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

Fall 2018 Ion Stoica

Third Midterm Exam

November 28, 2018

CS162 Operating Systems

Your name	
SID	
CS162 login (e.g., s162)	
TA Name	
Discussion section time	

This is a closed book and two 2-sided handwritten notes 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
P1	22	
P2	20	
P3	18	
P4	14	
P5	12	
P6	14	
Total	100	

Problem 1: True/False questions (22 points)

MARK THE CHECKBOX NEXT TO YOUR ANSWER. For each question: 1 point for true/false correct, 1 point for explanation. An explanation **cannot** exceed 2 sentences.

a) The UNIX Fast File System puts i-node metadata in the same cylinder group as the files they describe.

□ **TRUE** □ **FALSE**

Why?

This avoids long seeks between data and metadata and was an optimization in FFS.

b) As we add elements to a struct list in Pintos, the struct list itself will also grow in size.

□ **TRUE** □ **FALSE**

Why?

The elements are stored elsewhere and linked with list_elem's that also live elsewhere, so the size of the struct list is constant.

c) In a queueing system it is possible to have unbounded service times even when the arrival rate is less than the service rate.

□ **TRUE** □ **FALSE**

Why?

For example with M/M/1 queue the service time is proportional to 1/(1-u), where u is the arrival rate over service rate. So service time goes to infinity as u approaches (but is still less than) 1.

d) According to the end-to-end argument the reliability for a file transfer application should be implemented only in the network.

□ **TRUE** □ **FALSE**

Why?

Implementing reliability in the network is not enough to ensure end-to-end reliability. For example, the data can be corrupted when red/written on the disk.

e) Congestion control ensures that the sender will not overflow the receiver's buffer.

□ **TRUE** □ **FALSE**

Why?

Congestion control makes sure that the sender doesn't overflow the router's buffer. Flow control ensure the sender doesn't overflow the receiver's buffer.

f) Consider a resource allocation system in which we can always preempt a process holding a resource. Such a system can never reach a deadlock state.

□ **TRUE** □ **FALSE**

Why?

Non-preemption is a necessary condition of the deadlock.

g) Retrieving a data block using the file allocation table (FAT) may require more than two disk accesses.

□ **TRUE** □ **FALSE**

Why?

Consider a FAT table that does not fit into a single block which would require two or more disk accesses to retrieve the FAT table and one more to retrieve the data block from disk.

h) The shortest remaining job first scheduling discipline can lead to starvation.

□ **TRUE** □ **FALSE**

Why?

Consider an infinite stream of tasks that take less than a larger task in the queue.

i) The elevator algorithm (for disk scheduling) in general leads to lower seek times than the FIFO algorithm.

□ **TRUE** □ **FALSE**

Why?

Shortest seek time first may force the arm to constantly move back and forth between cylinders with increases the seek time. Elevator algorithms like CSPAN only go in one direction, negating the extra time incurred from changing directions.

j) For a single processor, write-back caching results in fewer writes to disk than write-through caching.

□ **TRUE** □ **FALSE**

Why?

With write-through caching every update leads to writing to disk, while with write-back caching we write to disk only when replacing the page/block.

k) RAID5 with one parity block can recover from one disk failure.

□ **TRUE** □ **FALSE**

Why?

If the block is on the failed disk we can reconstruct it using the parity block.

Problem 2: Disk & RAID (20 points)

Several CS162 students decided they are going to start a start-up of streaming videos called Calfix. This start-up will help stream videos of CS162 lectures around the world. To do so, they need to understand how to design their storage system. Since these are CS162 students, they don't have too many resources, so they decided to start with very old magnetic disks for their storage system.

They bought disks with the following specs:

- 15ms controller delay
- 10ms seek time
- 3000 rpm rotation speed
- 100 megabyte/s transfer rate
- 200 kilobyte sector size.

a) What is the *average* rotation time (in ms) of the disk to read 1 sector? (2 point)

60000 (ms/minute) / 3000 (rev/minutes) / 2 = 10 ms
Note that this is *average* rotation time, as opposed to max rotation time (20 ms)

10 ms

b) What is the transfer time to read 1 sector? (2 point)

Sector_size / transfer_rate = 200 kb / (100 mb/s) = 2 ms
Note: The solutions assumed kilo and mega are represented as powers of 10. We accepted answers that assumed any representations of kilo and mega as powers of 2 and powers of 10.

