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 9 questions on 16 pages (including this page). The value of each part of each question is indicated. The total value of all questions is 100.

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:	_____	(20)
Q2:	_____	(15)
Q3:	_____	(10)
Q4:	_____	(10)
Q5:	_____	(10)
Q6:	_____	(10)
Q7:	_____	(8)
Q8:	_____	(7)
Q9:	_____	(10)
TOTAL:	_____	(100)

Question 1. **Short Answers** [20 MARKS]

Part (a) [4 MARKS] **Threads:** Consider the user-level threads system that you implemented in the OS labs. Which of the following operations require support from the operating system (e.g., require issuing system calls) in their implementation? For those that do, write Yes, and provide a reason. Do **not** write Yes, if an operation does not directly issue system calls, but simply uses another operation that does.

Operation	**Yes**	**System Call Explanation**
Thread switching		
Creating a thread		
Disabling interrupts		
Enabling interrupts		
Thread exits itself		
Thread preemption		
Thread sleep		
Releasing a mutex lock		

Part (b) [6 MARKS] **File Systems:** A program issues the following system call on a Unix file system:

```
ret = mkdir("/a/b", 755);
```

This system call creates directory "b" with Unix permission 755. Assume that the directory "a" exists and directory "b" is created successfully as a result of this call. Assume that the super block has already been read at boot time, but otherwise the buffer cache is empty, and all directories use one data block. Also, assume that the file system does not use journaling.

Show the types of blocks that are read and written when the system call is performed. Add a brief explanation (e.g., read inode of "a") for why the block is read or written beside each block type.

Read	Block Type	Explanation
1		
2		
3		
4		
5		
6		
7		
8		
9		

Write	Block Type	Explanation
1		
2		
3		
4		
5		
6		
7		
8		
9		

This question continues
on the following page.

Part (c) [4 MARKS] **Scheduling:** If a thread is CPU-bound, would it make sense to give it higher priority for disk I/O than an I/O bound thread?

Part (d) [6 MARKS] **Virtual Memory:** Answer the following for a demand paging system. Assume that the physical memory size is fixed.

a) Can doubling the page size reduce the number of page faults? If so, describe how. If not, describe why.

b) Can halving the page size reduce the number of page faults? If so, describe how. If not, describe why.

c) Suppose a friend told you: If a system has lots of runnable threads, it can use them to hide the cost of page faults. Explain whether your friend is right, wrong, or both.

Question 2. Fill in the Blanks [15 MARKS]

Part (a) [3 MARKS] **TLB:** Consider a system with a software-managed tagged TLB. The tag uses 6 bits. The OS supports running a maximum of 1024 processes at a time. The processor provides the following options, in increasing cost, for programming the TLB: 1) invalidate a specific TLB entry, 2) invalidate the entries with a given tag, 3) flush the TLB (invalidate all entries). For each of the following events, indicate whether the OS software should take no action with respect to the TLB (**N**), invalidate an entry in the TLB (**I**), invalidate all entries associated with a tag (**T**) or flush the TLB (**F**). Use the most efficient option available. You may use the space on the right to explain your answer (optional).

_____ Context switch

_____ Page eviction

_____ Return from a recursive function call

_____ Process exit

_____ Copy-on-write fault

_____ Fork system call, without copy-on-write

Part (b) [4 MARKS] **OS Concepts:** Hardware support for operating systems includes privileged instructions (**P**), MMU support for virtual memory (**M**), interrupts (**I**), and the trap instruction (**T**). Indicate how the OS takes advantage of (one or more of) this hardware to ensure or support the following:

_____ a program doesn't modify the memory of another

_____ a program doesn't access hardware directly

_____ a program doesn't run forever

_____ a program doesn't jump to an arbitrary OS address

_____ a program doesn't modify its virtual memory mapping

_____ a program updates a block on disk

_____ a program sends a message to another process

_____ a program waits to receive a message from another process

This question continues
on the following page.

Part (c) [5 MARKS] **Disk scheduling:** Your OS *only* allows reading and writing whole files (similar to the web server implementation in your lab that read entire files). When files are accessed, the OS issues disk requests for all the blocks of the file. These requests can be scheduled using various disk scheduling algorithms.

We have discussed the First-Come-First-Served (FCFS), Shortest-Seek-First (SSF) and the Elevator scheduling algorithms in class. In addition, let's consider two other scheduling algorithms. The Shortest-File-First (SFF) is a non-preemptive algorithm that schedules all block requests for a file at a time, prioritizing the shortest file first. The Round-Robin (RR) algorithm is a preemptive algorithm that schedules requests one block at a time for all files that are being accessed concurrently.

These disk scheduling algorithms have various desirable properties shown below.

Exploit Locality: whether the algorithm exploits request locality.

Bounded Waiting: whether the algorithm bounds waiting time for all block requests it has received.

Fairness: whether the algorithm provides fairness, meaning that each block request is treated equally, regardless of what position on the disk the request is referencing.

Fill in the cells of the table below, using a check mark to indicate that the algorithm has the desired property. If the algorithm does not have the desired property, leave the table cell blank.

	Exploits locality	**Bounded waiting**	**Fairness**
FCFS			
SSF			
Elevator			
SFF			
RR			

Part (d) [3 MARKS] **Page Replacement:** You are familiar with the FIFO, Clock and LRU page replacement algorithms. The MRU page replacement algorithm replaces the *most* recently accessed page. Simply state **True** or **False** for the following statements regarding these four page replacement algorithms. You may use the space below each statement to explain your answer (optional).

	True or False
Clock and FIFO are easier to implement that MRU.	
A program accesses an array that is slightly larger than physical memory size repeatedly. MRU is the best choice for this program.	
A program accesses a large array just once. LRU is the best choice for this program.	
A program accesses the elements of a large array randomly. All the four page replacement algorithms will work equally well for this program.	
A program makes a deeply recursive function call. FIFO is the best choice for this program.	
All the four page replacement algorithm benefit from hardware support for tracking page accesses.	

Question 3. OS Design [10 MARKS]

In this question, you will add signals to the threads library that you implemented in the OS labs.

Recall that Unix signals allow the OS to notify a process (or thread) of some event of interest. For instance, the kernel delivers the SIGTERM signal to a process when a human user types `Ctrl-C`. If a process doesn't handle this signal, the process is killed by the kernel. Alternatively, the process can choose to ignore the signal, or perform some action (e.g., print "I am unkillable") upon receiving this signal.

A process registers a signal handler function that is invoked when a signal is to be delivered to the process. The signal can be delivered either by the kernel (e.g., when a process accesses bad memory), or at the request of another process (e.g., bash shell sends `Ctrl-C` to child process). Note that signal delivery interrupts the process. For example, in your labs, the SIGALRM signal was used to implement preemptive threading.

Given this information, describe how you would implement signal support in your threads library. In particular, your library should allow a thread to signal another thread. Make sure to describe the minimal, but complete, set of changes, including to data structures, interfaces, etc., that you would make to your current threads implementation. Write your answer as a set of bullet points, preferably with a short caption for each point. Be as precise as possible. Hint: think about the entire lifetime of the signaling mechanism.

Question 4. Red-Blue Race [10 MARKS]

Car owners are trying to decide whether red cars or blue cars are faster. They device an experiment in which all the red cars line up behind each other in a red line, and similarly all the blue cars line up behind each other in a blue line. Your job is to ensure that a red car and a blue car at the head of the two lines start the race at about the same time, i.e., either car should not start the race much before the other. You also need to ensure that the experiment makes progress as long as cars are present in both lines.

Part (a) [2 MARKS] You ask your friend John for help, and the pseudocode shown below is the best that he can come up with. A red car, upon arriving in the red line, invokes the function `red_lineup()`. When the function returns, the car starts racing. The blue solution is symmetric with the red solution (swap all occurrences of red and blue), and is not shown. Assume that `signal()` wakes up threads in the order they issued the `wait()`.

```
red_lineup()
{
        lock(lock);
        red_waiting++;
        while (blue_waiting == 0) {
           wait(red, lock);
        }
        signal(blue, lock);
        red_waiting--;
        unlock(lock);
}
```

You think about this solution and realize that it doesn't work correctly. Explain why.

This question continues
on the following page.

Part (b) [3 MARKS] Now you ask your friend Mary for help, and the pseudocode shown below is the best that she can come up with.

```
red_lineup()
{
        lock(lock);
        red_waiting++;
        if (blue_waiting == 0) {
           wait(red, lock);
        } else {
           signal(blue, lock);
        }
        red_waiting--;
        unlock(lock);
}
```

You think about this solution and realize that it still doesn't work correctly. Explain why.

Part (c) [5 MARKS] You think hard about why John and Mary's solutions didn't work and realize that a small change will make the solution work. Show that solution below.

Question 5. **Deadlocks** [10 MARKS]

Part (a) [5 MARKS] Alice, Bob, and Carol go to a Chinese restaurant at a busy time of the day. The waiter apologetically explains that the restaurant can provide only two pairs of chopsticks (for a total of four chopsticks) to be shared among the three people. Alice proposes that all four chopsticks be placed in an empty glass at the center of the table and that each diner should obey the following protocol:

```
while (!had_enough_to_eat()) {
    acquire_one_chopstick(); /* May block. */
    acquire_one_chopstick(); /* May block. */
    eat();
    release_one_chopstick(); /* Does not block. */
    release_one_chopstick(); /* Does not block. */
}
```

Can this dining plan lead to deadlock? Explain your answer.

