Fill out the bottom of this page with your computing ID.
Write your computing ID at the top of each page in case pages get separated.

On my honor as a student I have neither given nor received aid on this exam.

1. Suppose an operating system with virtual memory runs programs which access virtual pages in the following sequence, where each virtual page is identified using a letter (A, B, C, etc.):

A	B	C	D	E	C	A	B	A	C	A	D

The system has

- four physical pages of memory to support storing these virtual pages, and
- unless otherwise stated, physical pages start unused

(The system does not do readahead or otherwise replace pages other than to bring in a virtual page the program is accessing.)

(a) (8 points) With a strict LRU (least recently used) policy, what is the minimum number of page replacements that must occur to satisfy the accesses listed above? Count loading a virtual page into an empty physical page as a replacement. Identify what replacements would occur.

Solution: 5 (initial loading, E replacing A) + 1 (A replacing B) + 1 (B replacing D) + 1 (D replacing E) = 8

(b) (Page access pattern reproduced:)

A	B	C	D	E	C	A	B	A	C	A	D

Suppose we approximate LRU (least recently used) by:

- maintaining an ordered list of physical pages
- each time a page replacement is required, *including to replace an empty physical page*:
  - identify which pages have been accessed since the last page replacement
  - reorder physical pages in the list based on this information
  - choose a page from the end of the list

Since this technique can't precisely identify the order in which pages were accessed between replacements, it may need to break ties, which creates multiple possibilities for what replacement occurs.

i. (4 points) With this policy, what is the first page replacement that does not replace an empty page and what are the possibilities for what is replaced?

Solution: first access to E — only option is A

ii. (8 points) With this policy, what is the second page replacement that does not replace an empty page and what are the possibilities for what is replaced?

Solution: second access to A — replacing B (previous miskeyed as: options are B and D; this should be corrected OUTSIDE OF TPEGS)

2. Suppose an operating system with 4KB pages is running exactly two processes, both running `foo.exe`. **Each of the two processes** has the folllowing identical memory layout (but have different contents in their stack and heap):

   - 0x0-0xFFF: inaccessible (segfault)
   - 0x1000-0x2FFF: mmapped read-only backed by `foo.exe` bytes 0x0 through 0x1FFF
   - 0x3000-0x4FFF: mmapped copy-on-write backed by `foo.exe` bytes 0x2000 through 0x3FFF, initially the same as `foo.exe`'s contents
   - 0x5000-0x5FFF: heap
   - 0x6000-0x6FFF: inaccessible (segfault)
   - 0x7000-0x7FFF: stack
   - 0x8000+: inaccessible (segfault)

(a) (5 points) If all pages accessible to the two processes are allocated into physical memory*early versions did not include "into physical memory"*, then how many physical pages of memory must be initially present in memory to support the processes' memory? (Ignore page tables and other kernel data structures.)

2 (foo.exe RO) + 2 (foo.exe COW) + 2 ·2 (stack + heap of each program) = 8

(b) (5 points) Suppose each of the processes writes to a 512-byte (`0x200` byte) global buffer located at address `0x3800`. Then, (if all accessible pages are allocated into physical memory) how many physical pages of memory must be present to support the processes's memory?

9 (1 extra copy of page written to; note original can be evicted)

3. (7 points) In an inode-based filesystem that implements copy-on-write snapshots, ***moving*** a file that is included in a prior snapshot from one directory to another in the current version of the filesystem typically requires which of the following operations? Ignore updates to modification times, creation times, etc. **Select all that apply.**

   ○ creating a new copy of the inode for the file

   ○ creating a new copy of the data blocks for the file

   ✓ creating a new copy of the inode for the source directory

   ✓ creating a new copy of one or more data blocks for the source directory

   ✓ creating a new copy of the inode for the destination directory

   ✓ creating a new copy of one or more data blocks for the destination directory

   ○ creating a new copy of the inode for the parent directory of the source and/or destination directory

4. (4 points) Which of the following are advantages of a capability-based system over an access-control-list-based system? **Select all that apply.**

   ✓ it is easier for a user to give a particular program they are running access to fewer resources than their other programs *keyed wrong originally!*

   ○ it is easier to have consistent names for resource (like files) across the entire system

   ○ it is easier to change the resources (like files) that a process has access to from outside the process while it is running *early version missed 'to'*

   ✓ it is less likely for a system call implementation to accidentally omit a check of whether a process has access to resources (like files)

5. Consider a simple disk controller which has exposes the following registers via the memory bus on a system that does not use an IOMMU:

   - a status register, which reads as 0 if the current transfer was successful, as 1 if there was an error, and as 2 if the current transfer is still in progress
   - a control register, to which 0 is written to start a read and 1 to start a write
   - two registers used to specify the first and last sector numbers to read or write
   - a register used to specify a base *physical address* used for DMA (direct memory access)

   The simple disk controller is also capable for sending interrupts to the CPU.

(a) (4 points) When a program triggers a read from the disk, assuming no read is already in progress, it would probably be most efficient for the operating system with many active programs to

○ run the disk device driver, which will loop checking the status register until either the disk read completes or a timer interrupt causes the OS to switch to another process

○ run part of the disk device driver, then switch to another program until a timer interrupt occurs during which the disk controller's status register reads as 0 or 1

√ run part of the disk device driver, then switch to another program until an interrupt is triggered by the disk controller

○ switch to another program until an interrupt is triggered by the disk controller

(b) (8 points) Some operating systems try to perform *zero-copy I/O*, so, for example, that when a program reads from (or writes to) a file the data is never copied except when the hardware moves it between hardware and memory.

With the interface above, suppose an operating system tries to implement zero-copy I/O for `read()` calls that read a whole number of contiguous sectors from the disk into a program's heap by

- setting the first and last sector numbers based on the current offset in the corresponding file and the size passed to read
- setting the base physical address to the physical address corresponding to the virtual buffer address passed to read (and refusing to perform the read at all if the buffer is not already allocated)
- starting a transfer by setting the control register, and
- waiting for the status register to indicate the transfer was successful

The operating system discovers that this sometimes works, but sometimes only part of the buffer appears to be filled by the `read()` call and some other memory is corrupted by the operation. This problem is observed only for reads which are larger than the page size. Despite these failures, the device and device controller appear to be working properly and always indicate the transfer was successful. Explain briefly what most likely caused the erroneous result.

Solution: part of the buffer in a different area of physical memory (two adjacent virtual pages not adjacent in phsyical memory) or page replacement moved the buffer or part of the buffer while the I/O was happening

6. Consider a FAT filesystem with 2048-byte clusters which contains

   - 10 directories in the root directory
   - 100 regular files in each of those directories, and
   - each of those regular files contains 3KB ($3 \cdot 2^{10}$ byte) of data

   Recall that in the FAT filesystem, directory entries are 32 bytes.

(a) (4 points) How many entries in the FAT (file allocation table) correspond to directories?

1 (root directory) + 20 (2 clusters per non-root directory) = 21

(b) (4 points) How many entries in the FAT (file allocation table) correspond to regular files?

2000 (each file requires two clusters)

7. Consider inode-based filesystems with:

   - 4 byte block pointers, 3 direct pointers/inode, 1 indirect pointer/inode, 1 double-indirect pointer/inode, and 1 triple-indirect/inode

   and two different block size settings:

   - filesystem A: 4KB blocks
   - filesystem B: 8KB blocks

   Each filesystem does not use fragments or extents.

   Consider using each of these filesystems on a disk with a little more than a 1000 MB of available space. After allocating space for headers, inodes, free block maps, etc., exactly 1000 MB is available on the disk with each filesystem.

(a) (6 points) Suppose it's possible to store 900MB of files and directory entries on filesystem A but not filesystem B using the disk.

Briefly describe a set of files and/or directories which would have this effect.

> **Solution:** 900 MB of files taking up exactly 4KB which would require 1800MB of blocks on filesystem B

(b) (6 points) Suppose it's possible to store 900MB of files and directory entries on filesystem B but not filesystem A using the disk.

