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

Fall 2017 Ion Stoica

Third Midterm Exam

November 29, 2017
CS162 Operating Systems

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

General Information:
This is a **closed book and one 2-sided handwritten note** examination. You have 80 minutes to answer as many questions as possible. The number in parentheses at the beginning of each question indicates the number of points for that question. You should read **all** of the questions before starting the exam, 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.*** If there is something in a question that you believe is open to interpretation, then please ask us about it!

Good Luck!!

QUESTION	POINTS ASSIGNED	POINTS OBTAINED
1	20	
2	16	
3	12	
4	16	
5	12	
6	24	
TOTAL	100	

P1 (20 points total) True/False and Why? **CIRCLE YOUR ANSWER**. For each question: 1 point for true/false correct, 1 point for explanation. An explanation cannot exceed 2 sentences.

a) As the size of the read operations decreases the effective bandwidth of an SSD in bits/sec also decreases.

TRUE FALSE

Why?

b) File permissions are stored in the File Allocation Table in FAT32 filesystems.

TRUE FALSE

Why?

c) Finger tables are not necessary for the correctness of Chord (e.g. correct lookup of keys).

TRUE FALSE

Why?

d) According to Little's Law, the number of customers/jobs in the system depends on the type of arrival distribution.

TRUE FALSE

Why?

e) On a remote procedure call, the client code must first marshal the arguments before passing them into the RPC, which then passes them to a client stub that handles sending them to the remote server.

TRUE FALSE

Why?

f) Storing inodes in the same cylinder group as their underlying files improves performance compared to storing them all in the outermost cylinders.

TRUE FALSE

Why?

g) According to the end-to-end principle presented in this class, there is sometimes a benefit to implementing functionality at a lower layer.

TRUE FALSE

Why?

h) "Best effort" packet delivery ensures that packets are delivered reliably and in-order.

TRUE FALSE

Why?

i) Log structured file systems generally perform much better on random writes than random reads.

TRUE FALSE

Why?

j) NTFS stores file data through fixed-size blocks organized into lists or trees in records in the Master File Table.

TRUE FALSE

Why?

P2 (16 points) **Short Answers:**

a) (8 points) [Anime Party Synchronization] A party has two types of attendees: Rem fans and Emilia fans, and neither really likes each other. Each can try to enter, or can exit, at any time. However, any fan first checks in with a bouncer (your monitor synchronization) before entering the party. If an absolute majority of one type of fan exists inside the party, then the bouncer allows all majority fans in. Minority fans are only allowed to enter if they would not create a tie or flip the majority in the party, otherwise they are blocked outside. Example: if there were 5 Rem fans in the party and they held majority, only up to 4 Emilia fans would be allowed in and the rest would block, while any number of Rem fans would be allowed in.

Additional Notes:

- Blocked and exited fans outside the party do not count for determining majorities. In a tie there is no majority.
- Fans should never hang (e.g. as soon as the minority ties or become the majority in the party, their blocked friends should enter).
- Fans cannot be blocked from exiting or forced to exit.

Implement this as a synchronization monitor using the following struct definition. Do not modify the struct or attempt to add lines of code beyond that provided. Inefficiency is acceptable, the priority is correctness:

```
typedef struct {
      struct lock lock; //lock.acquire(), lock.release()
      struct cv rem_cv; //cv.wait(&lock), cv.signal(), cv.broadcast()
      struct cv emilia_cv;
      int rem_fans;
      int emilia_fans;
} party_monitor;

rem_fan_enter(party_monitor* re) {
      re->lock.acquire();
      ____________________________________
      ____________________________________
      ____________________________________
      re->lock.release();
}
rem_fan_exit(party_monitor* re) {
      re->lock.acquire();
      ____________________________________
      ____________________________________
      re->lock.release()
}
```

```
emilia_fan_enter(party_monitor* re) {
      re->lock.acquire();
      ____________________________________
      ____________________________________
      ____________________________________
      re->lock.release();
}
emilia_fan_exit(party_monitor* re) {
      re->lock.acquire();
      ____________________________________
      ____________________________________
      re->lock.release()
}
```

b) (8 points) [Supply & Demand Paging] Fill in the table below assuming the LRU and Clock page replacement policies. There are 4 frames of physical memory. The left column indicates the frame number, and each entry to the right contains the page that resides in the corresponding physical frame, after each memory reference (we have already filled in the row corresponding to accessing A). For clock, tick the hand first before checking the use bit on a fault. For readability purposes, **only fill in the table entries that have changed and leave unchanged entries blank**.

