Please read carefully.
Please write clearly.

Definitely write your CS username here:

__________________________
SOLUTIONS

Exam length: 11 questions
Exam time: 2 hours
Grading (do not mark)

=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
Question 1 (6 points)
What’s the key difference between RAID-4 and RAID-5?

<<< RAID-5 distributes parity across all disks; RAID-4 concentrates it >>>
<<< all on one disk. >>>

Which is typically advantageous over the other, and why?

<<< RAID-5 is generally better, because it alleviates the bottleneck of >>>
<<< the parity disk on small-write workloads >>>

Which will perform better for small random writes: RAID-0 or RAID-5?

<<< RAID-0 >>>

Why? (Briefly.)

<<< No parity to recompute/update >>>

Question 2 (6 points)
How many disk failures can each of the following survive without data loss:

- 2-disk RAID-1: <<< 1 >>>
- 4-disk RAID-5: <<< 1 >>>
- 2-disk RAID-0: <<< 0 >>>
- 6-disk RAID-5: <<< 1 >>>
- 6-disk RAID-6: <<< 2 >>>
- 4-disk RAID-0: <<< 0 >>>
Question 3 (8 points)

Imagine a stacked/nested 9-disk "RAID-55", with the top-level being a 3-"disk" RAID-5 array, each member of which is itself internally a 3-disk RAID-5 array of physical disks. If you were to overwrite single sector in the top-level array, write how many read I/Os and how many write I/Os would ultimately be performed at the underlying PHYSICAL disks with each of the following parity-recalculation arrangements:

- additive at both levels

  \[ <<< 1R + 2*(1R + 2W) = 3R + 4W >>> \]

- subtractive at both levels

  \[ <<< 2R + 2*(2R + 2W) = 6R + 4W >>> \]

- additive at top level, subtractive at bottom level

  \[ <<< 1R + 2*(2R + 2W) = 5R + 4W >>> \]

- subtractive at top level, additive at bottom level

  \[ <<< 2R + 2*(1R + 2W) = 4R + 4W >>> \]
Question 4 (12 points)

When a file is first created via a creat() or open(..., O_CREAT) system call, how many hard links does it have?

<<< 1 >>>

Imagine your hard drive is full. In an attempt to free up some space, you call unlink() on a large file, but after doing so you find you have no more free space than when you started. What might have happened to lead to this counter-intuitive result?

<<< The file had multiple hard links; we didn’t remove the last one >>>

Consider the following sequences of commands (# characters indicate that the rest of the line is a comment). For each one, write either "Success:" and the output produced by the final 'cat' command, or "Error:" and a brief explanation of what would go wrong with it.

```bash
$ echo hello > f1    # write "hello" to f1
$ ln -s f1 f2        # create a symlink to it
$ rm f1              # unlink f1
$ cat f2             # read and print f2
```

<<< Error: f1 not found, ENOENT, dangling/broken symlink >>>

```bash
$ echo goodbye > g1  # write "goodbye" to g1
$ ln g1 g2           # create a hard link to it
$ rm g1              # unlink g1
$ cat g2             # read and print g2
```

<<< Success: "goodbye" >>>

Write whether each of the following statements is true or false:

- An inode cannot be freed while a symbolic link still points to it.
  <<< False >>>

- Symbolic links can point from one filesystem to another.
  <<< True >>>

- Hard links can point from one filesystem to another.
  <<< False >>>

- Hard links can be left in a "broken" or "dangling" state.
  <<< False >>>

- Symbolic links can be left in a "broken" or "dangling" state.
  <<< True >>>
Question 5 (14 points)

In a basic Unix-style filesystem, creating and writing some data to a new file involves writes to the data bitmap, the inode table, and a data block (other things might potentially be involved as well, but set them aside for the purposes of this question). If a crash or power loss occurs when some but not all of these writes have been performed, it can potentially leave the filesystem in an awkward state. For each of the following potential crash points, state (briefly) what the ill effects would be, or write "OK" if there would be none.

- Data bitmap written, inode table and data block untouched

<<< space leak >>>

- Data bitmap and inode table written, data block untouched

<<< garbage data in file (security problem too -- data leakage) >>>

- Data block written, inode table and data bitmap untouched

<<< OK >>>

- Data block and inode table written, data bitmap untouched

<<< inconsistent metadata: block in use but marked free >>>

Journaling is a common technique for avoiding inconsistencies such as these. A region of the disk is reserved as the journal, and transactions describing an update to the filesystem are first written to the journal in a form of write-ahead logging. The journal has a finite capacity, however, and will eventually fill up after enough transactions are written to it. At what point can the space in which a given transaction is recorded in the journal be deallocated and reused to store a record of a new transaction?

<<< When the old txn is checkpointed (written to primary FS structures >>>

What is the main drawback of journaling?

<<< Writes everything to disk twice >>>

What’s one way this drawback can be mitigated?

<<< Metadata-only journaling >>>
<<< OR: keep journal on separate (smaller, faster) storage device >>>
Question 6 (5 points)

There are three main components of an NFS file handle. Two of them are a volume identifier and an inode number. What is the third, and (in a sentence or two) why is it necessary?

<<< inode generation number. Needed to avoid conflicting accesses from >>>
<<< old file handles when same inode number gets reused after a file’s >>>
<<< last link is removed and its inode is deallocated. >>>

Question 7 (7 points)

Consider a data structure (such as a list of outstanding I/O requests) that might be accessed by both system call code and an interrupt handler in a device driver. In order to synchronize these two access paths, the data structure is protected by a spinlock.

- What may occur if this locking is done naively, with only basic acquire and release operations unaccompanied by anything else?

<<< deadlock >>>

- In order to protect against this possibility, an additional operation "X" must be performed when acquiring the spinlock, and a corresponding inverse operation "Y" is required when releasing it. What are X and Y?

X: <<< disable IRQs >>>

Y: <<< enable IRQs >>>

- When acquiring the spinlock, should X be done before or after doing so?

<<< before >>>

- When releasing the spinlock, should Y be done before or after doing so?

<<< after >>>

- Why is it necessary to strictly follow these ordering requirements?

<<< If interrupts are ever enabled while the lock is held, deadlock >>>
<<< can still occur. >>>
Question 8 (10 points)

Progressing from the basic Unix-style filesystem design, we looked at two more advanced designs, FFS and LFS, that achieved substantial performance improvements by changing the characteristics of the pattern of I/O requests issued to the disk.

Briefly describe the improved I/O behavior of each:

- FFS: <<< shorter seeks >>>

- LFS: <<< large, sequential writes >>>

FFS and LFS also both, as a central aspect of their design, divide the disk’s address space into a number of moderately large contiguous units (typically on the order of 1-100MB): block (or cylinder) groups in FFS and segments in LFS. This introduces a design parameter for each that must be given an appropriate value. Briefly describe the negative effects of each of the following:

- A too-large FFS block group size:

  <<< seeks get longer (extreme: same as original unix FS design) >>>

- A too-small FFS block group size:

  <<< too much seeking between groups for large files >>>

- A too-large LFS segment size:

  <<< too much data buffered in memory before writing out >>>

  <<< OR: garbage collection gets harder >>>

- A too-small LFS segment size:

  <<< reduces sequentiality of writes (the whole point) >>>
Question 9 (6 points)

The SCAN family of disk I/O scheduling algorithms center on the idea of "sweeping" the read/write head across the platter.

The C-SCAN variant only services requests when moving in one direction (e.g. outer to inner, resetting to the outermost tracks upon reaching the innermost tracks of the platter). What advantage does this offer over a plain SCAN that, upon reaching the innermost tracks, services I/O requests along the way while returning to the outer tracks? Briefly explain how/why this works.

<<< Fairness -- doesn’t favor tracks in the middle by passing over them >>>
<<< twice as often. >>>

The F-SCAN variant "freezes" the current request queue at the start of each sweep, deferring acceptance of any new requests until the sweep has completed. What advantage does this offer over a plain SCAN that accepts new requests in its queue at any point in time? Briefly explain how/why this works.

<<< Fairness again -- avoids potential starvation of tracks at the end of >>>
<<< the sweep if a flood of nearby requests keep arriving mid-sweep. >>>

=====================================================================

Question 10 (5 points)

With a hypervisor running virtual machines, there are three main types of code that may be executing on the CPU at a given point in time: guest user code, guest kernel code, and VMM (hypervisor) code. Assuming basic hardware with only two execution modes, privileged and unprivileged, in which execution mode must each type of code run?

- Guest user code: <<< unprivileged >>>

- Guest kernel code: <<< unprivileged >>>

- VMM code: <<< privileged >>>

Why must the guest kernel run in that mode? (What would go wrong if it ran in the other?)

<<< It could take over hardware control from the hypervisor if it ran >>>
<<< in privileged mode. >>>
Question 11 (2 points)

In the "have I repeated this enough" (a.k.a. "free points") department:

When considering the overall design at the outset of a programming project, which is more important to get right: code or data structures?

<<< data structures! >>>
[This nutritious, educational, and detachable page provided for all your scratching needs.]