CS111 Final (part 1)

[redacted]

TOTAL POINTS

96 / 100

QUESTION 1

Necessary Conditions for Deadlock 10 pts

1.1 4 necessary conditions **2 / 2**

✓ **- 0 Correct**

1.2 1st condition and attack **2 / 2**

✓ **- 0 Correct**

1.3 2nd condition and attack **2 / 2**

✓ **- 0 Correct**

1.4 3rd condition and attack **2 / 2**

✓ **- 0 Correct**

1.5 4th condition and attack **2 / 2**

✓ **- 0 Correct**

QUESTION 2

Write-Through vs Write-Back caches 10 pts

2.1 What write-through optimizes **2 / 2**

✓ **- 0 Correct**

2.2 Why we must write-through **2 / 2**

✓ **- 0 Correct**

2.3 Write-back vs write-through **2 / 2**

✓ **- 0 Correct**

2.4 Advantage of write-back **2 / 2**

✓ **- 0 Correct**

2.5 problem of write-back **2 / 2**

✓ **- 0 Correct**

QUESTION 3

Copies and Links 10 pts

3.1 link vs copy **2 / 2**

✓ **- 0 Correct**

3.2 links and unlinks **2 / 2**

✓ **- 0 Correct**

3.3 symbolic link **2 / 2**

✓ **- 0 Correct**

3.4 what can a symlink do **3 / 4**

✓ **- 1 Second Difference is vague.**

QUESTION 4

Performance problems and Diagnosis 10 pts

4.1 2 common performance problems **4 / 4**

✓ **- 0 Correct**

4.2 why each might only show up at scale **2 / 2**

✓ **- 0 Correct**

4.3 testing methodology to discover **2 / 2**

✓ **- 0 Correct**

4.4 diagnostic methodology to isolate **2 / 2**

✓ **- 0 Correct**

QUESTION 5

Journaling and Logging file systems 10 pts

5.1 purpose of a journal **2 / 2**

✓ **- 0 Correct**

5.2 logging fs vs journaling **1 / 1**

✓ **- 0 Correct**

5.3 which does fewer writes **0 / 1**

✓ **- 1 Not correct**

5.4 index value & contents **2 / 2**

✓ **- 0 Correct**

5.5 index recovery **4 / 4**

✓ **- 0 Correct**

QUESTION 6

Cryptographic Hashes 10 pts

6.1 characteristics of cryptographic hash **4 / 4**

✓ **- 0 Correct**

6.2 validating crypto-hashed password **2 / 2**

✓ **- 0 Correct**

6.3 protection against dictionary attack **0 / 1**

✓ **- 1 Not correct**

6.4 message authentication **2 / 2**

✓ **- 0 Correct**

6.5 how to defeat would-be forgers **1 / 1**

✓ **- 0 Correct**

QUESTION 7

Distributed Locking 10 pts

7.1 why dist locking is hard **2 / 2**

✓ **- 0 Correct**

7.2 how to address that problem **3 / 3**

✓ **- 0 Correct**

7.3 new failure mode in dist locking **2 / 2**

✓ **- 0 Correct**

7.4 how to address that problem **3 / 3**

✓ **- 0 Correct**

QUESTION 8

Unforgeable Capabilities 10 pts

8.1 why unforgeable capabilities **2 / 2**

✓ **- 0 Correct**

8.2 using cryptography to make them so **2 / 2**

✓ **- 0 Correct**

8.3 creating an encrypted/signed capability **1 / 2**

✓ **- 1 vague about what is encrypted with which keys**

8.4 validating an encrypted/signed capabilitiy **2 / 2**

✓ **- 0 Correct**

8.5 why hard to forge **2 / 2**

✓ **- 0 Correct**

QUESTION 9

Consistency and Persistence 10 pts

9.1 ACID acronym **4 / 4**

✓ **- 0 Correct**

9.2 Posix read-after-write - define **2 / 2**

✓ **- 0 Correct**

9.3 Flush-on-close: what and why **2 / 2**

✓ **- 0 Correct**

9.4 Flush-on-close vs Read-after-write **2 / 2**

✓ **- 0 Correct**

QUESTION 10

Eventual Consistency 10 pts

10.1 Define Eventual Consistency **2 / 2**

✓ **- 0 Correct**

10.2 Why it might be faster **2 / 2**

✓ **- 0 Correct**

10.3 Why it might be higher availability **2 / 2**

✓ **- 0 Correct**

10.4 Why does it make sense in a cloud **2 / 2**

✓ **- 0 Correct**

10.5 Dealing w/inconsistency **2 / 2**

✓ **- 0 Correct**

Name [redacted]

Studen[redacted]

All questions are of equal value. Most questions have multiple parts. You must answer every part of every question. Read each question CAREFULLY; Make sure you understand EXACTLY what question is being asked and what type of answer is expected, and make sure that your answer clearly and directly responds to the asked question.

Many students lose many points for answering questions other than the one I asked. Misunderstanding a question may be evidence that you have not mastered the underlying concepts. If you are unsure about what a question is asking for, raise your hand and ask. Spend more time thinking and less time writing. Short and clear answers get more credit than long, rambling or vague ones. Write carefully. I do not grade for penmanship, spelling or grammar, but if I cannot read or understand your answer, I can't give you credit for it.

1.

2.

3.

4.

5.

6.

7.

8.

9.

10.

Total:

1: (a) List the four necessary conditions for deadlock:

1) Mutual Exclusion
2) Hold and Wait
3) Non-preemption
4) Circular Dependency

For each condition, briefly describe a specific software deadlock situation and yow you would use that condition to prevent a deadlocks.

(b) Deadlock problem and attack based on 1st necessary condition:

- If a process A has Lock 1 and process B needs Lock 1
- Turn resource into shared resource or if not possible then implement subdivided resource and only update the major resource periodically based off of subdivided resource i.e. sloppy counter

(c) Deadlock problem and attack based on 2nd necessary condition:

- If a process A has Lock 1 and then blocks waiting for Lock 2
- Don't allow a process to block if it is holding a lock

(d) Deadlock problem and attack based on 3nd necessary condition:

- If a process A has Lock 1 and then is blocked waiting for another resource but we cannot reclaim lock from process
- Instead make it so it is a lease instead of a lock so it will expire instead of being held indefinitely

(e) Deadlock problem and attack based on 4th necessary condition:

- If process A has Lock 1 but needs Lock 2 and process B has lock 2 and needs Lock 1 and both are blocked for each other
- Make it so there is total resource ordering so you must always acquire Lock 1 before Lock 2

2: (a) What common operation is a "write-through" cache designed optimize?

Designed to optimize read disk I/O as cache is always up to date so reads can be straight from cache

(b) Why is it necessary for writes to go "through" the cache?

Otherwise we would have a stale cache and reads would have to go to disk for up-to-date values eliminating the benefit of speedy cache

(c) How is a "write-back" cache different from a "write-through" cache?

Write-back cache will buffer writes and then pour into disk in a larger chunk and return success immediately. Write-through cache on the other hand writes to cache and then straight through to disk.

(d) What advantage does a "write-back" cache give, over a "write-through" cache?

It allows for more efficient disk I/O as building a large amount of writes can 1) Allow for improved scheduling/minimized seek time when writing to disk 2) eliminate unnecessary writes if file gets deleted/overwritten subsequently

(e) What problem does a "write-back" cache create?

Write-back cache creates issue where if there is failure of system before it is written to disk, the update will not persist

3: (a) What is the most important difference between a link and a copy?

A hard link is an actual/like original link to file and thus increments the reference count while a copy does not because it will create a separate instance of the file vs. having a new link to the same file

(b) What effect does the creation of a new link have on future unlinks of a file?

A creation of a new link will increment a reference count to the file so when a future unlink happens, if there is still any link left then the file will not be deleted even if it is not the original

(c) What is a symbolic link?

Symbolic link is a specially recognized file that really just has the full path to the linked file as its contents

(d) State two (functionality) differences between accessing a file through a hard link and a symbolic link:

1) Can use symbolic link to access link in a different file system because it doesn't replicate inode number, while hard link does, which would likely lead to the same inode number being assigned to 2 files if hard link could be used this way

2) Symbolic link has deferred binding so will get most recently up to date snapshot of linked file

4: (a) List two distinct (discussed in the reading or class) common causes of performance problems in large systems:

1) Bottlenecks for resource contention

2) Too many messages - distributed system

(b) Why might each of these problems often not show up until the systems are run at scale?

1) there may be a limitation in some system resource that is not detected until there is large contention for it and this develops a convoy due to contention for this resource (possibly a lock / primary server). Initially able to be handled.

2) At first the network may be able to handle small amount of messages but only when large number of systems are trying to send messages does the network become throttled and cause the recieving server to have to spend all its time processing / responding to messages (possibly checking if file is stale) instead of other CPU intensive activities

(c) Suggest a testing methodology (how you would exercise and what you would measure) that would identify the existence of each of those types of problem.

1) Use a load generator where we test response time so we can test on a heavy load and see how response time decreases w/ increasing the load and we how quickly the system can process many clients and check if response time sees a dramatic drop indicating some bottleneck

2) Use a standard benchmark that sends a large amount of client requests and measure the throughput, how many ops / second are processed, to see if few operations are being done because too much time is spent handling messages

(d) Suggest a diagnostic methodology (what tools you would use and how the results would lead you to the specific code that needs fixing) that would identify the specific cause.

Use a executing profiling tool that lists out which functions are consuming what amount of CPU cycles / time so that if one function is using a large amount of cycles we know that it contains the bottleneck and then we can run the profiling tool to focus on that specific function and see what specific lines of code are creating the bottleneck

5: (a) What purpose is served by a file system "journal"?

It records all intended updates to file system before they are made in order to have a log that can be replayed in case of system failure before they are committed to the disk

(b) How is a "logging" file system different from a "journaling" file system?

A logging file system has the entire file system be a log/journal while a journaling file system will just use the journal be a tool to track updates mostly as write only and then read only for recovery

(c) Which is likely to do fewer disk writes? Why?

A journaling file system is likely to do less writes because a logging file system uses redirect-on-write where it makes every data/inode write immutable so any updates to an inode cause a complete rewrite of that inode for example instead of just an update to a block pointer if that is what's being changed like a journaling file system would do

(d) What is the value of the index in a logging file system, and what information would be stored in it?

An index greatly speeds up reads as it stores a mapping of inode numbers to disk location so that way the log does not need to be searched for each file access as it would be a linear search most likely w/o index.

Index gets stored in memory so it was quick to look up files through inode numbers.

(e) How is the index recovered/reconstructed after a system crash?

A snapshot of the index will periodically be taken by the log file system and will contain a pointer to the most recent snapshot of the index. Upon failure the most recent snapshot will be read into memory and all updates after that snapshot will be scanned to update index appropriately for any of these new inode mappings that need to be put in index.

6: (a) List two of the key characteristics of a cryptographic hash (as distinct from an ordianry hash) algorithm.

1) It is infasible to compute M(H) (the original message) from the hashed message.

2) It is infeasible to find an M' such that H(M) = H(M')

(b) Briefly describe the process for validating a password that has been protected with a cryptographic hash.

A password will be put through the cryptographic hash once entered and only then will it be sent to password list to be compared to the stored password protected w/ the cryptographic hash.

(c) Briefly describe a technique for defending such passwords against "dictionary attacks"

~~the same sort of diagnostic method such as Captcha in order to~~

Force user to have uncommon passwords by use of symbols, no common words, names, etc.

(d) Briefly describe the process of using a cryptographic hash to authenticate the contents of a file or message.

Message will be sent over normal channel and then take the cryptographic hash of message (digital signature) and send it to reciever over a secure channel (SSL) and have the reciever take the cryptographic hash using same algorithm and compare it to be the same to ensure there was not man in the middle that altered the message.

(e) How do we prevent a would-be attacker from simplying computing a new cryptographic hash that matched the corrupted version of the file or message?

We would need to send the digital sig over a secure channel such as SSL or encrypt it w/ public/private key encryption

7: (a) Briefly describe a characteristic of distributed systems that makes it more difficult to implement locking operations.

There is no shared memory due to spatial separation thus making it difficult to have individual nodes agree on the ownership of a lock w/o centralized management.

(b) Briefly describe the mechanism that addresses this problem, and how it solves the problem.

Introduce a resource manager server that is responsible for giving leases to nodes that gives a cookie w/ node name, given resource, and allocated time/expiration of their allocated control of this resource. Thus allocation is all centralized on a single node w/ no issues of temporal/spatial separation. Also the cookie will likely be encrypted w/ worker servers public key so worker server can ensure the cookie comes from the resource manager server

(c) Briefly describe a mode of failure (not related to the above) in distributed systems that does not have to be addressed in a single-node locking.

If a node dies while it holds a lock then we need to preempt and recover the lock because otherwise other nodes will wait indefinitely for this resource. A single node system does not have this issue as it will auto renew all locks on reboot and there is no contention from other nodes for this resource once it fails.

(d) Briefly describe the mechanism that addresses this problem, and how it solves the problem.

This is solved w/ leases that lead to preemption of locks after allocated time of resource ownership expires. Thus if a node dies with a lease it will soon expire at a time known to the resource manager at which it can be allocated ~~to~~ someone else.

8: (a) Why is it important that "capabilities" be unforgeable?

Otherwise an unauthorized node/user/process could simply forge the capability and get access to a crucial system resource that it is not meant to have access to by copying the capability from someone who was correctly given it.

(b) Briefly describe a cryptographic approach to make capabilities unforgeable?

Use a 'cookie' in the form of a capability granted from a sparse name space that is unique, expires. It can be granted also from a central authorization server that is trusted by requestor and by verifier. The cookie can then be encrypted by authentication server with its private key so that it is guaranteed to have come from trusted source.

(c) Briefly describe the process of creating such a capability?

~~1) Go to authentication server for request of capability.~~
~~2) If Kerberos, server and client have shared secret key - symmetric - to confirm identity of authentication server and of client~~
~~3) Create capability/cookie ticket and encrypt with a server/worker symmetric key and then a similar ticket w/ client symmetric key. Each has the same symmetric session key w/ allowed capability~~

Have a requestor agent describe to OS/authentication server what it is looking to do, check permissions of requestor, and then generate cookie capability from sparse, large namespace. Then if authentication server, encrypt w/ private key and return to requestor to deliver to verifier.

(d) Briefly describe the process of verifying such a capability?

Have the verifier decrypt capability w/ public key of authentication server, thus ensuring it was from trusted source. Then inspect cookie/capability to check if id matches requestor's id and if the capability has expired yet. If not, then allow access.

(e) Briefly explain why they would be so difficult to forge?

Because they are from a large, sparse [illegible] that quickly expires so first off it would be difficult to guess what the cookie might be, then if it has already been used it may be marked as expired, and finally it will quickly expire so if it was forged it won't be useful anymore. Lastly, if used w/ public private key encryption, the capability could not be forged as it would be originally signed w/ private key unique to authentication server and the public key would only decrypt it from that private key signee.

9: (a) What does the aconym "ACID" stand for (each word, brief description of each)?

A - atomicity - all or nothing transactions

C - consistency - database goes from one consistent state to another

I - Isolation - two parallel requests will happen sequentially as if they were receiv[ed] sequentially

D - Durability - once a transaction has been committed it will persist through power failure/other system faults

(b) Briefly define POSIX Read-After-Write consistency.

Immediately after a write has been issued, all subsequent reads/opens of the file will display the most recent write.

(c) What is Flush-On-Close Persistence, and what problem motivates it?

Do not commit writes to persistent storage until close() or fsync() or similar function is called. Instead buffer all writes in the meantime.

This is motivated by slow disk I/O, especially write performance, because this can allow for better scheduling of disk writes and eliminate unnecessary writes in the case of file deletion seen after creation or subsequent quick updates to a file.

(d) Flush-On-Close Persistence would seem to negate Read-After-Write Consistency. How/Why is it possible for a system to simultaneously offer both?

It is possible by having a write-back cache that will keep all recent updates in cache thus allowing subsequent reads to still have the most recent updates by checking cache even though it hasn't been committed to persistent storage.

10: (a) Briefly define "Eventual Consistency"

At a given point in time, there is no guarantee that there is a single state of an object / resource. Instead it is only guaranteed that 'eventually' all nodes will agree upon a single state if there are no subsequent operations, through communication between nodes.

(b) Why might it offer a quicker response time than Posix-consistent operations?

It doesn't require an update to be spread to all nodes participating in the database / accessing the resource. Thus we can make the update locally and then take a snapshot and deliver that at a later time to other nodes but return completion just after we have updated locally. Thus each update is significantly faster allowing us to process many more requests w/o waiting for entire system to reach consensus.

(c) Why might it offer higher availability than Posix-consistent operations?

This way after a node failure, a system does not need to go through all updates that it missed so it can be readily available to handle other requests as mean time to repair will be faster.

Also if another node fails, don't need to wait for it to process update like w/ POSIX-consistent and thus availability is not held back by other nodes, instead allowing them to eventually receive update.

(d) Why (other than the above two reasons) might it make sense for a cloud-based service?

A cloud based service desires to be linearly scalable so if it requires POSIX-consistency the cost of each request will increase linearly as the number of nodes increase. W/ Eventual consistency this does not happen so cost does not scale linearly w/ node amount thus creating actual opp. for linear scalability as desired w/ cloud computing.

(e) Give an example of how an application might reasonably deal with temporary inconsistencies in returned results.

It might evaluate what the data type is / what it is being used for and then make decision on which result to trust based on this. Thus a client might see the data value is a counter and thus use the higher value of returned values.

CS111 Final (part 2)

[redacted]

TOTAL POINTS

88 / 199

QUESTION 1

User Mode Threads 33 pts

1.1 Data Structures 0 / 8

- **- 0** Correct
- ✓ **- 8 Wrong/No answer**
- **- 2** Descriptor id, scheduling info, pointer to stack allocated by thread_create function and deallocated by thread_destroy function
- **- 5** Thread stack allocated by thread_create function and deallocated by thread_exit function. Queue for runnable threads
- **- 3** Thread stack allocated by thread_create function and deallocated by thread_exit function, Descriptor id
- **- 6** Descriptor, id, scheduling info, pointer to stack, Thread stack allocated by thread_create function and deallocated by thread_exit function
- **- 1** Queue for runnable threads, scheduling info,

1.2 Methods 0 / 14

- ✓ **- 14 Did not answer / Wrong answer**
- **- 0** Correct
- **- 12** thread_create, dispatcher, thread_yield, thread_block, thread_unblock, thread _exit, thread_destroy
- **- 9** thread_create, dispatcher, thread_yield, thread_block, thread_unblock, thread _exit, thread_destroy
- **- 5** thread_create, dispatcher, thread_yield, thread_block, thread_unblock, thread _exit, thread_destroy
- **- 6** thread_create, dispatcher, thread_yield, thread_block, thread_unblock, thread _exit, thread_destroy
- **- 13** thread_create, dispatcher, thread_yield, thread_block, thread_unblock, thread _exit, thread_destroy
- **- 10** thread_create, dispatcher, thread_yield, thread_block, thread_unblock, thread _exit, thread_destroy

1.3 Preemptive Schedulng 0 / 5

- **- 0** Correct
- ✓ **- 5 Wrong / No answer**
- **- 2** Set an alarm while dispatching the thread, catch the result signal, call thread_yield on behalf of preempted thread.
- **- 3** Set an alarm while dispatching the thread, catch the result signal, call thread_yield on behalf of preempted thread.

1.4 Mutexes 0 / 6

- **- 0** Correct
- ✓ **- 6 Wrong/no answer**
- **- 3** Use atomic instructions to check/seize locks. Use thread_block to block a thread awaiting a mutex and thread_unblock to wake up a thread when the awaited mutex became available.
- **- 4** Use atomic instructions to check/seize locks. Use thread_block to block a thread awaiting a mutex and thread_unblock to wake up a thread when the awaited mutex became available.

QUESTION 2

Dynamic Equilibrium 33 pts

2.1 The Two Forces 4 / 5

- **- 0** Correct
- **- 5** Not Answered.
- **- 4** Click here to replace this description.
- **- 3** Click here to replace this description.
- **- 2** Click here to replace this description.
- ✓ **- 1 Click here to replace this description.**

2.2 A larger working set 5 / 6

- 0 Correct
- 6 Not Answered
- 5 Click here to replace this description.
- 4 Click here to replace this description.
- 3 Click here to replace this description.
- 2 Click here to replace this description.
- ✓ - 1 **Click here to replace this description.**

2.3 A smaller working set 6 / 6

- ✓ - 0 **Correct**
- 6 Not Answered.
- 5 Click here to replace this description.
- 4 Click here to replace this description.
- 3 Click here to replace this description.
- 2 Click here to replace this description.
- 1 Click here to replace this description.

2.4 More competing processes 6 / 6

- ✓ - 0 **Correct**
- 6 Not Answered.
- 5 Click here to replace this description.
- 4 Click here to replace this description.
- 3 Click here to replace this description.
- 2 Click here to replace this description.
- 1 Click here to replace this description.
- 0 Click here to replace this description.

2.5 Constructiveness? 4 / 4

- ✓ - 0 **Correct**
- 4 Not Answered.
- 3 Click here to replace this description.
- 2 Click here to replace this description.
- 1 Click here to replace this description.

2.6 Another example 6 / 6

- ✓ - 0 **Correct**
- 6 Not Answered.
- 2 Click here to replace this description.
- 3 Click here to replace this description.

QUESTION 3

Critical Sections 33 pts

3.1 Circle the Critical Sections 0 / 18

- 0 Correct
- 4 enqueue
- 4 getnext
- 4 dequeue
- 4 suspend_req
- 2 No extra places
- ✓ - 18 **not attempted**

3.2 Protection 0 / 15

- 0 Correct
- ✓ - 15 **not attempted**
- 2 coarse grained locking
- 4 missed transaction/deadlock in suspend_req

QUESTION 4

Deadlock Problems 33 pts

4.1 Distributed Lock Manager 8 / 8

- ✓ - 0 **Correct**
- 1 did not honor problem constraints
- 1 vague/confused
- 1 weak justification
- 2 ineffective/impractical
- 1 not robust in face of node failures
- 8 n/a

4.2 Device Driver Queue 4 / 8

- 0 Correct
- ✓ - 1 **vague/confused**
- 2 no protection against MP parallelism
- ✓ - 1 **no protection against interrupts**
- ✓ - 1 **no protection against int/deadlock**
- ✓ - 1 **inadequate deadlock protection**
- 1 terrible performance
- 2 no protection against deadlock
- 8 n/a

4.3 Message Buffers 3 / 8

- 0 Correct
- ✓ - 1 **weak justification**
- ✓ - 2 **misunderstood problem**
- ✓ - 2 **vague/confused solution**
- 2 did not prevent deadlocks
- 2 created bottleneck
- 8 n/a

4.4 Locks and Blocking Requests 7 / 8

- 0 Correct

✓ - 1 vague/confused solution

- 2 misunderstood problem
- 2 violated problem constraints
- 2 failed to prevent deadlock
- 8 n/a

4.5 freebie 1 / 1

✓ - 0 Correct

- 1 n/a

QUESTION 5

Copy on Write File Systems 33 pts

5.1 Define 10 / 10

✓ - 0 Correct

- 2 change pointers to point to new info.
- 10 Wrong/ No answer;

Correct answer: Don't overwrite old info, new copy of updated info, change pointers to point to new info.

- 7 Don't overwrite old info, change pointers to point to new info.
- 8 Don't overwrite old info, new copy of updated info, change pointers to point to new info.
- 4 Don't overwrite old info, change pointers to point to new info.

5.2 Robustness 5 / 5

✓ - 0 Correct

- 5 Wrong/No answer
- 2 Click here to replace this description.
- 4 Click here to replace this description.

5.3 Space Saving 3 / 3

✓ - 0 Correct

- 3 Wrong/No answer

Correct: A C-o-w clone allows multiple files to share all the same data blocks and only creates new copies when a change is made

- 2 A C-o-w clone allows multiple files to share all the same data blocks and only creates new copies when a change is made

5.4 Enabled Functionality 5 / 5

✓ - 0 Correct

- 5 Wrong/No answer
- 3 Since old copies are still available, user can see older versions of files as well.
- 2 Since old copies are still available, user can see older versions of files as well.

5.5 The problem 5 / 5

✓ - 0 Correct

- 5 Wrong/No answer

Correct: Reclaiming the space occupied by old copies as old copies are not overwritten

- 4 Click here to replace this description.
- 1 Click here to replace this description.
- 2 Click here to replace this description.

5.6 The solution 5 / 5

✓ - 0 Correct

- 5 Wrong/No answer Correct: Suggest a way for garbage collection of older versions of a file
- 2 Click here to replace this description.
- 3 Click here to replace this description.

QUESTION 6

A New Service 33 pts

6.1 Why RESTful 0 / 10

- 0 Correct
- 1 not a significant benefit
- 2 closely related benefits
- 2 not a recognized benefit
- 1 weak justification
- 2 no justification
- 2 definition is not justification
- 5 one good answer

✓ - 10 n/a

6.2 Authentication/Authorization 0 / 9

- 0 Correct
- 3 does not honor problem constraints
- 3 vague/confused
- 3 impractical/ineffective

✓ - 9 n/a

6.3 How Owner Obtains 0 / 3

- 0 Correct
- 1 does not honor problem constraints
- 1 ineffective
- 1 vague/confused

✓ - **3** n/a

6.4 How Owner Authorizes Others 0 / 3

- **0** Correct
- **1** does not honor problem constraints
- **1** vague/confused
- **1** ineffective/impractical

✓ - **3** n/a

6.5 Multi-Level Access Control 0 / 3

- **0** Correct
- **1** does not honor problem constraints
- **1** vague/confused
- **1** ineffective/impractical

✓ - **3 n/a does not respond to question**

6.6 Privacy and Integrity 0 / 5

- **0** Correct
- **1** RESTful protocols are layerable: just use SSL
- **1** vague
- **1** impractical
- **1** does not honor problem constraints
- **1** ineffective

✓ - **5** N/A

QUESTION 7

7 Freebie 1 / 1

✓ - **0** Correct

CS 111 Final (part 2) 06/12/2017

[redacted]

This exam is comprised of six problems, each problem being a full page of related questions. You must choose three of these problems to answer. You cannot answer more than three problems. Review all of the questions and consider what answer you would give to EACH part before deciding which three to answer. All problems have the same value.

Read each question CAREFULLY, make sure you understand EXACTLY what question is being asked and what type of answer is expected, and make sure that your answer clearly and directly responds to the asked question. Many students lose many points for answering questions other than the one I asked. If you are unsure of what a question is asking for, raise your hand and ask.

I am looking for depth of understanding and the ability to solve real problems. I want to see specific answers. One-liners and vague generalities will receive little or no credit. Superficial answers that I may have accepted previously will not be accepted on this exam.

None of these questions requires long answers, but many of the questions may require you to do a lot of thinking and sketching before you come up with a reasonable answer. Feel free to use scratch paper to organize your thoughts. If the correct part of your answer is buried under a mountain of rambling, we may not find it.

If you need more space, you can overflow onto the back page. Note (on the problem page) that you are doing this.

If you answer more than three problems, we will only grade the first three. Circle the numbers of the three problems you want us to grade:

1. (design) User-Mode Threads Package
2. (concept) Dynamic Equilibrium
3. (code) Find and Fix Race Conditions
4. (approach) Deadlock Problems
5. (concept) Copy-on-Write File Systems
6. (protocol design) A New Distributed Service

Total

1: Describe how you would implement a user-mode threads package with preemptive scheduling and mutexes on a system that supported only processes. Briefly describe the major routines in this package, the major data structures you would create, the problems you would have to solve, and how you would solve each of them.

(a) key data structures: purpose, contents, and management (allocation/dealocation)

(b) key methods: signatures, brief description of functionality, and how you would implement it.

(c) Explain how you would implement preemptive scheduling.

(d) Explain how you would implement mutex operations.

2: A dynamic equilibrium is the net result of two (or more) opposing processes that drive responses to ever changing system loads.

(a) What are the two competing forces that drive a process' dynamic working set size.

1) The amount of page faults that are occuring for the process

2) The amount of processes that are running and thus requiring pages/space in memory/their own working sets.

(b) Explain how these forces drive the system to respond when a process starts referencing more pages per second.

As a process references more pages/second then the number of page faults would likely increase. This looks at other processes that have not been referencing many of their pages and reallocate their number of pages in memory to be lower and grant them to this process needing more pages.

(c) Explain how these forces drive the system to respond when a process starts refrencing fewer pages per second.

As we see (possibly using a clock pointer) that pages are not being referenced since the last time the clock went around, we see that process only really needs smaller number of allocated pages and thus will grant these pages to someone else/free to speed up subsequent page loads as this will likely also mean fewer page faults indicating it has extra pages unneeded in memory.

(d) Explain how these forces drive the system to respond to an increase in the number of in-memory processes that are competing for memory.

As more processes are in memory, the working set average size will decrease per process. May need to swap out some process if we start experiencing thrashing because the page faults are increasing dramatically for a process but every process already only has a few pages in memory.

(e) Explain why the above (d) response is (or is not) a constructive one.

It is not constructive because the solution to more processes is getting rid of some of the processes from memory meaning while we won't have as much thrashing per process because a single process may have its necessary pages in memory, a process that needs to run will need to be completely swapped in w/ another one being swapped out instead for it.

(f) Briefly describe another (unrelated to paging) dynamic equilibrium process within an Operating System, and the competing forces that robustly drive "good" resource allocation decisions.

Choosing what processes get scheduled in what queue with Multi-Level Feedback Queue scheduling. Competing forces are # time slice ends, # yields, and length of time-slice for each queue. Need to move processes between queues depending on these factors and achieve dynamic equilibrium as well as changing time slice sizes based on feedback.
All can be dec w/ parameters

3: (a) Study the following code, taken from a high contention, multi-threaded application (that operates on numerous queues), and circle the actual critical sections (sensitive use/updates of shared variables).

```
struct request {
        struct request *next;          // pointer to next task in list
        long key;                      // record identifier (for sorting)
        ...                            // other info ... irrelevent to problem
};
// insert a request into a queue, keep them ordered by key
void enqueue( struct request *req, struct request **head ) {
        struct request *rp, **rpp;
        for (rpp = head; rp = *rpp; rpp = &(rp->next))
                if (rp->key >= req->key)
                        break;
        req->next = rp;                // that one is now after us
        *rpp = req;                    // previous pointer now points to me
}
// get the next request from a queue
struct request *getnext( struct request **head ) {
        struct request *rp;
        rp = *head;
        if (rp != 0)
                *head = rp->r_next;    // next becomes new first
        return( rp );
}
// remove a request from a queue
struct request *dequeue( long findkey, struct request **head ) {
        struct request *rp, **rpp;
        for (rpp = head; rp = *rpp; rpp = &(rp->next))
                if (rp->key >= findkey)
                        break;

        if (rp->key > findkey)         // passed it without finding it
                return( (struct request *) 0 );
        *rpp = rp->next;               // previous now points past us
        return( rp );                  // return found element
}
// all of the following functions must also be Deadlock and MT-safe
submit_req( ... ) {
        ...
        enqueue( req, &active_list );
        ...
}
process_req( ... ) {
        ...
        req = getnext( &active_list );
        ...
}
// this update must be made atomically
suspend_req( key, &old_queue, &new_queue  ) {
        ...
        req = dequeue( key, &old_queue );
        enqueue( req, &new_queue );
        ...
}
```

(b) Describe the approach you will take to protecting the critical sections, briefly explain why this is the right approach, and update the above code to show how you would implement safe and correct serialization.

4a: A network lock manager provides locking services for a very wide range of distributed resources that are shared by thousands of clients. The clients are a wide range of applications running on many different operating systems, and not all of the resources they need are managed by the network lock manager. How can we ensure network locks will not contribute to deadlocks among the client applications? Justify your choice.

We give all locks as leases so deadlock cannot occur because a requirement for deadlock is non-preemption which a lease avoids because it is preemptable as the network lock manager will be able to redistribute lock to other requesting processes once allotted time of lease is up.

4b: A device driver maintains a queue of pending requests. The read and write routines add requests to the queue, and the start-up routine (which can be called either from the read routine, the write routine, or the interrupt handler) takes requests off of the queue. For request scheduling reasons, the queue is implemented as a doubly linked list, which cannot be maintained with atomic instructions. How can we protect the queue without creating potential deadlocks? Justify your choice.

We can introduce a list wide lock and then also a per-request lock. Thus we introduce total resource ordering where before any request is taken off/added to the queue, the list wide lock must be acquired. Then to access any single request, the request lock must be obtained. This total resource ordering prevents deadlocks because we know that the list will not be modified by multiple routines as only one thread can have the lock at any given time. Additionally we know if we have request lock we can safely operate on nearby requests to keep list intact for removing, for example, because nearby requests will not be altered by other threads since we own list-wide lock. Releases in reverse order to ensure total ordering

Removes circular dependency

4c: A massively parallel data reduction engine eliminated mutual exclusion and achieved significant scalability by dividing the data set into thousands of partitions, and assigning a single thread to perform all operations on each data partition. The data reduction process is such that most operations require multiple requests for operations in other partitions. We have seen situations where the system deadlocked on memory, as all of the agents were trying to allocate memory at the same time to respond to requests from other agents. How can we prevent such deadlocks? Justify your decision.

We can introduce a specific thread/set of threads that are also partitioned. For memory allocation thread per 50 data partition threads, that are responsible for handling memory allocation and called by these set of partitioning threads. Now since these only have the responsibility of allocating memory, we know they will not block as they are not reliant on other threads. Then there will be no deadlocks with partitioned threads as there is a bounded wait time on each memory allocation request.

4d: A massively parallel, distributed application with thousands of concurrent threads operates on millions of mutex-protected objects. One thread only needs to lock one object at a time, but it often needs to await confirmation messages (from other threads which may have to lock other objects) before it can complete an operation. How can we prevent deadlocks between object locks and response awaiting? Justify your choice.

We introduce a correct, total ordering of events where an object does not and cannot acquire a lock and then be blocked waiting on a message. Any thread that tries to block is forced to relinquish locks, and response confirmations are blocked for before acquiring necessary mutex for a specific thread. This causes hold-and-wait condition for deadlocks to be removed as well as circular dependency thus there can be no deadlocks.

5: (a) What does it mean to describe a file system as Copy-on-Write?

A file system has each entry be immutable, meaning it doesn't alter copies that have been recorded, it either just writes out file data blocks/inode to new space in disk and update pointer to the inode if a complete overwrite or will read in old inode, update, and then write out new inode/data blocks to new space in disk/memory.

(b) Explain how it can improve file system robustness, and how it achieves this.

It improves robustness because 1) we can space out where we write to memory so for flash this will prevent signal degradation of certain memory locations being overused 2) it allows for old copies to be in storage so if one copy fails we can look at older copies and only lose some updates instead of the entire file

(c) Explain how it could achieve space savings when files (like VM images) are cloned.

Instead of recopying the entire file, we could initially just have both files point to the same file in disk and then only create the new copy when it has been written to/edited similar to how a child can share address space w/ parent until data is changed and only then it is copied over to a new place in memory. Can still share overlapping, unchanged code.

(d) Describe valuable new file system functionality it can provide, and how it achieves this.

It can provide sorts of time machines/backups of files w/o much extra work simply by adding in versioning of new files and keeping track of locations on disk of older file versions that can still be accessed via version number.

(e) What significant new problem is created by making a file system Copy-on-Write?

It introduces garbage collection/external fragmentation become new we need to figure out when/how to delete the old copies that are left in storage as eventually we will need to reclaim space for reuse. Then when we delete old copies we will still have some valid files around the old copies meaning that we need to deal w/ the external fragmentation.

(f) Briefly describe how you would implement an approach to solving that problem?

I would implement a background garbage collector that periodically scans the file system and deletes old versions of files that have not been accessed in X amount of time OR are Y many versions old. Then I mark the space as free, move nearby valid entries to the central cylinders of the disk (or some other more densely allocated area) and then do coalescing to provide large chunks of free space on the disk.

6: Amazon wants to offer distributed Key-Value Stores (KVS) as a new service to be exploited both by in-cloud Virtual Machines and WAN-remote clients. The supported KVS operations will be:

```
handle = CREATE_KVS(string name)
boolean = DESTROY_KVS(handle)
string = GET(handle, string key)
boolean = PUT(handle, string key, string value)
boolean = DELETE(handle, string key)
```

(a) It has been suggested that the service access protocol should be RESTful. State two very different benefits of a RESTful interface, that would be important to the described service, and briefly explain why each is valuable to this service.

(b) They want to ensure that a KVS is only accessible by its owner and owner-authorized agents. They do not want to have authentication dialogs or to have to consult an authorization database. How can a KVS manager determine whether or not a request is from an authorized agent?

(c) How would a KVS owner gain the (initial) authorization to use a particular KVS?

(d) How would the owner grant other agents the authorization to use a particular KVS?

(e) If we wanted to distinguish multiple levels of access (GET, PUT, DELETE, DESTROY), how could you extend your proposal to enable this?

(f) Providing access mediation at the KVS is very important, but requests and responses exchanged over WAN-scale links would still be subject to Man-in-the-Middle attacks. Briefly describe an extension to protect external service requests from such attacks.