		A	B	C	D	E	B	A	D	B	C
LRU	P1	A									
	P2										
	P3										
	P4										

Clock	P1	A									
	P2										
	P3										
	P4										

P3 (12 points) **Filesystem Magica:** Madoka Kaname is building a filesystem for Project 3. She implements a thread-safe write-back cache. She also implements extensible files and directories using her modified inode definition. For thread safety, she makes sure that a thread grabs a corresponding per-inode lock when it extends a file, and when it reads or writes a directory.

Madoka decides to implement a `move` syscall, which moves a file from directory to directory:

```
move(inode *dir1, inode *dir2, char *filename) {
 lock_acquire(dir1->lock);
 lock_acquire(dir2->lock);  // Assume no deadlocks.
 struct dir_entry entry;
 bool exists = lookup(dir1, filename, &entry);
 if (exists) {
   block_sector_t file_inode_sector = entry.sector;
   erase_directory_entry(dir1, entry);  // Writes to buffer cache.
   add_directory_entry(dir2, filename, file_inode_sector);  // Writes to
buffer cache.
 }
 lock_release(dir2->lock);
 lock_release(dir1->lock);
}
```

A user program:

```
// Assume all syscalls succeed
create("/hello/world", 12);
mv("/hello/world", "/goodbye/world"); //enters move() through syscall
printf("done!");
```

a) (4 points) To test her filesystem's reliability, Madoka runs the above user program and pulls the power after seeing the "done!" message. She finds that neither /hello/ nor /goodbye/ have her file! What happened? (1 sentence)

b) (4 points) By modifying the move function, how can Madoka make sure that at least one of the directories has a file named "world" after a crash? (1 sentence)

c) (4 points) By modifying the filesystem design, how can Madoka make sure that exactly one of the directories has a file named "world" after a crash? (1 sentence)

P4 (16 points) **No Game No Commit:** Consider the 2-phase commit protocol with a coordinator C and 3 workers W1, W2, W3. Assume:

- C communicates with W1, W2, W3 in parallel
- C timeout latency is 1s
- latency for sending/receiving messages from C to W1-W3, and for logging messages at W1-W3 are shown below (assume all other latencies are negligible):

	MSG Send/Receive Latency (each direction)	**Worker Logging latency per MSG**
W1	100ms	30ms
W2	200ms	20ms
W3	300ms	10ms

a) (4 points) What is the total amount of time taken for two-phase commit to complete successfully (starting from C sending VOTE-REQUEST messages, and ending with C receiving ACK messages from all workers)?

b) (4 points) Consider that in one particular execution of two-phase commit, **W2 crashes** during ***commit phase, and comes back up immediately after master times out***. Does the operation still commit **(circle YES or NO)**? ***What is the latency*** for the execution of two-phase commit (box final answer)?

YES NO

c) (4 points) Consider that in another execution of two-phase commit, **W1 crashes** during ***prepare phase, and comes back up immediately after master times out***. Does the operation still commit **(circle YES or NO)**? ***What is the latency*** for the execution of two-phase commit (box final answer)?

YES NO

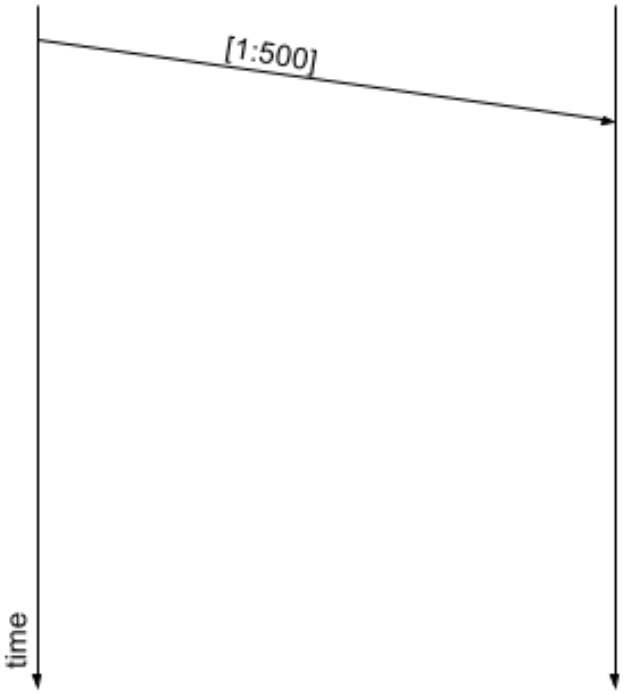
d) (4 points) If C, W1, W2, W3 were guaranteed not to crash, can we replace the two phases in 2PC with a single phase **(circle YES or NO)**? Why or why not (answer in a single sentence)?

