6.033 Computer System Engineering

Spring 2009

Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033 Computer Systems Engineering: Spring 2007

Quiz II Solutions

There are 11 questions and 12 pages in this quiz booklet. Answer each question according to the instructions given. You have **50 minutes** to answer the questions.

Most questions are multiple-choice questions. Next to each choice, circle the word **True** or **False**, as appropriate. A correct choice will earn positive points, a wrong choice may earn negative points, and not picking a choice will score 0. The exact number of positive and negative points for each choice in a question depends on the question and choice. The maximum score for each question is given near each question; the minimum for each question is 0. Some questions are harder than others and some questions earn more points than others—you may want to skim all questions before starting.

If you find a question ambiguous, be sure to write down any assumptions you make. **Be neat and legible.** If we can't understand your answer, we can't give you credit!

Write your name in the space below AND at the bottom of each page of this booklet.

THIS IS AN OPEN BOOK, OPEN NOTES QUIZ.
NO PHONES, NO COMPUTERS, NO LAPTOPS, NO PDAS, ETC.

CIRCLE your recitation section number:

10:00	1. Madden/Komal		
11:00	2. Madden/Zhang	3. Katabi/Komal	10. Yip/Chachulski
12:00	4. Yip/Zhang	5. Katabi/Chachulski	
1:00	6. Ward/Shih	7. Girod/Schultz	
2:00	8. Ward/Schultz	9. Girod/Shih	

Do not write in the boxes below

1-6 (xx/36)	7-10 (xx/39)	11 (xx/25)	Total (xx/100)

Name:

I Reading Questions

1. [4 points]: A choice motivated by an *end-to-end argument* in the design of the Ethernet (as described in reading #9) is

(Circle the BEST answer)

A. Carrier detection.
Incorrect. Ethernet uses carrier detection in its medium access algorithm. It has no direct relationship to the end-to-end argument.

B. Packet error detection.
Incorrect. Ethernet detects if packets are transmitted incorrectly so that some layer can retransmit them. See 1D.

C. Collision detection.
Incorrect. Ethernet uses collision detection in its medium access algorithm.

D. Best effort packet delivery.
Correct. Ethernet doesn't guarantee that packets are delivered reliable, but leaves it up to end-to-end layer protocols, such as EFTP, to deliver data reliably. This is a classic example of the end-to-end argument.

E. None of the above.
Incorrect. D is a good answer and the others are incorrect, so D must be the best answer. Thus, E is incorrect.

2. [4 points]: Minimum packet size is constrained by maximum Ethernet (as described in reading #9) cable length because of considerations relating to

(Circle the BEST answer)

A. Carrier detection.
Incorrect. A node decides locally if the carrier is currently in use. If not, it sends. This decision is independent of cable length.

B. Packet error detection.
Incorrect. Error detection relies on a CRC, which is independent of cable length.

C. Collision detection.
Correct. A node must wait a certain period of time to decide if a collision has happened. This period is directly related to the cable length.

D. Best effort packet delivery.
Incorrect. Best-effort packet delivery is a design decision that is independent of cable length.

E. None of the above.
Incorrect. 2C is the best answer.

Name:

3. [4 points]: The ordering of responses to a Google search (reading #13) is determined solely by
(Circle the BEST answer)

A. Auction among Google clients.

B. Number of citations.

C. Page ranks.

D. Hit lists.

E. None of the above.
Ordering in Google search is determined by a number of factors, including page ranks, hit lists, etc, and none of them solely determine the ordering. Therefore, 3E is right answer.

4. [4 points]: The LFS authors (reading #15) distinguish between ”hot” and ”cold” segments in order to
(Circle the BEST answer)

A. Control the temperature on the surface of the disk
Incorrect. Hot and cold segments have to do with how frequently recently they are used, and not with temperature.

B. Minimize garbage collection overhead
Correct. Garbage collecting old segments instead of hot segments is good, because it is likely that cold segments won't be overwritten and create new garbage.

C. Optimize performance through the use of RAM caches
Incorrect. Recently-accessed segments will be loaded into cache, independent of whether they were cold or warm.

D. Prioritize redundant storage of most valuable data
Incorrect. This answer has nothing to do with the question.

E. None of the above
Incorrect. 4.B is the best answer.

Name:

ACo, BCo, and CCo each pay their internet service provider, ICo, a femtopenny per packet for handling internet traffic. ACo's CFO, affectionately dubbed Peering Tom by his colleagues, has negotiated a deal between ACo and BCo for the free, unconstrained transfer of packets between the two companies. Routing information is exchanged between all four companies using BGP4.

5. [10 points]: After a single gateway is set up connecting ACo and BCo, ACo's bill from ICo drops drastically, while BCo's costs increase. Which of the following are possible explanations?

(Circle True or False for each choice.)

A. True / False ACo has a transit relationship with ICo, while BCo has a peering relationship.
False. A and B have a peering relationship. I transits packets for A, B, and C.

B. True / False ACo and BCo have set MED attributes differently.
False. This can increase traffic from A to B, or the other way around, but has no effect on C.

C. True / False BCo advertises its connection to ICo and elsewhere, while ACo only advertises connections within ACo itself.
True. If B does this, then I may decide to route packets through B to A.

D. True / False ACo uses distance-vector routing while BCo uses path-vector routing protocols.
False. The routing policy is independent of the routing algorithm.

E. True / False ACo sends more packets to BCo than BCo sends to ACo.
False. That has no impact on the bill for A and B from I.

6. [10 points]: Which of the following describe the BGP4 connection between the gateways at ACo and BCo?

(Circle True or False for each choice.)

A. True / False It uses TCP.
True. BPG uses TCP has its transport protocol.

B. True / False It uses iBGP rather than eBGP.
False. iBPG is for within a domain, while eBGP is for between domains. The question is about between domains.

C. True / False It carries route updates.
True. BPG4 sends route advertisements over the BGP4 connection.

D. True / False It carries all data packets exchanged directly between hosts at ACo and BCo.
False. BGP4 connections are not used to exchange data packets, only BPG4 protocol packets.

E. True / False It carries keep-alive packets.
True. The BGP4 protocol has keep-alive packets, which are sent over the BPG4 connection.

Name:

II File System for Dummies

Mystified by the complexity of NFS, Moon Microsystems guru Jill Boy decides to implement a simple alternative she calls File System for Dummies, or FSD. She implements FSD in two pieces:

A. An FSD server, implemented as a 2-page Python application. The server may be run by any user, and responds to FSD requests on port 11111. Each request corresponds exactly to a UNIX file system call (e.g. `read`, `write`, `open`, `close`, or `create`) and returns just the information returned by that call (status, integer file descriptor, data, etc).

B. An FSD client library, which can be linked together with various applications to substitute Jill's FSD implementations of file system calls like `open`, `read`, and `write` for their UNIX counterparts. For clarity, we refer to the FSD versions of these procedures as `fsd_open`, etc.

Jill's client library uses the standard UNIX calls to access local files, but uses names of the form

/`fsd`/*hostname*/*apath*

to access the file whose absolute path name is */apath* on the host named *hostname*. Her library procedures recognize operations involving remote files (e.g.

```
fsd_open("/fsd/csail.mit.edu/foobar", READONLY)
```

and translates them to RPC requests on port 1111 of the appropriate host, using the filename on that host (e.g.

```
RPC("/fsd/csail.mit.edu/foobar", "open", "/foobar", READONLY)).
```

The RPC call causes the corresponding UNIX command (e.g.,

```
open("/foobar", READONLY))
```

to be executed on the remote host, and the results (e.g. a file descriptor) to be returned as the result of the `RPC` call. Jill's server code catches errors in the processing of each request, and returns `ERROR` from the `RPC` call on remote errors.

Figure 1 on page 6 describes pseudocode for Version 1 of Jill's FSD client library. The `RPC` calls in the code relay simple RPC commands to the server, using *exactly-once* semantics. Note that no data caching is done either by the server or client library.

Name:

```
// Map FSD handles to host names, remote handles:
string handle_to_host_table[1000];     // initialized to UNUSED
int handle_to_rhandle_table[1000];     // handle translation table

procedure fsd_open(string name, int mode)
  integer handle = find_unused_handle();

  if name begins with "/fsd/" then {
     host = extract_host_name(name);
     filename = extract_remote_filename(name);

     // returns file handle on remote server, or ERROR
     rhandle = RPC(host, "open", filename, mode);
     }
  else {
     host = "";
     rhandle = open(name, mode);
     }

  if rhandle == ERROR then return ERROR;

  handle_to_rhandle_table[handle] = rhandle;
  handle_to_host_table[handle] = host;
  return handle;

procedure fsd_read(int handle, string buffer, int nbytes)

   host = handle_to_host_table[handle];
   rhandle = handle_to_rhandle_table[handle];

   if host == "" then return read(rhandle, buffer, nbytes);

   // The following call sets "result" to the return value from
   // the read(...) on the remote host, and copies data read into buffer:
   result, buffer = RPC(host, "read", rhandle, nbytes);

   return result;

procedure fsd_close(int handle)
   host = handle_to_host_table[handle];
   rhandle= handle_to_rhandle_table[handle];
   handle_to_rhandle_table[handle] = UNUSED;

   if host = "" then return close(rhandle);
   else return RPC(host, "close", rhandle);
```

Figure 1: Pseudocode for FSD client library, Version 1

7. [4 points]: What does the above code indicate via an empty string (`""`) in an entry of handle_to_host_table?
(Circle the BEST answer)

A. An unused entry of the table.
Incorrect, UNUSED has its own special value.

B. An open file on the client host machine.
Correct, the empty string indicates an open file on the client host machine. The library saves `""` *in the handle_to_host_table if the pathname does not begin with* `"/fsd/"` *which means it is not a remote FSD file.*

C. An end-of-file condition on an open file.
Incorrect, `""` *has nothing to do with an end of file condition.*

D. An error condition.
Incorrect, `""` *has nothing to do with an error condition.*

E. None of the above.
Incorrect. B is the correct answer.

Mini Malcode, an intern assigned to Jill, proposes that the above code be simplified by eliminating the `handle_to_rhandle_table` and simply returning the untranslated handles returned by `open` on the remote or local machines. Mini implements her simplified client library, making appropriate changes to each `fsd_` call, and tries it on several test programs.

8. [12 points]: Which of the following test programs will continue to work after Mini's simplification? Mark TRUE if the program will work, or FALSE if it is likely to encounter bugs.
(Circle True or False for each choice.)

A. True / False A program that reads a single, local file.
True: The FSD client library will use the local file descriptor as the FSD file handle.

B. True / False A program that reads a single remote file.
True: The FSD client library will use the remote file descriptor as the FSD file handle.

C. True / False A program that reads and writes many local files.
True: The FSD client library will use the local file descriptors as the FSD file handles.

D. True / False A program that reads and writes several files from a single remote FSD server.
True: The FSD client library will use the remote file descriptors as the FSD file handles.

E. True / False A program that reads many files from different remote FSD servers.
False: The FSD client library will use the remote file descriptors as the FSD file handles. This will cause bugs because the different FSD servers may return identical file descriptors. If two servers return the same file desciptor for different files, the client library will incorrectly assume the file handle maps to the remote host corresponding to the second open.

Name:

F. True / False A program that reads several local files as well as several files from a single remote FSD server.
False: Like the bug in Question 8E, the local file descriptors may conflict with the remote file descriptors and cause buggy file handle aliasing in the FSD client library.

9. [16 points]: Jill rejects Mini's suggestions, insisting on Version 1 code shown above. She is asked by marketing for a comparison between FSD and NFS. Complete the following table comparing NFS to FSD, by circling yes or no under each of NFS and FSD for each statement:

Statement	**NFS**		**FSD**	
remote handles include inode numbers *NFS: Yes, inode numbers help NFS server remain stateless* *FSD: No, FSD handles include file descriptors, not inode numbers*	Yes	No	Yes	No
read, write calls are idempotent *NFS: Yes, the RPCs msg content alone is sufficient for the server to service the call.* *FSD: No, the FSD server uses file descriptors as state. If the server crashes, it loses state like file descriptor offsets which are not included in the FSD RPCs.*	Yes	No	Yes	No
can continue reading an open file after deletion (e.g. by program on remote FSD host) *NFS: No, an "open" NFS file does not prevent the server's file system from deleting the file.* *FSD: Yes, since the FSD server actually opens the file on the server, the kernel increases the file's reference count, so the file system will keep the file around while the remote client has the file open.*	Yes	No	Yes	No
requires mounting remote file systems prior to use *NFS: Yes, the NFS client needs the remote file system's root directory inode number so that it can boot strap LOOKUP RPCs.* *FSD: No, FSD does not need to mount remote FSD file systems. The client library just intercepts opens and redirects them to the FSD server; FSD does not need to do any bootstraping.*	Yes	No	Yes	No

Convinced by Moon's networking experts that a much simpler RPC package promising *at-least-once* rather than *exactly-once* semantics will save money, Jill substitutes the simpler RPC framework and tries it out. Although the new (Version 2) FSD works most of the time, Jill finds that an `fsd_read` sometimes returns the wrong data; she asks you to help. You trace the problem to multiple executions of a single RPC request by the server, and are considering various suggestions for eliminating the bug while continuing to use the new RPC package. Among the additions you are considering are

- A response cache on the client, sufficient to detect identical requests and returning a cached result for duplicates without re-sending the request to the server;
- A response cache on the server, sufficient to detect identical requests and returning a cached result for duplicates without re-executing them;
- A monotonically increasing *sequence number* (nonce) added to each RPC request, making otherwise identical requests distinct.

10. [7 points]: Which of the following changes would you suggest to address the problem introduced by the *at-least-once* RPC semantics?

(Circle the BEST answer)

Client caching will not resolve the problem of duplicate requests arriving at the server. Adding a cache to detect duplicate requests on the server will eliminate duplicates but changes the semantics. Consider an application that reads a file by repeatedly issuing `read(host, ''read'', handle, 4096)`*: the cache will respond with the same answer every time, fooling the client into thinking that the file is just the same 4096 bytes repeated over and over. Adding a sequence number to each request changes nothing by itself, however it does allow a cache to distinguish between duplicate and similar requests.*

A. Response cache on client.

B. Response cache on server.

C. Sequence numbers in RPC requests.

D. Response cache on client AND sequence numbers.

E. Response cache on server AND sequence numbers.
Correct, sequence numbers help the server distinguish repeat requests and the response cache allows the server to send a repeat response without the side-effects of repeating the read system call.

F. Response caches on both client and server.

Name:

III RaidCo

RaidCo is a company that makes pin-compatible hard disk replacements using tiny, chip-sized hard disks ("microdrives") that have become available cheaply. Each RaidCo product behaves like a hard disk, supporting the operations

```
error = read(nblocks, starting_block_number, buffer_address)
error = write(nblocks, starting_block_number, buffer_address)
```

to read or write an integral number of consecutive blocks from or to the disk array. Each operation returns a status word indicating whether an error has occurred.

RaidCo builds each of its disk products using twelve tiny, identical microdrives configured as a RAID system. A team of ace 6.033 students designed RaidCo's system, and they did a flawless job of implementing six different RaidCo disk models. Each model uses identical hardware (including a processor and the twelve microdrives), but the models use different levels of RAID in their implementation and offer varying block sizes and performance characteristics to the customer. Note that the RAID systems' block sizes are not necessarily the same as the sector size of the component microdrives.

The models are

- **R0** sector-level striping across all twelve microdrives, no redundancy/error correction
- **R1** six pairs of two mirrored microdrives, no striping
- **R2** 12-microdrive RAID 2 (bit-level striping, error detection, and error correction); microdrive's internal sector-level error detection is disabled.
- **R3** 12-microdrive RAID 3 (sector-level striping and error correction)
- **R4** 12-microdrive RAID 4 (no striping, dedicated parity disk)
- **R5** 12-microdrive RAID 5 (no striping, distributed parity)

The microdrives each conform to the same read/write API sketched above, each microdrive providing 100K sectors of 1K bytes each, and offering a constant 10 ms seek time and a read/write bandwidth of 100 megabytes/second; thus the entire 100MB of data on a microdrive can be fetched using a single read operation in one second. The RaidCo products do no caching or buffering: each read or write involves actual data transfer to or from the involved microdrives. Since the microdrives have constant seek time, the RaidCo products do not use RAID optimizations for seek time.

As good as the 6.033 students were at programming, they unfortunately left the documentation unfinished. Your job is to complete the following table, showing certain specifications for each model drive - i.e., the size and performance parameters of the API supported by each RAID system. Entries assume error-free operation, and ignore transfer times that are small compared to seek times encountered.

Name:

11. [25 points]: Complete the following table:

	R0	**R1**	**R2**	**R3**	**R4**	**R5**
Block Size (KB) exposed to `read`/`write`	1KB	1KB	8KB	11KB	1KB	1KB
# of Blocks (K)	1200K	600K	100K	100K	1100K	1100K
Max Time for a single 100MB read (sec)	1/12 s	1/2s	1/8s	1/11s	1 s	1 s
Time for 1-block write (ms)	10ms	10ms	10ms	10ms	20ms	20ms
typical # microdrives involved in 1-block read	1	1	12	11	1	1
typical # microdrives involved in 2-block read	2	2	12	11	1	1
typical # microdrives involved in 2-block write	2	2	12	12	2	3

Block size times # of blocks is the capacity of the array. Max time for a single 100MB read is a measure of the data striping/redundancy. Time for a 1-block write is based on whether the array must do a read-modify-write. Typical # of microdrives involved in a 1-block read is based on block size. The # of microdrives involved in multi-block reads and writes differs from single-block due to striping (data and parity).

R0 has no parity disks or redundancy; hence all 100K sectors of each of the 12 microdrives constitute blocks, totaling **1200K** blocks The 12-way parallelism allows 100MB to be read in 1/12 the time to read a single microdrive, or 1/12 second. A single-block read involves a single sector of a single microdrive, requiring one 10-ms seek time. A 2-block read or write is striped across **2 disks**.

R1 has 2X redundancy, for **600K total blocks**. This allows twice the read bandwidth of a microdrive, so a 100MB read takes **1/2 second**. A 1-block read accesses **one** of the 2 copies from a single sector of a single microdrive; a 2-sector read can be optimized slightly by reading a single block from each of two disks, involving two disks. The mirroring requires writing two microdrives on any write.

Name:

R2 uses bit-level error detection and correction, for groups of bits spread over the 12 microdrives. The 12 bits of a group must be partitioned into D data and P parity bits, such that $2^P \geq D + P + 1$ to allow single-bit error detection and correction within each group, since there need to be sufficient distinct combinations of parity bits to encode no error as well as an error in any of the D+P bits. This requires 4 parity disks, leaving 8 data disks. The disks are thus organized as **100K 8KB blocks**, each comprising 8 data sectors and 4 parity sectors. The 8x parallelism afforded by the striping allows a 100MB read in **1/8 second**. Access to even a single 8K block involves all **12 disks**, as the parity disks must be read for error detection (the microdrive-level error detection is disabled for R2).

R3 stripes data across 11 disks, with a single parity disk; the data is organized into **100K blocks** of 11KB, each comprising a single sector of each microdrive. The 11-way striping gives 11x read parallelism, or **1/11 second** for a 100MB read. A single 11KB block can be written via a single write to each of the 12 disks, requiring a single seek time (**10ms**). A read requires reading each of the **11 data disks**, while a write involves all **12 disks** since the parity must be rewritten as well as the data.

R4 uses 11 data disks and a single dedicated parity disk, for a total of **1100K blocks** of **1KB**, each corresponding to a single sector of a microdrive. No striping means 1 second for a 100MB read. A 1-block write requires updating the parity disk, which can be done by a read followed by a write and takes **20 ms**. A 2-block write involves writing the data disk (only 1; no striping) and the parity disk, thus **two disks**.

R5's distribution of parity causes the common case of a 2-block write to involve a single data disk plus two different parity disks, for a total of **3 disks**. On a few 2-block writes, one of the parity blocks will reside on the same disk as the data – involving only two disks.

End of Quiz II

Name: