Rev. 03

Name: _____

Instructions

There are four (4) questions on the exam spread across 8 pages. You may find questions that could have several answers and require an explanation or a justification. As we've said, many answers in storage systems are "It depends!". In these cases, we are more interested in your justification. So make sure you're clear. Good luck!

If you have several calculations leading to a single answer, please place a box around your answer.

Problem 1 – Scheduling Algorithms. [30 points]

(a) Given the system setup you used in Project 1 (i.e, a single host issuing random 4KB requests to a single Quantum Atlas 10k disk) and a high frequency of arrivals (e.g., 300 I/Os per second), why do the CLOOK and SSTF scheduling algorithms perform very similarly?

Both algorithms attempt to schedule requests that will incur the shortest seek distance. CLOOK chooses a request that is closest in the seek distance to the current head position and is located in the "sweeping direction" of the disk heads. SSTF does the same without the restriction on the sweeping direction of the disk heads. With small inter-arrival times and random requests the conceptual difference of the sweeping direction is negligible.

(b) Given the existing storage interface between hosts and storage systems based on the SCSI common architecture model (CAM), can a host operating system implement Shortest-seek Time First (SSTF) scheduling algorithm inside its device driver (i.e., outside the disk firmware)? Justify your answer.

Yes. A device driver can approximate the seek distance by the distance of requests in the LBN space.
Alternative answer: *We also accepted No, if properly justified that the device driver does not know the exact geometry or the mappings of logical blocks to physical locations.*

(c) Why does the Shortest Positioning Time First (SPTF) scheduling algorithm typically outperform the other scheduling algorithms we discussed (for example SSTF and CLOOK)?

SPTF schedules requests that will incur the shortest positioning time as a sum of seek distance and rotational latency. SSTF or CLOOK are insensitive to the rotational latency and can schedule requests that have short seek distance but incur large rotational latencies.

(d) Give two reasons why it is infeasible (if not impossible) to implement the Shortest Positioning Time First (SPTF) scheduling algorithm outside disk firmware, given the existing storage interfaces.

Reason 1: *The device driver would have to know detailed mappings of logical blocks to physical locations as well as the seek profiles for all disks it would operate with.*

Reason 2: *The command overheads caused by the interconnect hardware and protocol can introduce enough error to diminish the effectiveness of the rotationally sensitive scheduling.*

(e) Why do modern disk drives implement Age-sensitive-SPTF (A-SPTF) in their firmware instead of the traditional SPTF you experimented with in Project 1?

Age sensitive scheduling prevents starvation (large response times). Requests that have been queued for longer periods of time are given preference over other requests even if they they incur longer positioning times compared to the majority of the other requests.

(f) One priority-based scheduling scheme uses two queues and assigns priority based on request criticality: one queue is for synchronous requests and the other one for asynchronous ones. For each category (synchronous and asynchronous) give two examples of operating system I/O activities that fall into each category.

	Synchronous	**Asynchronous**
Example 1:	reading a block of a newly opened existing file	prefetching blocks of a large file when sequential access is detected
Example 2:	FS metadata block writes for files opened with `O_SYNC` flag or after an fsync() call	flushing dirty pages/blocks from FS cache to the disk with `bdflush()` thread

Problem 2 – Storage Interconnects. [20 points]

(a) The new Serial ATA (S-ATA) standard defined a new link layer to replace the shared bus design of the Parallel-ATA. The S-ATA link layer uses point-to-point serial links between a controller and a *s*ingle device. Recall from the lecture, that in the "Master-Slave" Parallel-ATA architecture, two storage devices (e.g., disks) are connected to an IDE/ATA controller and hold the parallel bus during the entire execution of a command (i.e., they do not "disconnect" from the bus unlike SCSI devices). List at least three reasons why the S-ATA is "better".

Reason 1: *With point-to-point links, devices do not compete for a shared link, which can cause delays when one device wants to communicate while the other device is holding a bus.*

Reason 2: *Serial links allow transmission over longer distances; cross-talk between individual data signals is eliminated.*

Reason 3: *Serial-ATA protocol offers greater transmission speeds than Parallel ATA.*

(b) Assume an operating system that is running multiple applications sharing the same storage device. Give two reasons how Direct Memory Access (DMA) coupled with operating system's support for scatter/gather (`iovec[]`-based) I/O can be beneficial.