YES NO

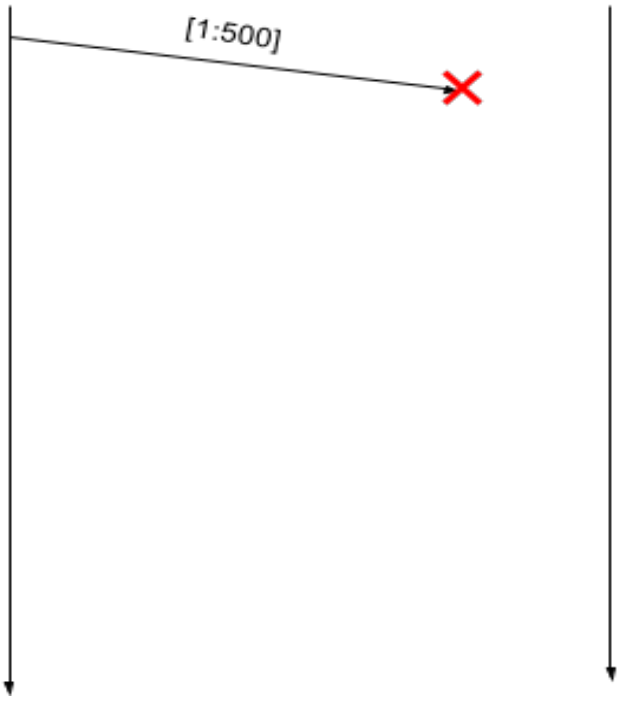
P5 (12 points) **CS168 Except Not Multiple Choice:** Consider a TCP connection, such that:

- the sender sends 1800 bytes worth of data;
- the receiver's *initial* advertised window is 1000 bytes;
- the maximum packet size is 500 bytes;
- the receiving process consumes each packet *p* that contains in-sequence bytes right after the ack for *p* is sent (by the receiver), and before the receiver receives the next packet; note that if packet *p* does not contain in-sequence bytes (i.e., the previous packet has not been received) that receiving process does not consume *p*.
- the sender can send all data before the first ack is received, of course, modulo the size of the advertised window and of the sender window.

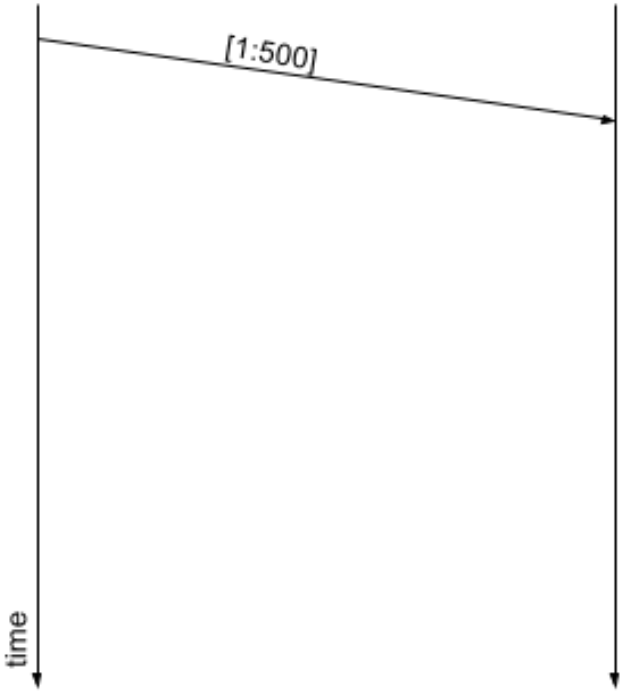
a) (4 points) Assume no packets are lost. Draw the time diagram of the packet delivery. For each packet indicates the bytes sent in that packet, and for each ack packet show the sequence number of the next expected byte, and the advertised window (denoted advWin).



b) (4 points) Assume now the 1st packet is lost and none of the other packets are lost afterwards. Draw the time diagram of the packet delivery. For each packet indicates the bytes sent in that packet, and for each ack packet show the sequence number of the next expected byte, and the advertised window.



c) (4 points) Assume now that the advertised window is 1200 bytes, and the 1st and 2nd packets are received by the receiver in reverse order, i.e., the receiver gets 2nd packet before the 1st packet. No packets are lost, and no other packets or acks are reordered. Draw the time diagram of the packet delivery. For each packet indicates the bytes sent in that packet, and for each ack packet show the sequence number of the next expected byte, and the advertised window.