Part (b) [5 MARKS] Suppose now that instead of three diners there will be an arbitrary number, `D`. Furthermore, each diner may require a different number of chopsticks to eat. For example, it is possible that one of the diners is an octopus, who for some reason refuses to begin eating before acquiring eight chopsticks. The second parameter of this scenario is `C`, the number of chopsticks that would simultaneously satisfy the needs of all diners at the table. For example, Alice, Bob, Carol, and one octopus would result in `C = 14`. Each diner's eating protocol is shown:

```
int s;
int num_sticks = my_chopstick_requirement();
while (!had_enough_to_eat()) {
    for (s = 0; s < num_sticks; ++s) {
        acquire_one_chopstick(); /* May block. */
    }
    eat();
    for (s = 0; s < num_sticks; ++s) {
        release_one_chopstick(); /* Does not block. */
    }
}
```

What is the smallest number of chopsticks (in terms of D and C) needed to ensure that deadlock cannot occur? Explain your answer.

Question 6. **Paging** [10 MARKS]

You have been hired by the MobARM Corporation to develop the virtual memory architecture on their next generation mobile processor. The virtual memory design has the following parameters:

- Virtual addresses are 54 bits.
- The page size is 64K byte.
- The architecture allows a maximum of 128 Terabytes (TB) of physical memory.
- The first- and second-level page tables are stored in physical memory.
- All page tables can start only on a page boundary.
- Each second-level page table fits exactly in a single page frame, but the first-level page table may be larger than a page frame.
- There are only valid bits and no other extra permission, or dirty bits.

Draw a figure showing how a virtual address gets mapped into a physical address. Your design should be as efficient as possible. You should list how the various fields of the virtual and the physical address are interpreted, including the size of each field (in bits). Show the format of the page table entry, the total number of entries each table holds, and the total size of each table (in bytes).

Question 7. **Sparse Files** [8 MARKS]

A file system stores files that are mostly empty as *sparse files* on disk. For example, if the length of a file is 1 MB, but the file only contains data in its first and last blocks, then the file system only stores these two data blocks. When an application reads a sparse file, the file system converts the empty blocks into ”real” blocks filled with zero bytes, without the application being aware of this conversion. Answer the following questions.

Part (a) [1 MARK] Briefly explain how sparse files might be implemented in a Unix file system.

Part (b) [3 MARKS] Say you wanted to upload the sparse files on your local disk to a file hosting service (such as Dropbox), and the file service supports sparse files as well. The problem will applications being unaware of sparse files is that your file synchronization client needs to read the entire sparse file, to determine whether any block contains all zeroes, before sending the sparse file to the file service. Show the function prototype of a new system call that you would implement that will not require reading the entire sparse file but just the blocks that are non-empty. How is the system call used? Are there any restrictions or limitations on the usage of this system call?

This question continues
on the following page.

Part (c) [2 MARKS] Briefly describe how a file system would implement your new system call.

Part (d) [2 MARKS] State two reasons why your file synchronization client will be more efficient when it uses your new system call.

Question 8. File System Extents [7 MARKS]

In an extent-based file system, storage is allocated in variable-sized, contiguous disk sectors called *extents*. Assume that extents are sized in multiples of blocks (e.g., 1 block, 13 blocks, etc.). In a Unix-style file system, suppose the only change you make to the on-disk format is that all block pointers are replaced with extent pointers. Each extent pointer is 8 bytes long, and contains a block location on disk, and the length of the extent in blocks. Answer the following questions. If you make any assumption, state them.

Part (a) [3 MARKS] State three advantages of using an extent-based file system design compared to a block based design.

Part (b) [3 MARKS] How would you change the block-based Unix file system code so that the extent-based file system will generally perform much better than the block-based design. Clearly explain your answer.

Part (c) [1 MARK] Give two examples where the extent-based file system design would be less efficient than a block-based design. Explain why.

Question 9. File System Consistency [10 MARKS]

Part (a) [4 MARKS] **Performance Optimization:** Suppose that you decide to improve the performance of the Berkeley fast file system (FFS) by removing all synchronous writes to disk. In your modified file system, all writes to disk are delayed 30 seconds (or until the kernel evicts blocks from the buffer cache). This allows two successive writes to the same inode, directory block, or other metadata structures to only require a single write to disk. Moreover, your kernel uses a disk scheduler that orders the writes to minimize disk seeks. Describe a way in which, after a power failure, a directory block may contain, instead of directory entries, data blocks from an existing user file. Make sure to show the sequence of system calls made before the power failure that cause this problem.

This question continues
on the following page.

Part (b) [6 MARKS] **Journaling:** Recall that journaling file systems are designed to speed up crash recovery. Explain clearly how a journaling file system would avoid or recover from the inconsistency caused by the scenario that you have outlined above. To do so, make sure to describe the types of blocks that are written, allocated or deallocated, and show the precise order of operations using a simple diagram. You do not need to describe any blocks in your scenario that are not relevant to crash recovery. Note that your journaling file system only journals metadata blocks. It may help to write your answer as a set of bullet points.

[Use the space below for rough work.]