Reason 1: *Smaller command processing overhead: an OS can issue a single I/O operation for data for two (or more) unrelated items when such items e.g., FS blocks are mapped to consecutive logical block numbers.*

Reason 2: *Offloading CPU: with properly set vector, a single I/O can deliver/fetch data to/from non-contiguous memory locations. This is for example useful when coalescing multiple blocks on different memory pages into a single I/O for journal-write in journaling file systems.*

(c) Fibre-channel (FC) uses 8b/10b encoding to create 10-bit symbols from 8-bit characters. List two benefits of the 8b/10b encoding.

Benefit 1: *It allows detection of errors such as bit flips.*

Benefit 2: *It allows for extra control characters that are not part of the original 256-character alphabet (e.g., ASCII). A third benefit is that one chooses a 10-bit encoding of symbols, such that no symbol has a long run of zeros or a long run of ones. Hence, self-clocking on a serial bus is easier to implement.*

(d) Give two reasons why the iSCSI target discovery mechanisms must be much more complicated than in the case of a target discovery in parallel SCSI architecture.

Reason 1: *The number of possible devices that can be discovered is not bound by the precisely defined number of "slots" on the parallel bus.*

Reason 2: *The addresses of the devices may not be known apriori.*

Bonus: Reason 3: *An internet network is generally less reliable: packet drop, congestion, or network partitioning can occur at any time.*

Problem 3 – File system basics. [15 points]

Assume a disk with two partitions, each containing a separate file system. The root file system is an FFS and contains all system binaries and libraries as well as a directory for temporary files, `/tmp`. The XFS file system contains all of the user data.

During the boot sequence, first the root file system is mounted and then the XFS file system is mounted underneath the `/home` directory.

```
[...]
mount -t ffs / /dev/partition1
mount -t xfs /home /dev/partition2
[...]
```

The structure of the resulting file system looks as follows:

```
/
+-- bin
|      ...
|
+-- lib
|      ...
|
+-- tmp
|
+-- home
     |
     +--joe
     |
     +--jim
```

Answer the following file system questions. In explaining your answer, assume a basic FFS implementation and think about the i-node operations performed.

(a) Can you create a symbolic link in your root file system, `/tmp/joes-home`, to a directory, `/home/joe`, in the mounted XFS file system with home directories? Explain your answer.

```
ln -s /home/joe/ /tmp/joes-home
```

Yes. Symbolic link is a special file whose content is the path to the object (i.e., directory or file) the link is referring to. Therefore, the symbolic link can refer to an object in a different file system because the path is traversed element by element.

(b) Now, assume the same setup as before in part (a). Can you create a hard link in the root file system's `/tmp` directory to another directory, `/home/jim`, in the file system? Explain your answer.

No. A hard link has its own i-node that references the i-node of the original file the hardlink is referring to. Since i-nodes and reference counts are specific to each file system implementation, it is not possible to have hard links cross file system boundaries.

(c) Does the Virtual File System (VFS) layer have to understand the structure of "i-nodes" for all file systems it supports? Justify your answer.

No. A VFS's virtual node, v-node, just includes a pointer to the i-node of the file in the respective file system. Once dereferenced, the i-node is accessed by filesystem-specific functions and hence VFS need not understand the structure of the i-nodes of every file system.

(d) The first DOS filesystem supported only filenames with "8+3" filenames (NAME.TYP, where NAME is at most 8 characters, and TYP is at most 3 characters). Furthermore, all filenames were uppercase. If that DOS filesystem is mounted at `/dos`, what would you expect to see when you type:

```
cp /home/joe/index.html /dos/index.html
ls /dos
```

INDEX.HTM or index.html. With proper justification, we accepted both answers. Which is returned depends on whether it is the VFS or the DOS-filesystem that returns the name.

(e) FFS uses blocks and clusters (fixed size extents). Fixed size and variable size extents have each their advantages and disadvantages. What is an advantage of fixed size extents over variable size extents?

There are many possible answers. One of them is that fixed-size extents allow very compact structures, i.e., bitmaps, for keeping track of which extents are allocated and which ones are not. Since the size of each extent is the same, simply checking the bitmap will reveal how much space is allocated. With variable extent-size, on the other hand, more complicated structures (e.g, B+tree with extents sorted by size as is the case in XFS), are needed to determine which extent(s) can be used to allocate a file.

Problem 4 – Building Filesystems with Consistent Metadata. [35 points]

Ron Hacker wants to build a "best-of-breed" file system that combines various features of the different systems we studied in the class. He models his system after the XFS (i.e., extent-based allocation, B+ trees, journaling, etc.) and wants to build his improvements on top it. His initial prototype uses a fixed-size journal of 32 MB located at the beginning of his filesystem.

However, he is not quite sure how to ensure that he can make heads or tails of the various metadata-integrity-ensuring techniques so he hires you as a consultant.

(a) His first improvement is to cache all of the data and metadata of a newly created file and flush them both to the disk only when the file is closed. He claims that with metadata-only journaling mode and large (volatile) memory used as a cache, he can ensure data integrity and good performance for creating and writing many large files. Is he correct? Justify your answer.

Yes. Using metadata-only journaling mode and writing data in place only when the file is closed allows for a newly created file to be written out sequentially and yields good throughput since data are written only once and metadata are written to nearby locations in the journal. It can also potentially eliminate extraneous writes when a file is updated frequently. However, it could also be argued that data integrity was not maintained, since the newly created file was not written out until the close. If the system crashes before the close, the data is lost. This answer was also accepted.

(b) Outline the steps that his filesystem must do to ensure integrity after an application's invocation of the `close()` system call.

It must (i) allocate blocks (extents) for the data (update allocation structures), (ii) create a new i-node and update an i-node block, (iii) insert a new entry into the directory entries B+tree, (iv) possibly update the i-node of the directory with new access/modification date. Recall that the data blocks are already in memory. So allocating blocks means assigning the in-memory blocks to extents that are free. Note also taht the ordering of the operations is dictated by "soft updates".

(c) What is the minimal number of I/O requests that the operating system must issue in order to create and write a single large file using Ron Hacker's new filesystem? Justify your answer. Assume that the log has been just checkpointed and that he can allocate the entire file into a single extent.

Two. Assuming a file fits into a single extent, then one I/O is for writing out the data to the disk and the other I/O is for all operations to the journal. Note that writing the operations (i), (ii), (iii) and (iv) together with the commit record can be coalesced into a single I/O operation.

(d) He is paranoid about data consistency, so he decides to use data journaling in addition to metadata journaling. However, to his dismay, he observes that his large-file (10MB+) write performance suffers. Explain to him what is going on.

With data journaling mode, the data is being written twice to the disk - once into the journal and second time in place onto the disk when the journal is checkpointed. Thus, he would observe performance that is approximately half of what he observed for large file writes in his original design.

After he understands what's going on, he is content to take the performance hit and to stick with both data and metadata journaling; he wants to make his filesystem a commercial success and thus wants to ensure that data is not lost. So he comes back with a different idea.

(e) He wants to implement a mixed strategy whereby he uses soft updates for writing large files and journaling mode (for both data and metadata) when writing small files. Explain to him why this strategy is not going to work. (Recall that soft updates consist of global ordering while journaling mode uses write-ahead logging.)

There are several reasons. Since soft-updates modifies meta-data in place (in disk blocks outside the journal), while journaling mode writes updates first into the journal before checkpointing them in place. Now suppose one is writing a large file. Without any additional structures, it is impossible to do recovery since soft-updates could update data in-place by writing blocks outside the journal. This renders the in-journal copy invalid.

(f) Since he is not really happy what he hears from you, he takes another approach for his startup and settles for much less. He wants to device a new way to improve the performance of ext2 (which is based on the Berkeley Fast File System). He wants to add a ticket number (also called a sequence number or nonce) to every metadata block written, and increment the ticket number by one when *each* successive block is written. He claims that this will speed up his recovery (`fsck`) after a crash because he will know exactly when his metadata updates failed and thus just repair those instead of having to crawl through every directory and file's metadata. Is he correct?

No. Since a single metadata update operation typically touches multiple structures, it will be impossible to determine the state after a crash if, for example, two update operations that do not have dependencies were executing concurrently when the crash occurred.

(g) In his desperate attempt to have some viable product, he wants to ditch his XFS prototype as well as his plans to improve ext2 and use LFS instead to penetrate into a new application domain. His new application generates a read-mostly workload that uses lots of large files that grow only slowly over time. What performance problems would you expect to observe? Then suggest your own patch to LFS to fix this problem.

Over time, the blocks of a single file will be spread across many segments. Reading a whole file will then result in random accesses with poor performance. The patch should ensure that blocks of the same file are in the same segment. One possibility to ensure this is to keep multiple segments in memory, instead of just one, and have each segment collect data from one files.