2 ms

c) Assuming a lecture is stored on 20 contiguous sectors on the same track, what is the average total time to read a movie? (2 points)

15ms + 10ms + 10ms + 20*2ms = 75 ms
Common mistakes included assuming that an average rotation time is required between each contiguous sector, not multiplying the transfer time by the number of sectors, and assuming that the sectors are not contiguous (hence - multiplying everything by 20)

75 ms

The Calflix team was able to fit the lecture of the last 20 years of CS162 lectures into their "data center" on 20 disks. Also, our Calflix founders remembered they learned in CS162 about something called RAID. They started with RAID0 which stripes a lecture on all 20 disks, i.e., each disk gets an equal number of sectors of each lecture. However, there has been a bug in the implementation of RAID0 and now each sector is stored randomly (i.e., non-contiguously) on each disk.

d) What is the worst-case time it takes to read a lecture? Assume each lecture is stored on 20 sectors, like in question 3 (4 points)

Each movie has a sector stored on every disk. Each of these sectors can be read in parallel so the time to read the movie is the slowest time to read a sector. This time is (controller_delay + seek_time + max_rotation_time + sector_transfer_time) = 15ms + 10ms + 20ms + 2ms = 47ms.

Note: Here we use max_rotation_time since it's likely that across 20 disks in one case we might need to wait for a full rotation. However, partial credit was awarded to solutions that used the avg_rotation_time calculated in part a) correctly. Solutions were also not deducted points for using the sector transfer time from part b) if it was wrong. We assumed it was correct for the purposes of this problem.

Common misconception: Movie sectors had to be read serially. This is incorrect because we stripe movie sectors across RAID partitions and can read them in parallel as stated above. Many solutions used max_rotation_time/avg_rotation_time and sector_transfer_time correctly, but were not awarded points because they applied the common misconception as follows: 20 * (controller_delay + seek_time + max_rotation_time + sector_transfer_time)

47 ms

e) The Califix engineers decided to increase the quality of the movies, and now it takes 40 sectors to store each movie. What is the worst case time takes to read a higher quality movie movie? (2 points)

2*(controller_delay + seek_time + max_rotation_time + sector_transfer_time) = 15ms + 10ms + 20ms + 2ms = 94ms.

94 ms

The Calflix engineers would like to make their storage fault-tolerant so they have decided to implement RAID5 by using a parity sector for every 19 data sectors. Again the sectors of a movie are striped across all disks, and we are still incurring the previous bug in which the sectors are randomly placed on each disk. Assume each lecture is stored on 40 sectors.

f) Assume a server fails. What is the time it takes to read a movie? (3 points)

In this case we can write only 19 data sectors in parallel as we also need to write one parity sector on the 20th disk. Thus, there will be two disks storing 3 sectors of the file. As such, it will take:
3*(controller_delay + seek_time + max_rotation_time + sector_transfer_time) = 3*(15ms + 10ms + 20ms + 2ms) = 141ms.

141 ms

g) Finally the Califix engineers have found and fixed the bug, so now all sectors of a movie that are stored on the same disk are stored contiguously. What is the *maximum* time it takes to read a movie in this case? (3 points)

15ms + 10ms + 20ms + 3*2ms = 51ms.

51 ms

Problem 3: Malloc (18 points)

Recall the simple first-fit memory allocator you implemented in Homework 3.

a) (4 points) Given the following series of allocations, write out a series of calls to mm_free and mm_malloc that would cause your memory allocator to request more space using sbrk even though enough memory has already been requested. For this question, you can assume the size of each block's header is 32 bytes. Also, briefly explain in 1 or 2 sentences why this happens.

```
void *p1 = mm_malloc(1000);
void *p2 = mm_malloc(1000);
void *p3 = mm_malloc(1000);
```

```
mm_free(p1);
mm_free(p3);
mm_malloc(Z); // where 1001 <= Z <= 2032
```

Any answer mentioning something about external fragmentation gets full points.
Because the first-fit allows for external fragmentation, a request to allocate a block of the sum of the sizes of two non-contiguous free blocks will result in a request to the system for more space, even though there is enough total free space. Common incorrect answers seemed to recognize that mm_alloc also requires the metadata header space in addition to the amount of data requested but did not account for the fact that mm_free frees the block which contains the metadata header as well as the data.

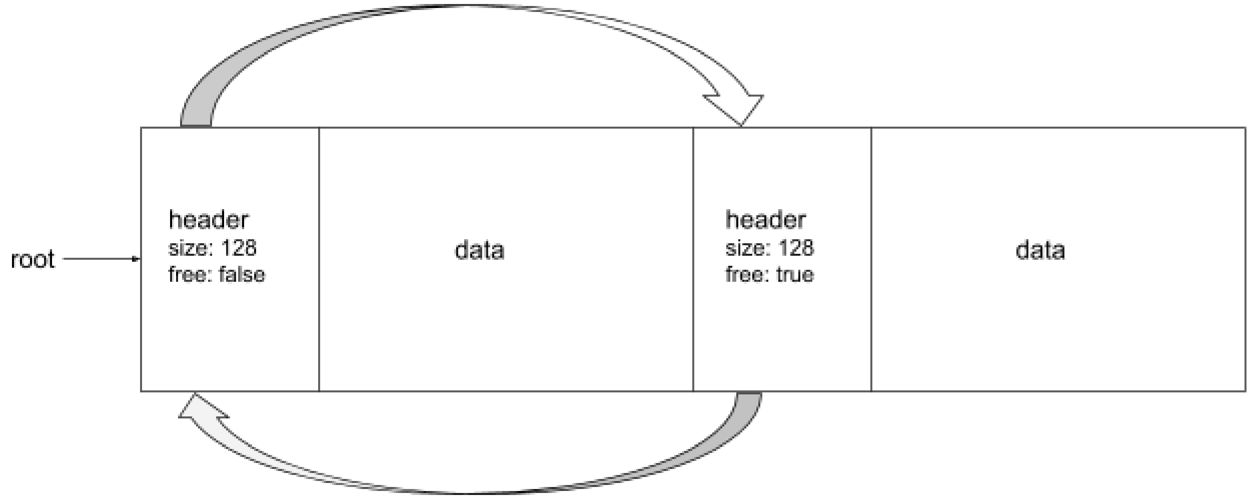
b) (2 points) You want to fix this fragmentation issue because it can potentially waste a lot of memory if it goes unchecked for too long, but you don't want to write a new allocation algorithm. Instead, you change your memory allocator so that around every one hundred calls to mm_malloc, a compaction routine is called that causes all of the allocated blocks to be moved to the beginning of the heap with no free blocks in between each non-free block. Briefly (1 sentence) explain what could go wrong with this strategy. Bad performance will not considered a potential issue.

It's possible that pointers that were returned to the user previously will no longer be valid.
Pointers set by the user in the data sections will point to invalid memory.
We did not accept answers that stated the next/prev pointers would be invalid, as the memory allocator is in complete control of the header blocks and would be able to adjust those during the compaction routine.

c) Describe the layout of the heap after the given calls to mm_malloc and mm_free using the first-fit allocator with free block coalescing specified in homework 3 by indicating the size of each block and whether each block is free or not in the tables below. Consider each part independently from the rest, and assume that metadata headers are 32 bytes large. If there are fewer blocks in the heap than there are columns, leave the extra columns blank. A block with a lower number corresponds to a block that comes earlier in the linked list. ***Block numbers may or may not correspond to pointer numbers.***
An example of an acceptable answer as well as a drawing of the corresponding layout on the heap for the following allocations is given below:

```
void *p1 = mm_malloc(128);
void *p2 = mm_malloc(128);
mm_free(p2);
```

	Block 1	Block 2	Block 3	Block 4
Size	128	128		
Free	false	true		



a) (3 points)

```
void *p1 = mm_malloc(128);
void *p2 = mm_malloc(128);
void *p3 = mm_malloc(128);
void *p4 = mm_malloc(128);
mm_free(p2);
mm_free(p3);
```

	Block 1	Block 2	Block 3	Block 4
Size	128	288	128	
Free	false	true	false	

b) (3 points)

```
void *p1 = mm_malloc(128);
void *p2 = mm_malloc(128);
void *p3 = mm_malloc(128);
mm_free(p2);
void *p4 = mm_malloc(32);
```

	Block 1	Block 2	Block 3	Block 4
Size	128	32	64	128
Free	false	false	true	false

c) (3 points)

```
void *p1 = mm_malloc(128);
void *p2 = mm_malloc(128);
void *p3 = mm_malloc(128);
mm_free(p2);
void *p4 = mm_malloc(256);
```

	Block 1	Block 2	Block 3	Block 4
Size	128	128	128	256
Free	false	true	false	false

d) (3 points)

```
void *p1 = mm_malloc(128);
void *p2 = mm_malloc(128);
void *p3 = mm_malloc(128);
void *p4 = mm_malloc(128);
mm_free(p2);
mm_free(p3);
void *p5 = mm_malloc(255);
```

	Block 1	Block 2	Block 3	Block 4
Size	128	255	1	128
Free	false	false	true	false

Problem 4. File System (14 points)

Donald wants to make some money by what he has learned from CS162. He has a customer named Hillary. Donald decided to use a new SSD drive to implement the file system. The SSD has a block size of 1KB, and the latencies to read and write a block are 50 usec, and 100 usec, respectively.

Notes:

- Ignore the block transfer time.
- All throughput results should be given in MB/sec.
- Recall that 1 sec = 1,000,000 usec.

a. (3 points) Hillary runs a program that reads 64B *random* data chunks, i.e., each chunk is stored in a different block. What is the read throughput of Hillary's program?

Since each chunk is stored in a different data block, we need to read a block for each chunk. Since we can read a block in 50 usec, it follows that we can read 20,000 block/s = 10^6 usec / (50 usec/block). Since each chunk has 64B of data, the program reads (64 B/block) * (20,000 block/s) = 1.28 MB/s.

