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. (4 points) Direct memory access (DMA) improves performance (compared to normal 'programmed I/O') primarily by

   ◯ allowing the operating system to avoid reading status registers

   ◯ lowering contention for a shared memory bus

   √ allowing data transfers to or from device to occur while user code is running

   ◯ using special I/O addresses to access devices

   ◯ avoiding the use of interrupts when performing I/O

   ◯ avoiding the use of bus adaptors

2. For this question consider an inode-based (Unix Fast File System-like) filesystem which does **not** use journalling.

   Consider creating a new file A in a directory B which is a subdirectory of a directory C. The new file contains 1 block of data on this filesystem. Ignore changes to update modification, creation, or other timestamps stored in the filesystems. Besides the writing the data for the file itself, which of the following data structures must always or sometimes be written or modified to do this?

(a) (1 point) one of the data blocks for directory B

√ always ◯ sometimes ◯ never

(b) (1 point) one of the data blocks for directory C

◯ always ◯ sometimes √ never

(c) (1 point) the inode for A

√ always ◯ sometimes ◯ never

(d) (1 point) the inode for B

◯ always √ sometimes ◯ never

(e) (1 point) the inode for C

◯ always ◯ sometimes √ never

(f) (1 point) the part of the free block map corresponding to C's data blocks

◯ always ◯ sometimes √ never

(g) (1 point) the part of the free block map corresponding to B's data blocks

◯ always √ sometimes ◯ never

(h) (1 point) the part of the free block map corresponding to A's data blocks

√ always ◯ sometimes ◯ never

3. For each of the following protection policies, identify whether using access control lists or capabilities would be better suited toward implementing it.

(a) (2 points) Decreasing the number of programs that are permitted to access a file while those programs are running.

√ access control lists *adjusted 2019-05-08 – this is probably the better answer; assuming the set of programs are identified by usernames, etc.* ◯ capabilities ◯ both equally suited

(b) (2 points) Allowing a program to delegate its access to a file to another program on the same system run on behalf of a different user.

◯ access control lists √ capabilities ◯ both equally suited

4. (4 points) Consider a filesystem that uses redo-logging and performs a rename operation that requires writing both a directory entry and, to update modification times, the directory inode. The filesystem will update the log to include this rename operation _____.

   ◯ in the same disk write as the update to the directory entry and directory inode itself
   √ before the corresponding updates to the directory entry and the directory inode
   ◯ after the corresponding directory entry update and before the correspodning directory inode update
   ◯ none of the above; the filesystem can order the log updates and directory entry and inode updates arbitrarily
   ◯ after the corresponding updates to the directory entry and the directory inode
   ◯ none of the above; the filesystem will not use the log for this operation

5. (6 points) Which of the following is true about an inode-based copy-on-write filesystem that supports snapshots? **Select all that apply.**

   √ appending to a file will require creating a new copy of the file's inode
   ◯ appending to a file will require creating a new copy of all the file's data
   ◯ appending to a file will require creating a new copy of the directory entry for the file
   ◯ moving a file from one directory to another will require creating a new copy of the file's data
   √ overwriting the data in a file will require writing the new data to a different location on disk
   ◯ this filesystem will not also support redo-logging/journalling

6. (7 points) Which of the following are properties resulting from NFS's close-to-open consistency model, where client A is only guaranteed read updates made by client B to a file on the network files system only if client B closes the file before client A opens it. **Select all that apply.**

   ◯ clients must contact the server every time a file is closed
   ◯ clients can sometimes avoid contacting the server when re-opening and re-reading a file they recently closed
   ◯ clients cannot return from a write system call without sending a message to the server
   √ clients can cache the contents of a file between calls to open on that file
   ◯ it is the strongest consistency model that can be implemented without making the server more stateful
   ◯ after two clients have a file open for writing at the same time and then close it, it is possible for the file to contain a mix of data written by both clients *not really resulting from the consistency model, but probably should've been dropped*
   ◯ this model requires the server to track which files each client has open

7. (5 points) Which of the following statements about the POSIX socket API are true? **Select all that apply.**

   ◯ for a server to handle multiple clients at once, it must call `listen()` and/or `bind()` multiple times
   √ a connected client's socket is assigned a port number on the client machine
   ◯ to connect a TCP socket as a client, one must specify a remote DNS name
   √ to connect a TCP socket as a client, one must specify a remote port number
   ◯ when one end of a connection `write()`s to a socket, the other end will receive the data written with *exactly one* `read()` call (assuming it receives the data at all)

8. Suppose a process is running on an operating system that supports memory-mapping of files and copy-on-write and uses 4KB (2^{12} byte pages). Its memory is laid out as follows:

   - `0x01000`–`0x03FFF`: read-only mapping of file 'foo.exe' (first `0x3000` bytes)
   - `0x04000`–`0x04FFF`: copy-on-write ('private') mapping of file 'foo.exe' (bytes `0x3000-0x4000`)
   - `0x10000`–`0x12FFF`: read/write shared mapping of file 'data.raw'
   - `0xFF000`–`0xFFFFF`: stack

(a) (5 points) Suppose that all the virtual memory this program can access is loaded into physical memory. Ignoring kernel data structures like page tables and process control blocks, how many pages of physical memory must be allocated to the process to do this?

8 (corrected from previous misposted version 2019-05-08; virtual pages 0x1, 0x2, 0x3, 0x4, 0x10, 0x11, 0x12, 0xFF)

(b) (5 points) Suppose this process writes 16 bytes to address `0x04010`, then forks, then the child writes 64 bytes to address `0xFF090`. If all virtual memory allocated to both processes is loaded into physical memory, then, ignoring kernel data structures like page tables and process control blocks, how many pages of physical memory must be allocated to the two processes?

9 (corrected from previous misposted version 2019-05-08 5PM; one copy of virtual page 0x4, two copies of virtual p

(c) (5 points) Suppose this process forks, then the child writes 1024 bytes (`0x400` bytes) to address `0x04090` and 5000 bytes (`0x1388` bytes) to address `0x10F00`, then both processes terminate. How many pages of data will the operating system eventually write to disk as a result?

3 (corrected 2019-05-09 — modify bytes 0x10F00-0x10FFF (0x100 bytes), 0x11000-0x11FFF (0x1000 bytes), 0x120

9. (8 points) Readahead is an operating system feature where sequential reads of files are detected and used to speculatively read later parts of the file. Which of the following is true about readahead? **Select all that apply.**

   ◯ a high-quality readahead implementation relies on

   ◯ implementing readahead for memory-mapped file reads requires keeping the mapped pages of the file invalid in the page table and emulating reads to them

   √ readahead usually decreases the amount of time a reading process is waiting and not runnable

   ◯ readahead usually decreases the amount of page replacements that occur

   ◯ readahead is much more difficult to implement on a system that uses a least-recently used approximation for page replacement

10. (6 points) Which of the following are possible attributes of a client/server filesystem protocol which supports **stateless servers**? **Select all that apply.**

    √ clients can write a whole file in one operation

    ◯ servers can inform clients which have cached copies of a file when that file is updated

    ◯ when communicating with the server, the client identifies open files using a process ID and open file descriptor number

    ◯ to recover from server failures without disrupting clients, the server requires a log with information about active clients

    √ values written could be reflected immediately by future reads *updated 2019-05-08: true as long as we don't care about caching*

    ◯ the server needs to be informed when a client crashes or loses power to avoid leaking memory

11. Consider an operating system that uses a second-chance page replacement strategy and is running exactly one process whose memory has virtual pages A, B, C, D, and E. Initially, the list of physical pages, and the state of their accessed bit (updated by the processor on a read or write) is as follows:

physical page #	virtual page loaded	accessed?
0	A	0
1	B	1
2	C	1
3	D	0

In accordance with the second-chance strategy, the operating system considers pages at the bottom of the list (so starting with page 3) for page replacement first, returning pages marked as accessed to the top of the list with the accessed bit clear rather than replacing them.

Suppose the process then accesses pages in the following order:

- A, B, A, B, E, D, B, C

(a) (16 points) List the page replacements that must occur as a result of accesses listed above, in order, identifying each physical's page's contents before and after the replacement.

Solution: E replaces D; D replaces C; C replaces A

12. Most operating systems maintain a reverse page table mapping, where given the address of a physical page, they can identify all of its page table entries.

(a) Briefly, give two examples of scenarios in which a physical page would have at least three page table entries.

i. (5 points) Example 1.

> **Solution:** three processes running the same executable sharing the same code page

ii. (5 points) Example 2.

> **Solution:** multiple processes mapping the same data file;

(b) (8 points) Give an example of why the reverse page table mapping is useful when implementing page replacement.

> **Solution:** once a page is chosen for replacement, to invalidate all of its page table entries; or, to determine if a page should be replaced, to read all of its dirty/accessed bits

13. Suppose a worker is running a system using two-phase commit that is processing several transcations.

It has processed transaction T1 and sent an agree-to-commit message for it, but has not received indication of whether the transaction is successful after a **very long** wait. It then is asked to 'prepare' a new transaction T2, which would conflict with the update requested by transaction T1.

(a) (4 points) What should this worker do for transaction T1 described above?

- ◯ abort the transaction and send an agree-to-abort message to the coordinator
- ◯ abort the transaction, waiting for the next time the coordinator resends a prepare message to send an agree-to-abort message
- ◯ send the same agree-to-commit to each worker that it sent to the coordinator
- ◯ abort the transaction and send an agree-to-abort message to each worker
- √ wait for a message from coordinator about whether the transaction should be committed
- ◯ perform the update specified by the transaction's original prepare message

(b) (4 points) What should this worker do for transaction T2 described above?

- ◯ send an agree-to-abort message to each worker
- √ send an agree-to-abort message to the coordinator
- ◯ send an agree-to-commit message to the coordinator
- ◯ wait for a message from coordinator about whether the transaction should be committed
- ◯ perform the update specified by the transaction's original prepare message
- ◯ send an agree-to-commit message to each worker

14. For this question, consider three filesystems:
    A. an inode-based filesystem, which
       - stores 256-byte inodes, containing 5 direct block pointers, 1 indirect block pointer, and 1 double-indirect block pointer
       - uses 4KB (2^{12} byte) blocks and 4 byte block pointers
       - tracks free blocks using a bitmap with one bit of storage per data block

    B. the inode-based filesystem in A, but with block pointers replaced with a list of 10 extents, each of which contains a starting block number and length in blocks
    C. a FAT32 filesytem with 4KB clusters

(a) Besides space used to store directory entries and free block bitmaps, but including space for the actual file data, inodes, relevant file allocation table (FAT) entries, and any other disk space that must be alloated, what is the minimal space each of these filesystems need to store a 1.5KB ($1.5 \cdot 2^{10}$ byte) file? You may leave your answers as unsimplified arithmetic expressions and identifying the components you are adding may help you get partial credit.

i. (4 points) Filesystem A

4KB + 256B

ii. (4 points) Filesystem B

4KB + 256B

iii. (4 points) Filesystem C

4KB + 4B

(b) Besides space used to store directory entries and free block bitmaps, but including space for the actual file data, inodes, relevant file allocation table (FAT) entries, and any other disk space that must be allocated, what is the minimal space each of these filesystems need to store a 16 MB (2^{24} byte) file? You may leave your answers as unsimplified arithmetic expressions.

i. (4 points) Filesystem A

16MB + $4 \cdot$ 4KB (4 indirect block w/ 2^{10} pointers) + $1 \cdot$ 4KB (double-indirect block) + 256B

ii. (4 points) Filesystem B

16MB (one extent is enough) + 256B

iii. (4 points) Filesystem C

16MB + $4 \cdot 1024 \cdot$ 4B

15. Consider a hypervisor that runs the guest operating system code in user mode and does not require significant modifications to the guest operating system. To implement virtual memory, this hypervisor uses shadow page tables, where valid entries are only added to the shadow page tables when a page fault occurs in the guest operating system. The shadow page table is the page table that is active in the hardware whenever the guest operating system is running.

    Assume this hypervisor does not protect (e.g. disable the writeable bit in) the shadow page table entries corresponding to the guest operating system's page table.

(a) (4 points) When the guest operating system kernel runs a privileged instruction that, on normal hardware, would invalidate its TLB, an exception occurs because the instruction is priviliged. What should the hypervisor's exception handler for this exception do before returning the next guest OS instruction?

- ◯ nothing (just return from the exception handler starting at the next guest OS instruction)
- √ invalidate one or more entries in the current shadow page table and corresponding TLB entries
- ◯ mark one or more entries in the current shadow page table as read-only (and invalidate corresponding TLB entries)
- ◯ clear the accessed and dirty bits in the guest operating system's page table
- ◯ invalidate the TLB (without editing the shadow page table)
- ◯ run the guest operating system's corresponding exception handler (and do not return to the next guest operatings system instruction)
- ◯ copy entries from the shadow page table into the guest operating system's page table (possibly making changes to metadata bits)

(b) Suppose the guest operating system's page table entry for virtual page number `0x1FFF` contains:

- a valid bit that is set,
- a kernel-mode-only bit that is set,
- a writeable bit that is set,
- the physical page number `0x4432`

and that the hypervisor has decided to locate the guest's physical memory contiguously starting at hardware physical page `0x1000`.

When the guest operating system is running in what the guest operating system believes is kernel mode, describe the corresponding shadow page table entry.

i. (1 point) Valid?
   √ set ◯ clear

ii. (1 point) Kernel-mode only?
   √ clear ◯ set

iii. (1 point) Writeable?
   ◯ clear √ set

iv. (5 points) Physical page number:
   ◯ `0x4432` ◯ `0x3432` ◯ none of the above; or this value does not matter ◯ `0x3FFF`
   ◯ `0x2FFF` √ `0x5432`
   ◯ `0x1FFF`

(c) (4 points) For which of the following operations will this hypervisor need to change the shadow page tables and/or switch to a new shadow page table? **Select all that apply.**

- ◯ when the guest operating system runs a privileged instruction to read from the keyboard
- √ when the guest operating system, while running its kernel, switches to a new page table
- √ when a program running in the guest operating system makes a system call
- ◯ when the guest operating system handles an interrupt triggered while the guest OS kernel was running

16. (25 points) Suppose we want to implement a data structure for matching exams to graders. The system will have a queue of exams and a queue of graders and provide a function for returning a pair of an exam and a grader, so it provides the following interface:

    - void InsertExam(Exam *exam)
    - void InsertGrader(Grader *grader)
    - void GetExamAndGrader(Exam** pExam, Grader **pGrader)

    The `GetExamAndGrader` function will wait until an exam and grader are queued, then set `*pExam` to dequeued exam and `*pGrader` to a dequeued grader. Complete the following implementation of these functions using mutexes and condition variables: (Assume that appropriate code to initialize the mutexes and condition variables is run elsewhere.) Some blank lines may not be needed.

```
const int MAX_QUEUE = 5;
list<Exam *> exams; list<Grader *> graders;
pthread_mutex_t lock; pthread_cont_t remove_ready, insert_ready;

void InsertExam(Exam *exam) {
    pthread_mutex_lock(&lock);
    while (exams.size() >= MAX_QUEUE) {
        pthread_cond_wait(&insert_ready, &lock);
    }
    exams.push_back(exam);
    if (exams.size() > 0 && graders.size() > 0) {
        pthread_cond_signal(&remove_ready);
    }
    pthread_mutex_unlock(&lock);
}

void InsertGrader(Grader *grader) {
    pthread_mutex_lock(&lock);
    while (graders.size() >= MAX_QUEUE) {
        pthread_cond_wait(&insert_ready, &lock);
    }
    graders.push_back(grader);
    if (exams.size() > 0 && graders.size() > 0) {
        pthread_cond_signal(&remove_ready);
    }
    pthread_mutex_unlock(&lock);
}

void GetExamAndGrader(Exam **pExam, Grader **pGrader) {
    pthread_mutex_lock(&lock);
    while (graders.size() == 0 || exams.size() == 0) {
        pthread_cond_wait(&remove_ready, &lock);
    }
    *pExam = exams.pop_front(); *pGrader = graders.pop_front();
    pthread_cond_broadcast(&insert_ready);
    pthread_mutex_unlock(&lock);
}
```