. You configure FAST to run with 4 threads on a 4 core UG lab machine. The different threads allocate memory dynamically using `malloc()` from the same heap region of the program.

Part (a) [4 MARKS] You find that as long as `total_heap_size()` returns less than roughly 480 KB, FAST works well. Otherwise, the performance of FAST degrades significantly. You suspect the problem is with TLB usage. You look at the processor specifications, and find that each core has its own TLB, and the page size of the paging MMU is 4 KB. Based on your calculations, how many entries does each TLB have? Briefly explain your answer. Hint: In each of these questions, think about how the linked-list `malloc` implements allocation and freeing of memory.

Part (b) [4 MARKS] While debugging the performance problem in FAST, you get side tracked because you recently learned about the `mmap()` system call and would like to make FAST a more reliable program. So you replace each FAST thread with a process. Then you use `mmap()` to map a large file in the address space of each of the processes. The file is mapped at the same virtual address range in the different processes. You know that this new mapped region is shared by the different processes, and you reimplement a new `process_malloc` library using this mapped region. Your new implementation still uses the linked list implementation. Does the performance of the process-based FAST application degrade similar to thread-based FAST? If so, does it degrade when the `total_heap_size()` is 1) much less than 480 KB, 2) roughly 480 KB, or 3) much more than 480 KB? Briefly explain your answer in terms of TLB usage. You can ignore the cost of reading or writing the file on disk.

Part (c) [4 MARKS] Coming back to thread-based FAST, you decide to reimplement a new `thread_malloc` library in which each thread uses its own heap area. To do so, you split the heap region into roughly four parts (called t-regions), and each thread allocates memory from its own t-region. Your new implementation still uses the linked list implementation within a t-region. You can assume that each t-region is sized on program startup and does not need to grow. Does the t-region based FAST application perform similar to the original thread-based FAST? If not, how does its performance change? Briefly explain your answer in terms of TLB usage and `total_heap_size()`.

Part (d) [2 MARKS] Can you think of any other reason (besides TLB usage) why t-region based FAST may perform slower/faster than the original FAST?

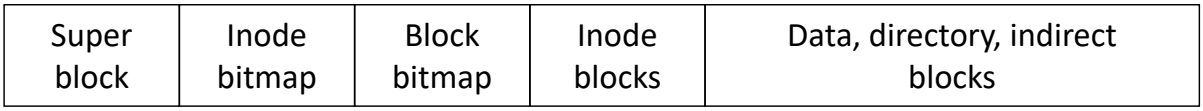
Part (e) [4 MARKS] You would like to improve the performance of FAST so that it can handle much larger amounts of heap memory, without suffering as much from TLB-based performance degradation. Suggest a way in which you would change the `malloc` library implementation to improve its performance.

Bonus. [2 MARKS]

As a bonus, suggest a second, different way to change the `malloc` library implementation to improve its TLB performance. If your changes require any changes to the callers of the library (e.g., the FAST program), mention what changes are needed.

Question 6. File System Design [18 MARKS]

The `testfs` file system that you have been working on in the lab has an annoying limitation. It only allows you to create a fixed maximum number of files. If you try to create any additional files, you are returned an ENOSPC error, and the file (or directory) creation fails. The `testfs` file system layout is shown below:



Part (a) [2 MARKS] Based on the file system layout, clearly explain why the ENOSPC error is returned.

Part (b) [8 MARKS] You modify `testfs` so that it can create any number of files, as long as there is enough space on disk. You call this file system `cxfs` (c - creates lots of files, x - because it sounds good). To ensure that you don't spend too much time writing and testing new code, you should make as **few changes** as possible to `testfs`. Hint: Think about other resizeable data structures in `testfs`.

Show the format of the `cxfs` file system. Using this format, describe the changes you have made to `testfs` to implement `cxfs`. Clearly explain how it imposes no limit on the number of files in the file system (other than when there is not enough space on disk).

Part (c) [2 MARKS] Does the `cxfs` file system perform better, worse or the same as the `testfs` file system? Briefly explain your answer.

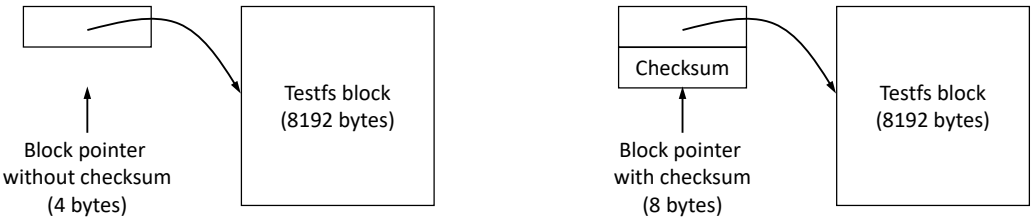
Part (d) [2 MARKS] You decide to improve the performance of both `testfs` and `cxfs` file systems by implementing a block-based caching scheme. The block cache uses LRU replacement. You find that the `cxfs` file system performs slight worse than the `testfs` file system. What could be the reason?