1.28 MB/s

b. (2 points) Hillary is not happy with the throughput. To improve throughput, Donald helps Hillary to redesigning her application to store the chunks contiguously in a block. What is Hillary's program's throughout after this change?

The throughput is now (1 KB/block) * (20,000 block/s) = 20 MB/s.

20 MB/s

To implement the file system, Donald decides on the following inode implementation. Recall that the block size is 1 KB.

```
struct inode_disk {
      off_t length; /* File size in bytes. */
      block_sector_t direct[15]; /* 15 direct pointers */
      block_sector_t indirect; /* a singly indirect pointer */
      block_sector_t doubly-indirect /* a doubly indirect pointer */
      block_sector_t triply-indirect /* a triply indirect pointer */
      uint32_t unused[237]; /* Not used. */
};
```

c. (4 points) What is the maximum file size that this file system can support? (We will accept unsimplified answers.)

(15 + 2^8 + 2^16 + 2^24) * 1KB = ~16GB

~16 GB

d. (5 points) How long does it take to write an 1MB file?

An 1MB file consists of 2^10 blocks. Direct pointers account for 15 blocks and 1-indirect block of pointers for 2^8 blocks. This is not enough, so we need 3 more 2-indirect blocks, which can hold pointers for 3x2^8 blocks. So now we need:

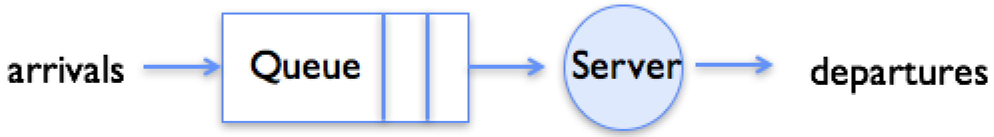
- **1** i-node block
- **1** indirect block
- **3** doubly-indirect blocks plus **1** block to maintain the pointers to the doubly-indirect blocks
- **2^10** blocks for data

In total, thus we need to write 6+2^10 blocks. Since each block write takes 100usec it will take us (6+2^10) * 0.1ms = 103 ms.

103 ms

Problem 5: Queuing Theory (12 points)

Consider a single queue/single server system as below. The mean service time of each request is 20ms.



a. (2 points) Assume the mean arrival rate is λ = 200/3 request/sec. As the time goes to infinity, what is the queueing delay?

$T_{ser}\,(avg.\ time\ to\ service\ request) = 20ms/request\ =\ 20ms/request\ \cdot (1s\ /\ 1000ms)\ = 1s/50\ requests$

$\mu\ (service\ rate)\ =\ 1/T_{ser}$ = 50 requests/s

The mean arrival rate (λ = 200/3 request/sec) is larger than the service rate, so the queuing delay grows greater and greater towards infinity.

$\boxed{\infty}$

First engineer improves the server performance so that service time is brought down to 10ms. Assume that both arrival and service times are *fixed* (i.e., a request arrives every 15ms and it takes exactly 10ms to serve a request), and that at time 0 there are 3 requests in the queue.

b. (2 points) What's the queueing delay when the system stabilizes (i.e., when queueing delay becomes a constant)?

The service time (10ms) is smaller than the arrival time (15ms), so there is no delay when the system stabilizes.

$\boxed{0}$

c. (4 points) How long does it take to stabilize?

It takes 90ms. The system stabilizes when the # of requests serviced = the # of requests arrived

t / 0.01 = 3 + (t / 0.015)

t = 0.09 s

We also accepted 60 ms as an answer for the interpretation where no request arrives at t = 0.

$\boxed{90\text{ ms}}$

Over time the load is changing and the engineers observe that requests spend now 30ms in the system (this includes both the time in the queue and at the server) on average. Assuming the mean of the arrival rate has not changed.

d. (4 points) How many requests are in the system on average?

Apply little law: N = λ * Tsys = 200/3 requests/sec * 0.03 sec = 2 requests.

2 requests

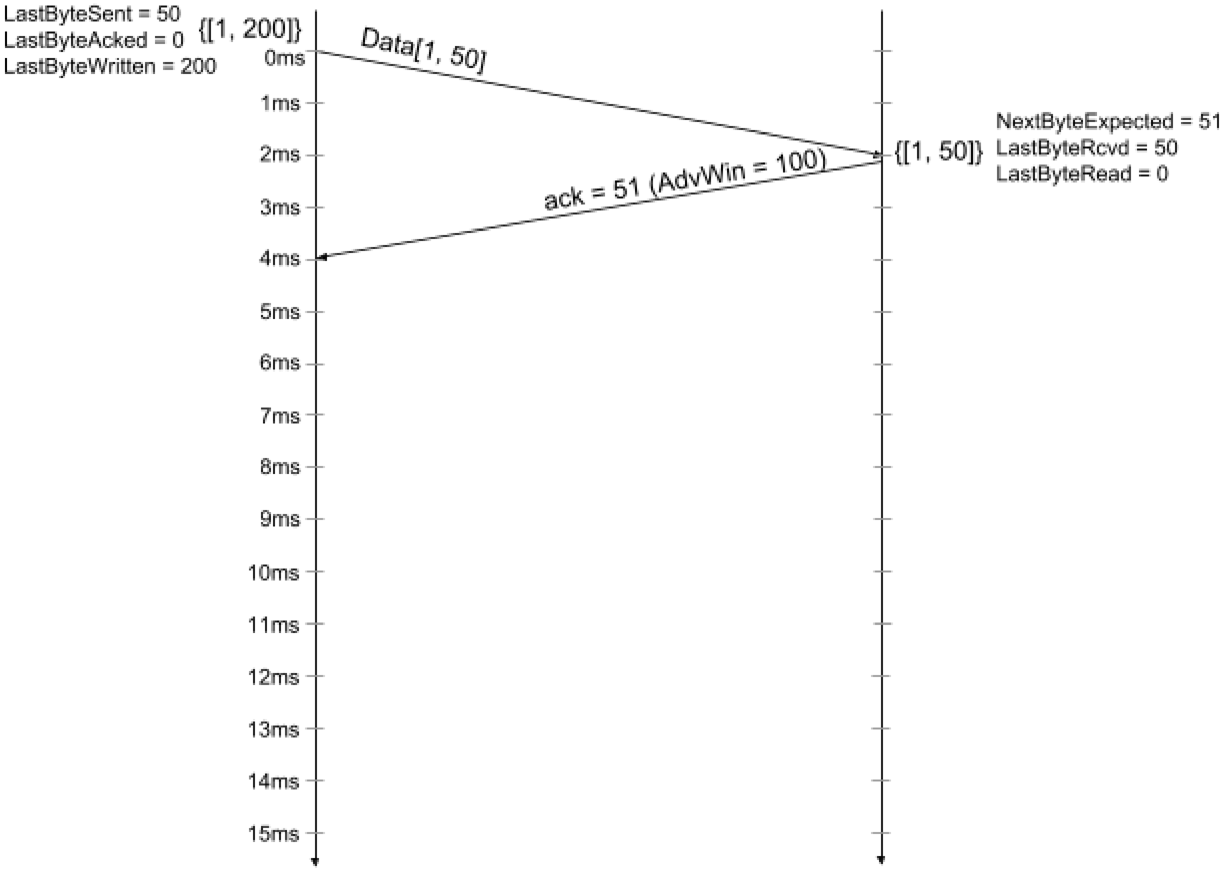
Problem 6: TCP Flow Control (14 points)

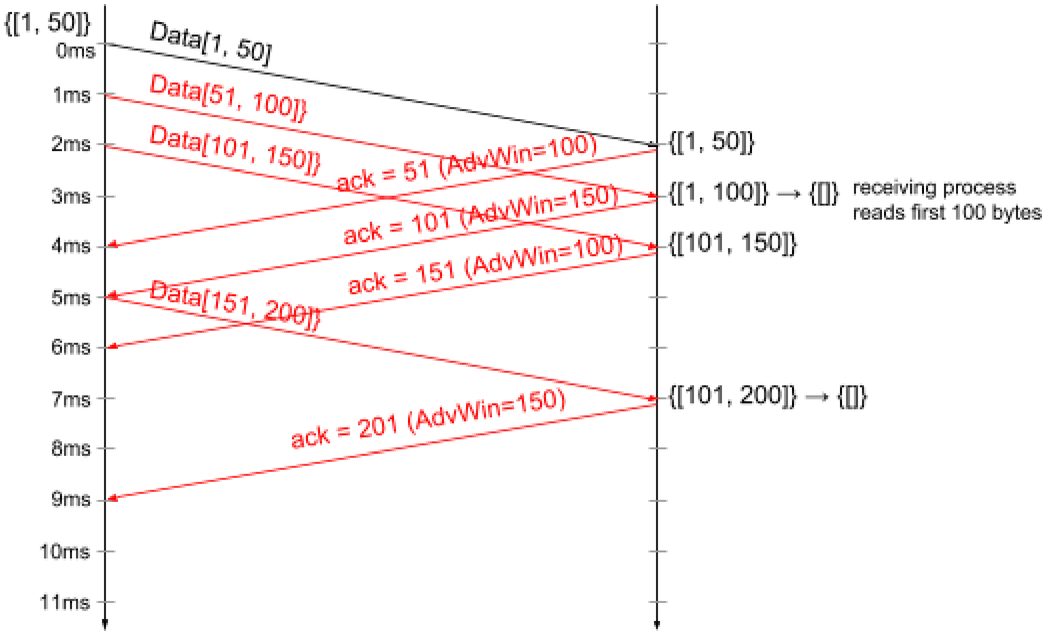
Assume we use the TCP flow control protocol to transfer 200 bytes. Next, assume the followings parameters:

- Packet size: 50 bytes
- Maximum receiver buffer (MaxRcvBuf) is 150 bytes.
- The receiving application reads from the receiver buffer 100 bytes at a time. (Recall that the receiving application reads only bytes which are insequence, i.e., with no gaps).
- Acknowledgement contains the sequence # of the next expected in-sequence byte.
- Sender can send a new packet every 1ms, and the delay between the sender and receiver is 2m each way, so it takes 4ms for the sender to get an acknowledgement.
  - ○ Each tick on the timing diagram is 1 ms.
- The sender retransmit a packet in one of the following two cases:
  - ○ If it doesn't receive an ack for that packet in 5ms.
  - ○ If it gets a duplicate acknowledgement, i.e., an acknowledgment with the same sequence number as a previous one.

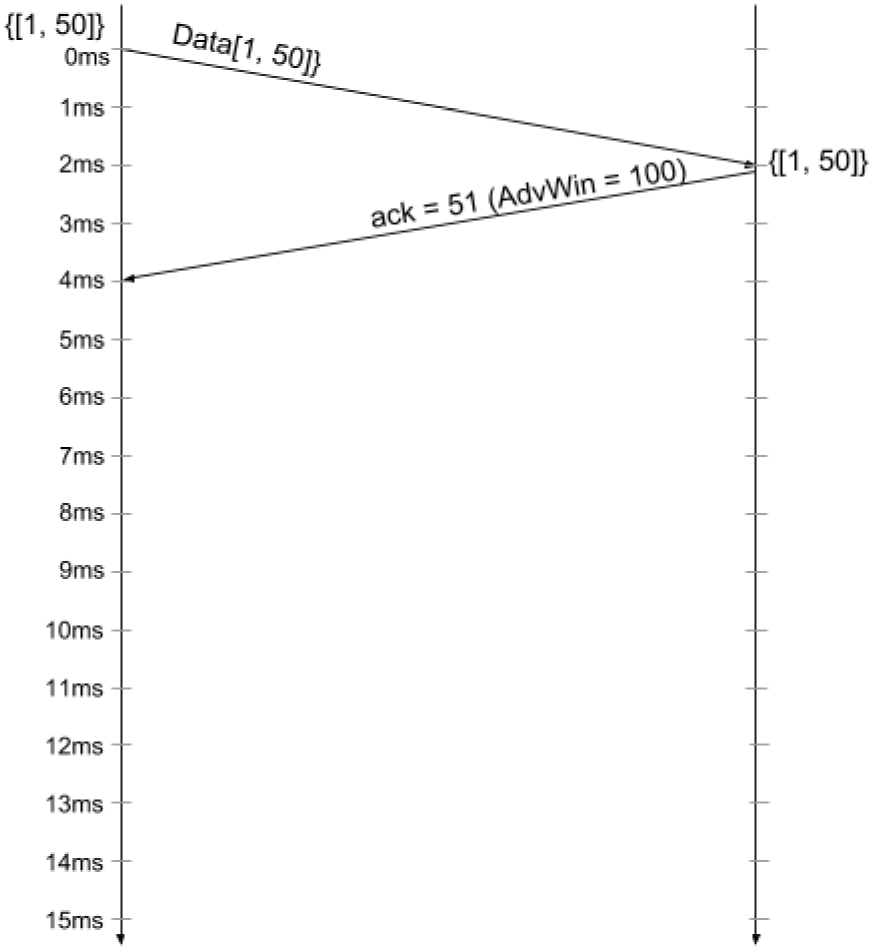
a) (4 points) Use the following figure to draw the packet diagram of the transfer. The first packet transfer and its acknowledgment is shown for you. Assume no losses.

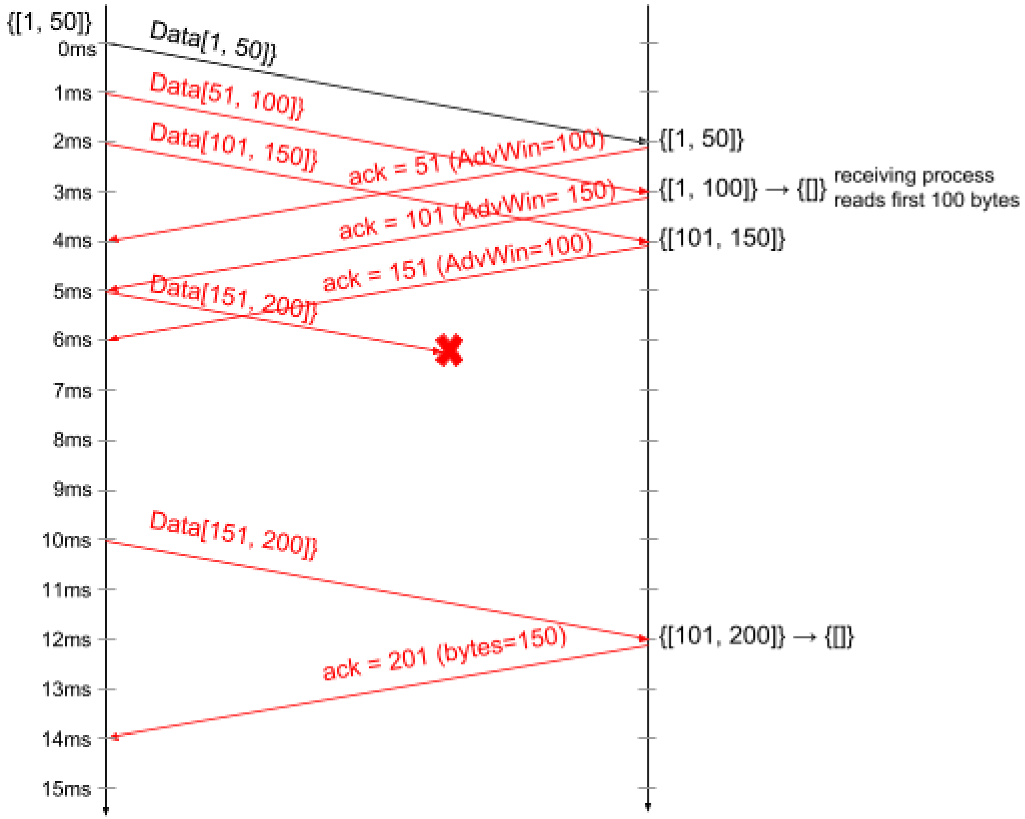
   i) Note: *LasByteSent, LastByteAcked, LastByteWritten, NextByteRcvd, LastByteRead, and LastByteExpected* are shown for your convenience. You do not need to update them after each data packet / ack is sent /received in your solution.



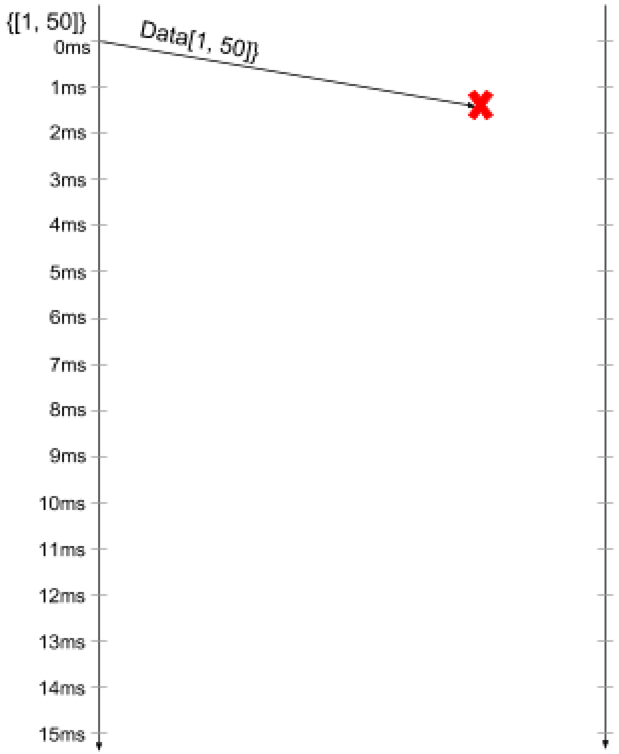


b) (5 points) Draw the packet transfer diagram assuming the *last data packet* is lost.

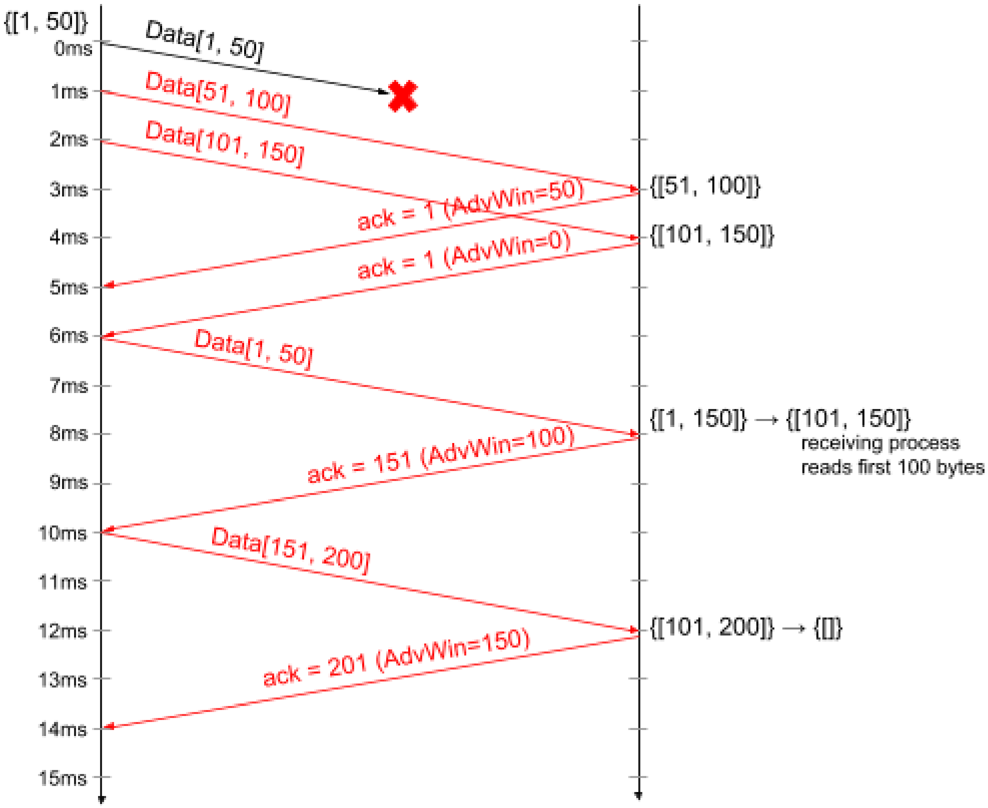


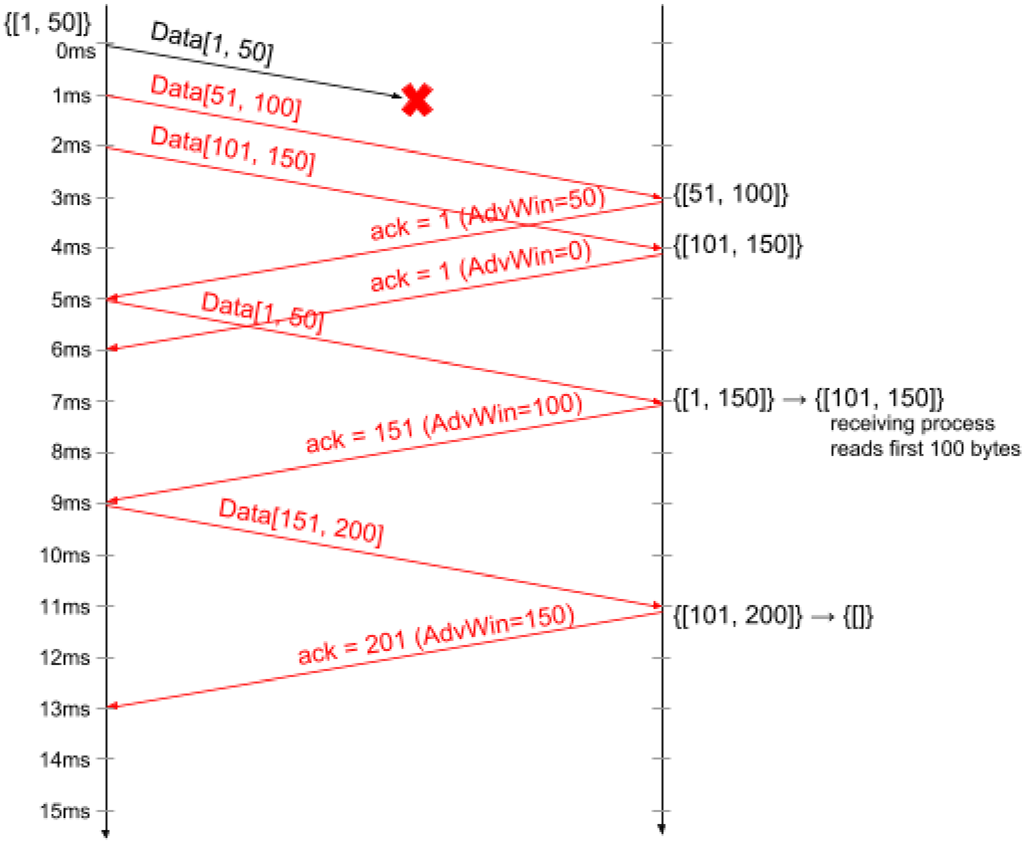


c) (5 points) Draw the packet transfer diagram assuming the first packet is lost. Assume the receiving process consumes 100 bytes of in order received data before the ack is sent out.



Both the following answers were considered correct:





nO mOrE AnIMe