University of California, Berkeley
College of Engineering
Computer Science Division — EECS

Spring 2024 John Kubiatowicz

Midterm III
SOLUTION

April 25th, 2024

CS162: Operating Systems and Systems Programming

Your Name:	
Your SID:	
TA Name:	
Discussion Section Time:	

General Information:
This is a **closed book** exam. You are allowed 3 pages of notes (both sides). You have 2 hours to complete as much of the exam as possible. Make sure to read all of the questions first, as some of the questions are substantially more time consuming. Write all of your answers directly on this paper. *Make your answers as concise as possible.* On programming questions, we will be looking for performance as well as correctness, so think through your answers carefully. If there is something about the questions that you believe is open to interpretation, please ask us about it!

Problem	**Possible**	**Score**
1	18	
2	20	
3	23	
4	17	
5	22	
Total	100	

[This page left for π]

3.14159265358979323846264338327950288419716939937510582097494459230781640628620899

Problem 1: True/False [18 pts]

Please *EXPLAIN* your answer in TWO SENTENCES OR LESS (Answers longer than this may not get credit!). Also, answers without an explanation *GET NO CREDIT.*

Problem 1a[2pts]: It is possible to make Remote Procedure Calls (RPCs) with integer arguments between clients that use big-endian integers and servers that use little-endian integers.

[✓ True] ☐ False

Explain: *An important aspect of RPC is the fact that arguments are marshalled into a standardized format at the source and unmarshalled at the destination. Thus, integers will be converted to and from a network standard ordering into the local ordering on each end.*

Problem 1b[2pts]: In the Pintos kernel, if you call intr_disable() and then dereference a NULL pointer, your kernel will be stuck in an infinite loop, because the page fault trap cannot reach the CPU when interrupts are disabled.

☐ True [✓ False]

Explain: *Synchronous traps, such as page-faults, cannot be disabled by* `intr_disable()`.

Problem 1c[2pts]: The Clock algorithm is used to find a physical page that has not been accessed in a while, thereby approximating a MIN replacement algorithm.

[✓ True] ☐ False

Explain: *Since the Clock algorithm provides an approximation to LRU, which provides an approximation to MIN, the Clock algorithm approximates MIN. Another way to look at this, is that the Clock algorithm finds old pages to replace, which are likely to be accessed further in the future than recently access pages.*

Problem 1d[2pts]: The FAT file system links together blocks in a file by including a pointer in each block on disk to the next block on disk. In a FAT32 file system, this means that each block stores 4 bytes less of user data than the size of the block on disk.

☐ True [✓ False]

Explain: *The FAT file system does not link blocks together with pointers in the data blocks. Instead, it collects all of the pointers into the FAT data structure.*

Problem 1e[2pts]: An M/M/1 queue has a deterministic distribution for both arrival times and service times.

☐ True [✓ False]

Explain: *An M/M/1 queue has a memoryless (exponential) distribution for both arrival times and service times.*

Problem 1f[2pts]: A storage system protected by RAID6 can identify up to two faulty drives by looking at the data returned from reads.

☐ True [✓ False]

Explain: *RAID6 is an erasure code. Consequently, it can only recover data from failed drives after they have been identified; some other mechanism must first identify drives as failed (such as error codes on the drives themselves).*

Problem 1g[2pts]: Journaling file systems use a log to make multi-operation updates atomic in the face of OS crashes.

[✓ True] ☐ False

Explain: *These file systems write operations to the log first, without changing the actual file system. Only after all of the operations have been made durable in the log does the file system write a final "commit" record, thereby making sure that partial updates will be ignored if the OS crashes before the "commit" is written.*

Problem 1h[2pts]: For modern disk drives, the transfer rate for sectors on outer cylinders is higher than for sectors on inner cylinders.

[✓ True] ☐ False

Explain: *Since the bit density is constant on the surface of a platter and the disk rotation speed is constant, more bits per unit time go under the read heads when they are on the outer cylinders than when they are on the inner cylinders. Consequently, the transfer rate (in bits/second) is higher on the outer cylinders than on the inner cylinders.*

Problem 1i[2pts]: SSDs utilize FLASH memory which wears out as it is written.

[✓ True] ☐ False

Explain: *As FLASH cells are repeatedly written and erased, charges get stuck in the insulators, thereby compromising the ability of these cells to differentiate between a 0 and a 1.*

Problem 2: Multiple Choice [20pts]

Problem 2a[2pts]: Which of the following best describes the source of the Meltdown security flaw (made public in 2018)? (*Choose one):*

A: ◯ A system call implemented by multiple operating systems neglected to properly check arguments and could thus be fooled into returning the contents of protected memory to users.

B: [✓ ⊗] User code could execute speculative loads to addresses marked as "kernel-only" in a user's page table and use the result to effect the state of the cache before the load results could be squashed by the processor pipeline.

C: ◯ Unmapped physical pages (i.e. that were not in an page table) could be accessed by user-level code because of a timing bug in the x86 processor between TLB lookup and cache access.

D: ◯ Linux version 2.4 had a stack-overflow bug which would allow it to be tricked into mapping kernel pages into the user page table.

E: ◯ Speculative execution of arithmetic operations on out-of-order pipelines could be made to reveal the contents of kernel memory by tricking the kernel into changing the timing of operations based on the contents of its private data.

B: TRUE. The Meltdown flaw involved speculative out-of-order execution, in which speculative operations could be made to impact a cache timing channel. E is FALSE because the Meltdown flaw involved loads and stores to protected kernel memory.

Problem 2b[2pts]: Little's Law has the following properties (*Choose all that apply):*

A: ☐ It applies only to memoryless arrival distributions.

B: [✓ ☑] It says that the average number of jobs in the system is equal to the average arrival rate of jobs multiplied by the average length of time that a job stays in the system.

C: ☐ It explains why the average length of time spent in a queue grows without bound as the system utilization approaches 100%.

D: ☐ It can be applied to systems that are not in equilibrium.

E: ☐ None of the above.

A: FALSE. Little's Law is correct regardless of the distribution of arrivals.
B: TRUE. This is literally a statement of Littles law in words (which is $N = \lambda T$).
C: FALSE. Little's Law doesn't explain why queues grow in length, merely allows you to compute the length of a queue, given the average wait time.
D: FALSE. Little's Law is only for systems in equilibrium.
E: FALSE. Obviously.

Problem 2c[2pts]: Memory-mapped I/O is *(Choose one)*:

A: ○ A security mechanism for I/O devices that that prevents user-mode applications from directly accessing these devices, forcing device access to go through the system call interface.

B: ○ A software protocol that constructs a coherent distributed shared memory between multiple physical nodes, allowing communication between nodes to occur as reads and writes to the shared memory (address) space.

C: ○ A technique for communication between processes using the mmap() system call. As a result, processes can interact via reads and writes to shared memory addresses

D: [✓ ⊗] A hardware mechanism for assigning physical memory addresses to devices such that processor read and write operations to these addresses become commands to devices.

E: ○ None of the above.

D: TRUE. As discussed multiple times in class, memory-mapped I/O is an alternative to port-mapped I/O as a way for the processor to access device controllers. Note that C is false because the `mmap()` system call is for mapping memory addresses to files, but this is not typically called "memory-mapped I/O" (and D is more correct).

Problem 2d[2pts]: Which of the following are true regarding buffer caches? *(Choose all that apply)*:

A: [✓] In file systems without a journal, delayed writes can allow temporary files to be created, written, read, and deleted without ever impacting the disk.

B: ☐ Since FIFO is an approximation to MIN, we can easily handle buffer cache replacement by linking all buffer cache pages into a circular list.

C: [✓] The buffer cache can improve file system performance in situations in which multiple processes simultaneously write to random parts of the disk.

D: [✓] The buffer cache can participate in the prevention of compulsory misses during sequential reads from large files.

E: ☐ The buffer cache is a write-through cache.

A: TRUE. Until they are flushed to disk, file system modifications are kept in the buffer cash. This includes creation and deletion of files.
B: FALSE. FIFO replacement makes no attempt to choose pages based on expected future usage, so cannot be an approximation to MIN.
C: TRUE. When multiple processes write to files, updated disk blocks are likely to be randomly distributed over the disk. By placing writes in the buffer cache, the OS can reorder blocks to better reflect disk-level locality (e.g. using the elevator algorithm), thus improving performance.
D: TRUE. The buffer cache provides an ideal place for prefetched blocks, since they can be easily found by reads and writes.
E: FALSE. Write-through to disk would be a very bad idea (millions of instructions/write). In fact, the buffer cache provides a write-back cache.

Problem 2e[2pts]: Which of the following are true about the Pintos file system? *(Choose all that apply)*:

A: ☐ File names for a given directory are stored as strings inside the directory's inode.

B: [✓] A directory can only hold up to a maximum number of directory entries.

C: ☐ In order to resolve a relative path passed into the open syscall, the root inode sector has to be read from the disk.

D: [✓] A directory entry always corresponds to an inumber on the disk.

E: ☐ The directory path "/../../" is invalid.

A: FALSE. Filenames are stored in datablocks, not the inode (which points at blocks).
B: TRUE. Since Pintos has a maximum file size, this means that directories (which are just files) have a maximum number of directory entries that they can hold (limited by maximum directory size).
C: FALSE. If the current working directory (CWD) has already been used, then the resolution of a relative path could start there, rather than with the root directory.
D: TRUE. Since directories are just files, they have inodes just like files. Inodes have an associated inumber on the disk.
E: FALSE. In the root directory, ".." just resolves to the root (like "."). So, "/../../" is just another name for the root directory.

Problem 2f[2pts]: Which of the following statements about I/O performance are true? *(Choose all that apply):*

A: [✓] The elevator algorithm can improve hard disk drive performance by handling I/O requests in order of physical location rather than in order of arrival.
B: [✓] For many I/O devices, the effective bandwidth of a request can be improved by increasing the size of the request (in bytes).
C: [✓] If the requests entering a queue are combined from many different (uncorrelated) sources, then the arrival distribution can be roughly modeled by an exponential (memoryless) distribution.
D: ☐ SSDs have a seek time that is higher than the time to perform a write.
E: ☐ None of the above.

A: TRUE. This is literally a description of what the elevator algorithm does.
B: TRUE. Assume the time to perform an access can be expressed with a simple linear equation such as: $T = T_o + \frac{S}{B_R}$, where T_o is overhead and B_R is the raw bandwidth, then effective bandwidth is just: $B_E = \frac{S}{T} = \left(\frac{S}{T_o \times B_R + S}\right) \times B_R$, which grows (increases) toward B_R as S grows.
C: TRUE. This is the basic premise for using a memoryless distribution on arrival times in the queueing equations given in class.
D: FALSE. SSDs don't have seek times.
E: FALSE. For obvious reasons.

Problem 2g[2pts]: In UNIX, which of the following I/O devices can be accessed using file operations like `open()` and `close()`? *(Choose all that apply):*

A: [✓] Network devices (WiFi, Bluetooth, wired Ethernet).
B: [✓] A mouse.
C: [✓] The internal hard drive.
D: [✓] A serial device connecting to a terminal.
E: [✓] A printer.

A-E: TRUE. UNIX provides an "everything is a file" API. In addition to the normal I/O access interfaces, such as Socket I/O and File I/O (which explains A and C being true), all I/O devices have raw device access files in the /dev directory and can be accessed with `open()` and `close()`.

Problem 2h[2pts]: Which of the following are true about modern hard disks? (*Mark all that apply):*

A: ☐ They have an independent disk head on every surface of every platter. As a result, the disk can simultaneously read or write from different tracks on different platters.

B: ☑ Some of them gain bit density by overlapping tracks.

C: ☑ Their internal controllers contain memory for caching, allowing them to read a whole track at a time.

D: ☐ They have a lower bit density on the outside tracks from the inside tracks (because the surface of the outside tracks move under the disk head faster than that of the inside tracks).

E: ☑ Their internal controllers can queue requests and perform variants of the elevator algorithm without consulting the operating system.

A: FALSE. Disk heads are locked together on a single disk arm. Consequently, the disk heads are not independent. Without moving the disk arm, they need to read from/write to the same track on different platters (i.e. the same cylinder). As a secondary reason for this to be false, the DSP processors are typically able to read from only a single head on a single surface at a time.

B: TRUE. Disks that use SMR ("Shingled Magnetic Recording") write data in a pattern that resembles shingles on a roof: each write overlaps an adjacent track.

C: TRUE. Disks have plenty of RAM on the controllers for caching (just look at disk specs from, say Seagate).

D: FALSE. Bit density (bits/square inch) is constant across the writeable surface of the disk. The reason for this is that maximum bit density is a property of the magnetic material and recording technique. As an asside, since the rotational rate is constant independent of disk head position, this means that the read/write rate is higher on the outside tracks than it is on the inside tracks.

E: TRUE. Disk controllers have internal queues and perform their own optimization of accesses.

Problem 2i[2pts]: The Berkeley FFS has the following properties (*Mark all that apply*):

A: ☐ It changed the inode format from the BSD 4.1 file system to better reflect the overwhelming presence of small files in a typical UNIX filesystem.

B: ☑ It reserved 10% of the blocks to ensure that new files could get sequential groups of blocks for better read performance.

C: ☑ It performed skip-sector allocation of blocks to prevent processor delays during reading from missing blocks and forcing a complete rotation for each block read.

D: ☑ It placed the inodes and blocks for files within a directory close to the blocks and inodes for the directory to improve performance.

E: ☐ It introduced a B-tree format for directories in order get better lookup performance for directories with large numbers of files.

A: FALSE. The FFS (BSD 4.2) kept the same format for inodes as BSD 4.1.

B: TRUE. As discussed in class.

C: TRUE. As discussed in class.

D: TRUE. As discussed in class.

E: FALSE. Directories were still formatted in a linear, unsorted fashion.

Problem 2j[2pts]: Which of the following are true about the Clock Algorithm? *(Choose all that apply):*

A: ☑ To examine the state of a particular *physical* page and thus determine if it should be replaced, the Clock Algorithm must examine all PTEs (in all of the page tables) that point at the physical page.

B: ☐ The Clock Algorithm arranges *physical* pages in a ring and keeps a pointer (clock hand) to a page within the ring. During a page fault, it uses the first physical page pointed at by the clock hand to hold incoming data, after which it advances the clock hand to the next page.

C: ☑ The Clock Algorithm arranges *physical* pages in a ring and keeps a pointer (clock hand) to a page within the ring. During a page fault, it chooses a page to hold incoming data by looking at pages one at a time, starting with the current clock hand, until it finds one that hasn't been used recently.

D: ☑ The Clock Algorithm provides an approximation to LRU.

E: ☐ The Clock Algorithm arranges *physical* pages into two groups – an active group that is mapped as "valid" and managed in a FIFO list and an inactive group that is mapped as "invalid" and managed as an LRU list.

A: TRUE. Since the clock algorithm must figure out if the physical page under the clock hand has been touched ("used") recently, it must hunt down the PTEs pointing at the page to see if any of them have their "Use" bit set. .

B: FALSE. This description would be true if we were trying to implement a FIFO replacement policy. However, it is not what the clock algorithm does.

C: TRUE. This is a valid description of what the clock algorithm does. Note that this answer differs from B in that it describes walking through pages under the clock hand until it finds one that hasn't been used recently – rather than grabbing the first page.

D: TRUE. This is true because the clock algorithm finds "an old page", which is an approximation to LRU.

E: FALSE. This description is for the "Second Chance" algorithm, not the clock algorithm

[This Page Intentionally Left Blank]

Problem 3: Potpourri [23pts]

Problem 3a[2pts]: In Pintos, what is the significance of the PHYS_BASE constant? *No more than two sentence answer.*

This is the beginning of kernel virtual memory (i.e. is above the maximum user virtual address). The virtual address at PHYS_BASE maps physical page 0 (i.e. the base of physical memory).

Problem 3b[3pts]: Why is it important for the page fault exception to be precise? Make sure that you define "precise exception" in your answer. *No more than 2 sentences for the definition and one sentence for why the page fault exception needs to be precise.*

A precise exception is one which has a well-defined instruction in the execution stream for which the processor state is as if (1) all prior instructions have finished and committed their results to registers/memory and (2) the given instruction and all following instructions have not started nor committed results.

If the page fault exception is precise, it is easy for the operating system to restart a user program after a page fault.

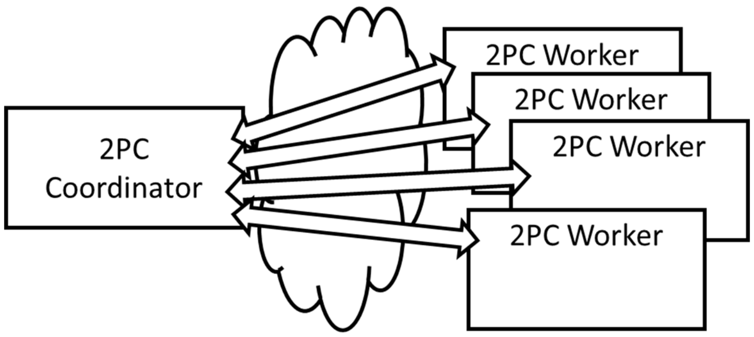
Problem 3c[8pts]: For the following problem, assume a hypothetical machine with 4 pages of physical memory and 7 pages of virtual memory. Given the access pattern:

```
A B C D E A A E C F F G A C G D C F
```

Indicate in the following table which pages are mapped to which physical pages for each of the following policies. Assume that a blank box matches the element to the left. We have given the FIFO policy as an example. Complete LRU and CLOCK. For CLOCK, assume that we check the page under the clock hand, then increment; also, assume that pages brought in have Use=0 initially.

Access→		**A**	B	C	D	E	A	A	E	C	F	F	G	A	C	G	D	C	F
FIFO	1	A				E									C				
	2		B				A										D		
	3			C							F								
	4				D								G						
LRU	1	*A*				*E*								*A*					*F*
	2		*B*				*A*						*G*						
	3			*C*															
	4				*D*						*F*						*D*		
CLOCK	1	*A*				*E*			*1*				*0*		*C*			*1*	*0*
	2		*B*				*A*	*1*					*0*	*1*			*0*		*F*
	3			*C*						*1*	*0*		*G*			*1*	*0*		
	4				*D*						*F*	*1*			*0*		*D*		

Hint on solving CLOCK problem above: The blue numbers represent the value of the Use bit for a particular page. You were not required to include them. When page brought in, Use=0, so a letter implicitly includes a blue 0. When a page is referenced, we set Use=1. Further, if we are looking for new page but Use=1, we set to zero and advance clock hand.



Two-Phase Commit examples: In the following scenarios, consider the following responses to a failure condition. Answer if it renders the system SAFE or UNSAFE and explain why:

Problem 3d[2pts]: The Coordinator sends a *VOTE-REQ* to the Workers and fails immediately. After recovering, it sends a *GLOBAL-ABORT* to all Workers.

[✓ SAFE] ☐ UNSAFE

Explain: *It is always safe for the coordinator to choose to abort, as long as it hasn't sent commit messages yet. In this instance, abort is the right thing, since the coordinator has no idea whether workers have responded, much less what they may have responded.*

Problem 3e[2pts]: Suppose a Worker sends a vote to *VOTE-COMMIT* to the Coordinator and does not hear back a response. After some timeout period, this means that the Coordinator lost its vote, so the Worker aborts.

☐ SAFE [✓ UNSAFE]

Explain: *The communication channel between the Coordinator and the worker may just be slow, and all other nodes received a decision to COMMIT already.*

Problem 3f[2pts]: Suppose the Coordinator had a 30 second timeout frame. A Worker logs their *VOTE-COMMIT* and before it can send its vote, it crashes and recovers within 2 seconds. This Worker sends its vote to the Coordinator and crashes and recovers again. Then this Worker sends its vote to the Coordinator for the second time.

[✓ SAFE] ☐ UNSAFE

Explain: *The worker is making sure that its vote makes it to the coordinator. Since it is just repeating what it has already voted, it is not violating any semantics for 2PC.*

Problem 3g[2pts]: Explain why RAID5 is no longer considered sufficient protection for storage systems with large hard disk drives. *No more than two sentence answer.*

RAID 5 is only able to tolerate the failure of a single disk. Modern hard disk drives are so large that it is possible that a second disk fails during the (long) process of recovering from a first disk failure.

Problem 3h[2pts]: List two reasons that SSDs must have a Flash Translation Layer (FTL). *One sentence per reason.*

There are a number of reasons for this. Here are three:

1. *The FTL is necessary to provide a layer of indirection between the block numbering of the OS and the physical identities of individual pages inside the FLASH.*
2. *The FTL is necessary to track the frequency of writes to FLASH pages to avoid wearing them out prematurely (this is called wear-levelling).*
3. *The FTL is necessary to garbage collect FLASH blocks by moving live data into other pages so that empty blocks can be erased.*

[This Page Intentionally Left Blank]

Problem 4: Pintos/File Systems [17pts]

Consider the following inode_disk and indirect_block struct similar to what is in Pintos. Suppose sectors are 512 bytes.

```
struct inode_disk {
    block_sector_t direct[12];
    block_sector_t indirect;
    block_sector_t triply_indirect; // Note that this is a triply
                                    // and that there is no doubly!
    size_t size;
    void* unused[113];
};

struct indirect_block { // doubly and triply indirect blocks look like this
    block_sector_t block_nums[128];
}
```

Problem 4a[2pts]: What is the maximum file size supported by this design? Leave your answer unsimplified in the box:

$512 \times (12+128+128\times128\times128)$

Problem 4b[4pts]: How many sectors of each type are required to represent a maximum sized file in this design? Leave your answer unsimplified in the box:

Data blocks:

$12+128+128\times128\times128$

Indirect blocks:

$1+128\times128$

Doubly indirect blocks:

128

Triply indirect blocks:

1

Problem 4c[6pts]: Suppose that we have a file that is represented using the inode structures of (4a) and (4b) and that the current size of the file is 12 * 512 bytes long. We now want to write 512 characters to the end of this file. You may use the following functions:

```
void block_read(void *buffer, block_sector_t block_num);
void block_write(void *buffer, block_sector_t block_num);
block_sector_t block_allocate();
```

The block sector corresponding to the file's inode_disk struct is 3:

```
block_sector_t FILE_INODE_BLOCK = 3;
```

Fill in the blanks in the following function such that after running this function, 512 characters are written to this file and persisted. Assume that the file is exactly 12 blocks long already. You may not need all of the blanks, but may only have 1 semicolon per line.

```
void Append512Chars () {
   struct inode_disk inode;

   block_read(&inode, FILE_INODE_BLOCK);  // Read inode from disk

   inode.indirect = block_allocate(); // Allocate new block for indirect

   struct indirect_block indirect_block;
   indirect_block.block_nums[0] = block_allocate();
   char buffer[512];
   memset(buffer, 'a', 512);

   inode.size += 512; // Change the size

   block_write(buffer, indirect_block.block_nums[0]); // Persist data

   block_write(&indirect_block, inode.indirect); // Persist indirect

   block_write(&inode, FILE_INODE_BLOCK); // Persist inode
}
```

Problem 4d[2pts]: Explain what the mmap() system call does and how it can be used to enable fast communication between two processes. *No more than two sentence answer.*

The mmap() system call is used to map a file into a region of a process' address space such that reads and writes to addresses in that region translate into reads and writes to the file. If two different processes mmap() the same file into their separate address spaces, then they will be able exchange information through shared memory (i.e. memory writes in one process will show up in the other one).

Problem 4e[3pts]: Rather than writing updated files to disk immediately when they are closed, many UNIX systems use a delayed *write-behind policy* in which dirty disk blocks are flushed to disk once every 30 seconds. List two advantages and one disadvantage of such a scheme: *Only one sentence for each category:*

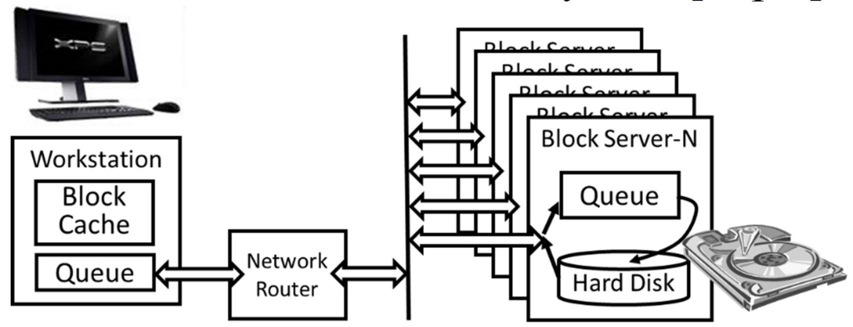
Advantage 1: *Writes can be merged/rearranged for better disk performance (e.g. for elevator scheduling).*

Advantage 2: *Temporary files can be created/deleted without ever going to disk.*

Disadvantage: *Data can be lost if system crashes before data pushed to disk.*

[This Page Intentionally Left Blank]

Problem 5: Network Disk System [22pts]



In this problem, we are going to build a network storage system as shown above. This system is going to serve storage blocks over the network to the workstation. This is a so-called "iSCSI" system, in which the client sends requests over the network to remote disks using the iSCSI protocol. The iSCSI protocol takes the normal SCSI commands that would be sent to a disk and "wraps" them with TCP/IP and iSCSI overhead to transport over the network.

The workstation will be responsible for building the file system out of blocks that it accesses. Consequently, the workstation will contain the block cache for this file system. Rather than requesting blocks from a local set of disks, it will send requests over the network to a set of block servers, each of which has its own queue and disk.

Properties of the network are as follows:

- Network bandwidth: **1Gb/s.** Assume that transmission errors do not occur.
- Maximum packet size in network (MTU): **9KB** (so-called "jumbo frames")
- Overhead for packets: **40B** (this is the TCP/IP packet overhead) + **60B** (iSCSI protocol). So, assume that every iSCSI packet is **100B** + data size. *Request packets have no data.*
- The router is fully pipelined and can forward packets at network speed with a **2.2μs** delay.

Properties of the disks on the block servers are as follows:

- There are **10** servers. Each disk is **16TiB** in size with a **4096B** sector size.
- Disks rotate at **10,000 RPM**, have a data transfer rate of **65,536 KB/s**, and have a **5.8ms** average seek time. They also have a SCSI interface with a **200μs** controller time. Assume that a group of consecutive sectors can be fetched with a single request.
- Our use of the disks to build a file system has a disk service distribution with **C=1.5**.
- Each disk can handle only one request at a time, but each disk in the system can be handling a different request.

EACH OF THE FOLLOWING ANSWERS SHOULD BE SIMPLIFIED TO SINGLE NUMBERS. HOWEVER, YOU MUST SHOW YOUR WORK TO GET CREDIT.

Problem 5a[3pts]: Suppose that the server takes **1μs** to receive packets after all the bits have arrived (this is the interrupt service time). What is the latency for a request to make it from the client to one of the servers? *Hint: for network piece, since router is pipelined, compute transmission time for request (iSCSI packet without data) along one network hop, then simply add router delay.*

$$T_{NET} = T_{Wire} + T_{Router} + T_{INT} = \left(\frac{100B\times 8\frac{b}{B}}{10^9\frac{b}{s}\times 10^{-6}\frac{s}{\mu s}}\right) + 2.2\mu s + 1\mu s = \boxed{4\mu s}$$

Problem 5b[3pts]: Suppose that the network controller on clients takes **1μs** to receive packets (this is the interrupt service time). Also, suppose that the disk controller on servers is capable of DMA directly into the network. How long will it take to send a sector worth of data from server to client after the disk controller retrieves the data from disk? *Hint: don't forget the iSCSI packet overhead.*

The only difference here with 5a is that we use 100B+4096B=4196B in the T_{Wire} term:

$$T_{NET} = T_{Wire} + T_{Router} + T_{INT} = \left(\frac{4196B \times 8\frac{b}{B}}{10^9\frac{b}{s} \times 10^{-6}\frac{s}{\mu s}}\right) + 2.2\mu s + 1\mu s = 4\mu s + 32.768\mu s$$

$$= \boxed{36.768\mu s}$$

Problem 5c[4pts]: What is the average *service time* to retrieve a *single* sector from a random location on a *single* disk, assuming no queuing time (i.e. the unloaded request time)? *Hint: there are four terms in this service time! Note: 4096 x 16 = 65,536 and 1/16=0.0625*

$$T_{Ser} = T_{Ctrl} + T_{Seek} + T_{Rot} + T_{Xfer} =$$

$$= \left(200\mu s \times 10^{-3}\frac{ms}{\mu s}\right) + 5.8ms + \frac{1}{2}R\left(\frac{60000\frac{ms}{Min}}{10000\frac{R}{Min}}\right) + \left(\frac{4096B}{65536 \times 10^3\frac{B}{s} \times 10^{-3}\frac{s}{ms}}\right)$$

$$= 0.2ms + 5.8ms + 3ms + 0.0625ms = 9ms + 0.0625ms = \boxed{9.0625ms}$$

Problem 5d[2pts]: How many sequential sectors would we have to combine together into a block in order to achieve an *effective transfer rate* of at least 10% of the data transfer rate when reading a block of data?

Assume we have N sectors in a block. Then, from 5c, our time to read a block (including overhead) would be: $T_{Ser} = 9ms + N \times 0.0625ms$

Effective Transfer Rate $= \frac{Bytes\ in\ a\ block}{T_{Ser}} = \frac{N \times 4096B}{9ms + N \times 0.0625ms} = 0.1 \times \left(65536 \times 10^3\frac{B}{s} \times 10^{-3}\frac{s}{ms}\right)$

Rearranging: $\frac{N \times 0.0625}{9ms + N \times 0.0625ms} = 0.1 \Rightarrow \boxed{N=16}$

Note: you could have done this simply by saying: $\frac{Time\ without\ overhead}{Time\ with\ overhead} = \frac{N \times 0.0625ms}{9ms + N \times 0.0625ms} = 0.1$

Problem 5e[3pts]: Assuming that we arrange to keep all 10 of our servers busy with large requests, such as in **(5d)**, what fraction of our network bandwidth will be used for responses? You can assume that the protocol will never split a sector across network packets. Express your answer as a fraction. *Hint: consider using the effective transfer rate from* ***(5d)****. Also, compute number of complete sectors that will fit into a jumbo packet and account for the iSCSI packet overhead as a fractional increase to data bytes sent.*

This problem simply requires you to figure out how much overhead is introduced in the network by the iSCSI, then multiply by the data bandwidth (in bits!) coming off all 10 disks. Since a sector is 4096B, we can fit 2 of them into a jumbo packet. Such a packet would have an extra 100B of iSCSI packet overhead. Thus, iSCSI blows up data BW by $\frac{4096B \times 2 + 100B}{4096B \times 2} = \frac{8292}{8192}$

$$\frac{EffRate\frac{B/s}{Disk} \times 8\frac{b}{B} \times 10Disks \times \left(\frac{8292}{8192}\right)}{10^9\frac{b}{s}} = (65536 \times 10^3 \times 0.1) \times 8 \times 10 \times \frac{8292}{8192} \times 10^{-9}$$

$$= \boxed{\frac{65536 \times 8 \times 8292 \times 10^{-6}}{8192}} = 64 \times 8292 \times 10^{-6} = 0.530688$$

Problem 5f[4pts]: Suppose that the distribution of arrivals is memoryless. Assume that we choose to use the large block size from **(5d)** and do not want add more than 75% latency because of queueing time. Also assume that the combination of file system layout, access behavior, and disk characterization leads to a coefficient of variance **C=1.5**. What is our target utilization on each node? How many requests will be queued on average in each node (leave as a fraction)? *Hint: Little's law might be helpful here, as would be $u=\lambda T_{ser}$*

$$T_q = T_{Ser} \times \frac{1}{2}(1+C) \times \frac{u}{1-u} \quad \Rightarrow \quad 75\% = \frac{3}{4} = \left[\frac{T_q}{T_{Ser}} = \frac{1}{2}(1+C) \times \frac{u}{1-u}\right] = \frac{2.5}{2} \times \frac{u}{1-u}$$

$$\frac{6}{10} = \frac{u}{1-u} \Rightarrow \boxed{u = \frac{3}{8}}$$

$$L_q = \lambda T_q = \lambda\left(T_{Ser} \times \frac{1}{2}(1+C) \times \frac{u}{1-u}\right) = \frac{1}{2}(1+C) \times \frac{u^2}{1-u} = \frac{5}{4} \times \frac{\left(\frac{3}{8}\right)^2}{\left(1-\frac{3}{8}\right)} \Rightarrow \boxed{L_q = \frac{9}{32}}$$

Problem 5g[3pts]: Assume that we chose to use the large block size from **(5d)** and keep the latency increase due to queueing to be 75%, as mentioned in **(5f)** and that we assume data is stored using RAID6 (which means that 2 of the 10 disks are used for redundancy), what is the bandwidth of actual data we can get from our system? You can leave this result as a product of integers.

This computation is just like (5e) except that we are only looking at data of reduced utilization (not 100%, but rather 3/8 utilization) with no packet overhead and from 8 disks (at any given time, only 8 disks have actual data, since the others have parity data for RAID6). We will also leave the bandwidth in B/s, although b/s could be accepted as well:

$$Answer = (65536 \times 10^3 \times 0.1)\frac{B/s}{Disk} \times \frac{3}{8} \times 8 Disks = \boxed{6553600 \times 3\frac{B}{s}} \equiv 6553600 \times 24\frac{b}{s}$$

An alternate way to get the same answer would be to compute λ as a way to find out number of Blocks transferred per time: $u = \lambda T_{Ser} \Rightarrow \lambda = \frac{\frac{3}{8}}{10ms} \times 1000\frac{ms}{s} = \frac{300}{8}s^{-1}$

$$Answer = \lambda \times BlockSize \times 8 = \frac{300}{8}s^{-1} \times (4096B \times 16)Disk^{-1} \times 8Disks = 6553600 \times 3\frac{B}{s}$$

[This Page Intentionally Left Blank]

[Scratch Page: Do not put answers here!]

[Scratch Page: Do not put answers here!]