Part (e) [4 MARKS] Can you think of any way to modify the block replacement policy in `cxfs` so that its performance is close to `testfs`?

Question 7. File System Reliability [14 MARKS]

You have recently heard that disks are not reliable. After a sector is written, a *latent sector error* may occur where the contents of the sector may become incorrect, and further reads of the sector will then return incorrect contents. Worse, the disk may not know or tell the file system that such an error has occurred. As a result, the file system reads the sector and may return the incorrect data in the sector to the user application. Even worse, if the sector is part of file system metadata, then the file system may crash or cause further corruption when interpreting the data in the sector.

You decide to handle this serious data reliability problem in the `testfs` file system that you have been working on in the lab. You have heard that *checksums* are small fixed-sized data that are derived from some arbitrary-size data for the purpose of detecting errors. You decide to add a checksum to each file-system block pointer to help detect corruption of the pointed-to block. You call the resulting file system `sumfs`.



The picture above shows how a checksum can be added to a block pointer. The checksum is a value that is derived from the contents of the `testfs` block. You apply the string hash function (from Lab 1) to the contents of the `testfs` block by treating the entire (8192 byte) contents as a string (you can ignore the fact that the `testfs` block may contain the c-string termination character). The hash value that is generated is the checksum value: `checksum = Hash(block_data)`.

Checksums are used as follows: when a block is written, the checksum value is generated based on the updated block contents, and stored with the block pointer that points to the block. When a block is read (via the block pointer), the checksum is regenerated using the block contents, and this generated checksum is compared with the checksum stored in the block pointer. If there is a mismatch, the data that is read is not the same as the data that was written to the block, and hence a latent sector error has been detected. Note that checksums may lead to certain errors being undetectable due to hash collisions, i.e., `Hash(block_data_with_error)` is the same as `Hash(block_data)`, but we will ignore that issue.

The program listing below shows the inode structure of `testfs` on disk. If the questions below ask for a calculation, you may provide your answer as a formula or as a final value.

```
struct dinode {
  s32 i_type;           /* s32 is 4 bytes, i_type is file type */
  s32 i_block_nr[10];
  s32 i_indirect;
  s32 i_dindirect;
  s32 block_count;
  off_t i_size;         /* off_t is 8 bytes, i_size is file size */
};
```

Part (a) [2 MARKS] What is the maximum size of a file in the original `testfs` file system? Provide your answer in terms of number of `testfs` blocks.

Part (b) [2 MARKS] What is the maximum size of a file in the `sumfs` file system? Assume that the number of block pointers in the inode structure of `sumfs` are the same as in `testfs`. Provide your answer in terms of number of `sumfs` blocks.

Part (c) [1 MARK] Roughly, what is the ratio of the maximum size of a file in `sumfs` to the maximum size of a file in `testfs`?

Part (d) [2 MARKS] Suppose `testfs` allows creating a maximum of roughly 14000 files. Roughly, how many files can be created in `sumfs`? Show your calculations.

Part (e) [3 MARKS] Are all block corruptions detectable in `sumfs`? If so, explain why. Otherwise, explain where else you would add checksums to detect all block corruptions.

Part (f) [4 MARKS] With all the changes above to `sumfs`, suppose that you create a new file of 8 KB size on both file systems. Are the number of blocks that are modified the same or different in `testfs` and `sumfs`. Clearly explain your answer.

Question 8. Block-Level Crash Consistency [10 MARKS]

After gaining much experience with designing `sumfs`, a reliable, checksum-based file system, you realize that checksums can also help with ensuring crash consistency.[1]

You know that disks update a sector atomically, but they provide no such guarantees when updating multi-sector blocks. Assume that you need to update a single disk Block B consisting of 8 sectors. You have a spare Block S, and another Sector D, available to you. You design the following scheme for updating Block B atomically:

1. Write old version of Block B in memory to Block S on disk.
2. Write the checksum of the old version of Block B to Sector D on disk, flush blocks.
3. Write new version of Block B from memory to Block B to disk, flush blocks.
4. Write a special, invalid checksum value to Sector D on disk, flush blocks.

Part (a) [5 MARKS] Does this scheme update Block B atomically? If not, explain why there may be a problem. If it is correct, describe the block recovery procedure you would use after a crash, and also mention the step number at which the updated Block B is committed.

Part (b) [2 MARKS] The design above flushes blocks in Steps 2, 3, and 4. Briefly explain what happens on a block flush.

[1] If you haven't done so yet, you should read about checksums on the first page of the File System Reliability question before reading this question any further. In particular, you can ignore checksum collisions for this question.

Part (c) [1 MARK] Describe a potential problem with the scheme if blocks are not flushed in Step 2.

Part (d) [1 MARK] Describe a potential problem with the scheme if blocks are not flushed in Step 3.

Part (e) [1 MARK] Describe a potential problem with the scheme if blocks are not flushed in Step 4.

[Use the space below for rough work.]