Briefly describe a set of files and/or directories which would have this effect.

> **Solution:** files taking up exactly 16 KB will require 16 KB on filesystem B, but 20 KB on filesystem A (16 KB data + 1 indirect block), so around 900 MB of those files will exceed the 1000 MB capacity

8. An inode-based filesystem is asked to append K bytes of data to a file. This requires at least the following operations:

   A. updating the inode for the file
   B. writing data for new blocks that are allocated
   C. writing any new indirect or double-indirect, or triple-indirect blocks
   D. updating any existing indirect, double-indirect, or triple-indirect blocks that require updating
   E. updating the free block map entries corresponding to any newly allocated data blocks
   F. updating the free block map entries corresponding to any newly allocated indirect blocks

(a) (5 points) To ensure consistency, suppose the filesystem performs the updates in a carefully selected order and performs a recovery operation on reboot if the filesystem may have been interrupted before shutdown.

What would a good choice of the carefully selected order of the operations above?

(B, C, D in any order), (E, F in any order), A

(b) (4 points) To ensure consistency, suppose the filesystem instead uses redo logging and uses the log for both data and metadata. This filesystem treats the entire append operation as one transaction in its log. Suppose the filesystem updates the inode for the file, then *immediately* after this, the system loses power. What will happen after when the system is rebooted?

○ the system will do nothing since with redo logging it doesn't rely on the values of inodes for consistency

○ the system will perform each of steps B-F that were not performed before the loss of power, then write each of these operations and a commit message to the log

✓ the system will reperform all the steps A-F above, regardless of whether they were completed before the loss of power

○ the system will copy the updated inode to the log

○ the system will update the inode again to the state it had *before* the append operation to return the state before the operation

9. (4 points) In two-phase commit, suppose a worker sends a 'vote-to-commit' message to the coordinator, then loses power before it can send or receive any other messages. After the worker comes back up, it will first

○ determine it agreed to commit from its persistent log, then finish committing the transaction

○ send a 'vote-to-abort' message to the coordinator to indicate that the transaction will need to be retried

○ sometimes determine it agreed to commit from its persistent log, but sometimes need to contact the coordinator to determine this, depending on exactly when it lost power

✓ determine it agreed to commit from its persistent log, then resend that message to the coordinator and wait for a response

○ perform the transaction it agreed to commit locally

○ inform the other workers that the transaction committed

○ contact the coordinator to determine whether or not it agreed to commit, and wait for a response

10. The Andrew File System (AFS) guarantees that clients will see the results of modifications to a file after the program modifying the file closes it. Rather than having a stateless server like NFS, this is done using *callbacks*, where a client can arrange to proactively be notified about writes observed on the server.

(a) (8 points) Suppose a client program A opens a new file, then runs a loop that writes 1KB at a time until a total of 1MB is written, then closes the file. Each open, 1KB write, and close is a single system call. Assuming the client does as much caching as possible without breaking AFS's consistency model and does not start with anything cached, how many times must the client communicate with the server to complete this operation? Identify what system call (if any) triggers each of these messages.

Solution: 1 from open, 1 from close

(b) (4 points) Suppose the client from part (a) crashes in the middle of one of its write, and the client loses all cached data as a result of the crash. What will be the state of the new file be?

✓ it will not exist or be empty

○ it will contain only the data written before the crash

○ it will contain the data written before the crash and possibly some of the data that would have been written during the failed write

○ none of the above

(c) (4 points) Suppose the server loses power while the client from part (a) is writing. Assuming the server comes back up before the client finishes writing, what will the state of the new file be when the client finishes writing and closes it?

○ it will not exist

○ it will be empty

○ it will contain the data written before the crash

○ it will contain the data written after the crash

○ it will contain the data written before the crash and after the crash, but not during it

✓ it will contain all the data written, before, after and during the crash

○ none of the above

11. (4 points) Which of the following are true regarding stateful and stateless protocols? **Select all that apply.**

✓ one can implement a stateful service (e.g., a remote shell that remembers the current directory of the current user) with a stateless protocol.

✓ building a large-scale server for a stateless protocol is relatively easier than doing it for a stateful protocol.

○ a stateful protocol can be described as a finite-state machine (e.g., finite-state automaton) and each command will change the current state of the finite-state machine.

○ when a service using a stateless protocol crashes, the client must know the last state to recover from the crash.

12. (5 points) Which of the following is correct about the mailbox abstraction? **Select all that apply.**

○ mailbox abstraction can only describe the client/server model as there are only two machines described in the abstraction.

○ mailbox abstraction is not compatible with IPv6 network messages

○ mailbox abstraction includes network message formats (e.g., string)

✓ mailbox abstraction does not specify how to deliver messages through the network (in other words, a message delivery path is not part of the abstraction).

○ mailbox abstraction is obsolete because it does not support message encryption and is hence insecure.

13. Consider a virtual machine that is implemented using trap-and-emulate, does not have any special hardware virtualization support (beyond the minimum necessary to ensure that all instructions that behave differently in kernel and user mode trigger an exception in kernel mode), and runs the guest operating system machine without modifications.

(a) (8 points) A program in the guest operating system triggers a page fault, then the guest operating system's page fault handler allocates the page that triggered the fault, and restarts the faulting instruction. What exceptions (of any kind, including faults, traps, interrupts, etc.) does the hypervisor and/or host operating system handle during this process? For each exception, briefly describe its cause.

Solution: page fault for the guest program, protection fault for return-from-exception operation

(b) (8 points) Program A in the guest operating system makes a system call to read from a pipe, but no data is available so the guest operating system decides to switch to program B. Then, program B makes a system call to write to the pipe, and the guest operating system decides to immediately switch back to program A. What exceptions (of any kind, including faults, traps, interrupts, etc.) does the hypervisor and/or host operating system handle during this process? For each exception, briefly describe its cause.

Solution: system call (program A), protection fault for changing page table, protection fault for return-from-exception, system call (program B), protection fault for changing page table, protection fault for return-from-exception

14. (20 points) Suppose we want to implement a library that maintains a priority queue of items represented by strings that provides the following operations:

- `void EnqueueItem(int priority, string item)`: add an item to the queue with a particular priority
- `string DequeueItem(int minimumPriority)`: wait for an item with priority ≥ minimumPriority to be available in the queue, then dequeue it

When enqueuing an item an integer priority is specified which ranges from 0 to 2 inclusive, where items with a higher priority number are more important. (Items with priority 2 will always be dequeued first.)

Fill in the blanks in the following implementation sketch that uses mutexes and condition variables. Assume all global variables are initialized appropriately by code that is not shown and ignore any minor syntax errors, missing headers, etc. Refer to the comments for the methods provided by the PriorityQueue class.

Avoid busy-waiting. *In particular, a thread waiting in DequeueItem() should usually not be run (even just to check the head of the priority queue) when another thread calls EnqueueItem() for an item with a lower priority than the Dequeue's minimum.*

```
PriorityQueue queue;
    // provides methods:
    //   queue.EnqueueWithPriority(priority, item)
    //   queue.GetMaximumPriorityInQueue() —— returns −1 if queue is empty
    //   queue.Dequeue()
pthread_cond_t ready_at_priority_cv[3];
pthread_mutex_t queue_lock;

void EnqueueItem(int priority, string item) {
    pthread_mutex_lock(&queue_lock);
    queue.EnqueueWithPriority(priority, item);
    for (int i = 0; i <= priority; ++i) {
        pthread_cond_signal(&ready_at_priority_cv[i]);
    }
    pthread_mutex_unlock(&queue_lock);
}

string DequeueItem(int minimumPriority) {
    pthread_mutex_lock(&queue_lock);
    while (queue.GetMaximumPriority() < minimumPriority) {
        pthread_cond_wait(&ready_at_priority_cv[i]);
    }
    string result = queue.Dequeue()
    pthread_mutex_lock(&queue_lock);
    return result;
}
```