P6 (24 points) **Sword Art I/O:** You're excited to download the 1st episode of Sword Art Online (SAO) from Netflix. Your client (browser) connects with the master server in the nearest Netflix datacenter, which coordinates many worker servers. The workers store an unreplicated distributed key-value store mapping "TV Show Title" -> video file.

a) [Queuing & Networking] Assume all network links have 80 gigabits/second (10 gigabytes/s) of bandwidth. The latency between you and any server in the datacenter is 100 ms, while the latency between servers in the datacenter is 10 ms. Assume that all requests and responses except for the video itself are negligible (~0 bytes) in size, and all arrivals & service times are memoryless. The video is 1 gigabyte in size. Assume in parts i), ii), & iii) that the master cached the video (no communication with workers).

i. (2 points) What is the network round trip time (from your sending the request to getting the video), ignoring queueing and processing delays?

ii. (3 points) Assume the master has a service time of 10ms and 25 requests arrive per second. Assuming steady state, what is the utilization, time spent in the queue, and the length of the master's queue?

u	
T_q	
L_q	

iii. (3 points) SAO turns out to be very popular, and the arrival rate jumps to 162 requests per second (service time is still 10ms). Assuming steady state, what is the utilization, time spent in queue, and length of the master's queue?

u	
T_q	
L_q	

iv. Assume the master no longer caches the video and you query the distributed key-value store iteratively.

α) (1 point) What is the network round trip time, ignoring processing & queuing delays?

β) (2 points) Consider the utilization, queuing time, & service time in part ii): u_{old}, T_q, T_{ser}. In the master, express the new utilization u_{new} in terms of u_{old} and compute the new queuing time T_{new} in terms of T_q and/or T_{ser}.

u_{new}	
T_{new}	

v. Assume the master no longer caches the video and you query the distributed key-value store recursively.

α) (1 point) What is the network round trip time, ignoring processing & queuing delays?

β) (2 points) Consider the utilization, queuing time, & service time in part ii): u_{old}, T_q, T_{ser}. In the master, express the new utilization u_{new} in terms of u_{old} and compute the new queuing time T_{new} in terms of T_q and/or T_{ser}. Assume it takes the same time to service either a client or a worker.

u_{new}	
T_{new}	

vi. (2 points) You want to download your SAO episode as fast as possible. Do you prefer recursive or iterative queries? Explain briefly in 1 sentence.

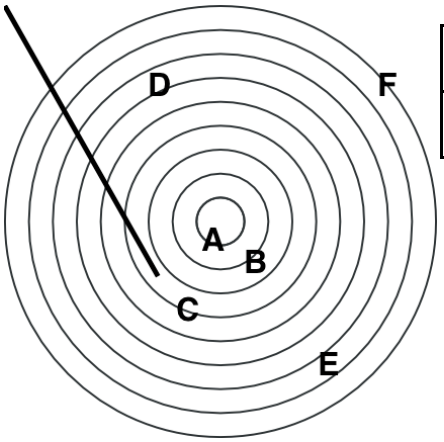
vii. (2 points) Suppose you now want to download all 50 episodes of SAO in a batch as fast as possible rather than just 1 episode. Do you prefer recursive or iterative queries? Explain briefly in 1 sentence.

b) [Disk Latency] Within a single worker server, a disk has the following characteristics:
- 1ms controller delay
- 1ms seek time
- 30000 rpm rotation speed
- 1 megabyte/s transfer rate
- 10 kilobyte sector size

i. (3 points) Compute the average rotation time, transfer time, and average total time to read 1 sector. Ignore queuing delay.

Rotation	
Transfer	
Total	

ii. Schedule the I/O operations using the following disk scheduling algorithms. The arm is initially over cylinder 35. Assume SCAN and C-SCAN begin by traversing ascending cylinder numbers, and ignore rotational delay. List the operations scheduling order from left to right.



	A	B	C	D	E	F
Cylinder #	10	21	40	64	75	90

α) (1 point) Shortest Seek Time First:

β) (1 point) SCAN (Elevator):

γ) (1 point) C-SCAN: