DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033—Computer System Engineering | Handout 10—March 9, 1995

Last Year's Spring 1994 QUIZ #1.—Solutions

157 students took this quiz.

The median grade (half the class got better, half got worse) was 75 points.

Score

```
26--30      * (1)
31--35      ** (2)
36--40      **** (4)
41--45      ******* (7)
46--50      **** (4)
51--55      ********* (9)
56--60      *************** (15)
61--65      *********** (11)
66--70      ********** (10)
71--75      ****************** (18)
76--80      ******************** (20)
81--85      ******************** (20)
86--90      ************************* (25)
91--95      ********** (10)
96--100     * (1)
```

 Name ______________________

1. *(10 points)* According to the paper by Schroeder et al., "Except during reconfiguration, Autonet never discards packets." (p. 1319, col 2). Louis Reasoner takes this statement to mean that if he is careful never to launch a packet during a reconfiguration, he can dispense with end-to-end verifications and simply assume that all packets will be delivered. Set Louis straight by pointing out two substantially different situations that may occur and that require that the sender verify that the intended recipient got the packet, even in Autonet.

Situation 1. A reconfiguration might occur after the sender launches a packet, but before the Autonet delivers it. Similarly, a hardware fault may occur along the path just as the packet is coming by.

Situation 2. The target host, which is required to accept the packet whether it is prepared to do so or not, may discard it for lack of buffers.

Situation 3. The Autonet hardware or software may experience a fault that results in the packet being destroyed or damaged so badly that it isn't clear where to deliver it any more.

2. *(5 points)* The X Window system deals with both big-endian and little-endian clients by providing two different network ports. Big-endian clients send requests and data to one port, while little-endian clients send requests and data to the other. The server may, of course, be implemented on either a big-endian or a little-endian machine. This approach is unusual. Most network servers provide only one network port, and require that all data be presented at that port in little-endian form. Explain the advantage of the structure chosen by X, as compared with the usual structure.

The two-port structure minimizes byte-swapping for all possible combinations of big-endian and little-endian clients and servers, resulting in better overall performance. Specifically, if both client and server are same-endian, then there will be no byte-swapping. If they are different-endian, then multi-byte data such as long integers will be swapped exactly once.

The alternative, in which the server accepts only little-endian data, requires a big-endian client to byte-swap its data into little-endian form before sending it, even if the server happens to also be big-endian and will have to byte-swap it again. This additional byte-swap might result in a substantial performance decrease.

3. *(25 points)* For each of the following statements, circle whether it is True or False, and briefly explain. An answer without an explanation will receive no credit. And if you give a good enough explanation you may be able to get credit even if you say False to a point we thought was True, or vice-versa. *But keep it brief!*

a. One advantage of a microkernel over a monolithic kernel is that it reduces the load on the Translation Lookaside Buffer (TLB), and thereby increases its hit rate and its consequent effect on performance. FALSE

When a microkernel is used, there will be more context switches per unit time than with a monolithic kernel. If anything, this would lead one to expect more pages in use, more frequent TLB invalidations, and overall more, rather than less, load on the TLB

b. Ethernet is not very scalable because a centralized mechanism is used to control network contention. FALSE

Contention control in the Ethernet is actually quite decentralized--each transceiver independently plays its own part. The lack of scalability of the Ethernet comes from its broadcast nature, and the resulting need for every signal to propagate to every point on the net, the speed of light being the fundamental limiting factor.

c. The primary way that hierarchy reduces complexity is that it reduces the size of individual modules. FALSE

Hierarchy has no direct bearing on module size. Instead, it provides structure for the interconnections between modules. Its primary impact on complexity comes from constructing stable subassemblies and thus reducing the number of modules that interact at each assembly point.

d. The primary way that modularity reduces complexity is that it eliminates incommensurate scaling. FALSE

Modularity has no effect on incommensurate scaling. Its impact on complexity comes from reducing the number of interconnections.

e. As linear feature size goes down, the number of components that can be placed on a chip goes up with the inverse square. But overall computation capacity improves with the inverse cube of feature size. TRUE

The speed of the components also increases with the inverse of feature size. Assuming that computation capacity is roughly measured by the product of number of components and their speed, the computation capacity then increases with the inverse cube of feature size.

One can plausibly argue that it is hard to actually realize this increased capacity, since more components will require more interconnection and coordination overhead.

4. *(25 points)* Ben Bitdiddle has written a "style checker" intended to uncover writing problems in technical papers. The program picks up one sentence at a time, computes very hard for a while to parse it into nouns, verbs, etc., and then looks up the resulting pattern in an enormous database of linguistic rules. The database was so large that it was necessary to place it on a remote server and do the lookup with a remote procedure call.

Ben is distressed to find that the RPC's have such a long latency that his style checker runs much more slowly than he hoped. Since he is taking 6.033 he wonders if adding multiple threads to the client could speed up his program.

a. Ben's checker is running on a single-processor workstation. Explain how multiple client threads could reduce the time to analyze a technical paper.

> Assuming that for purposes of style analysis sentences are independent, while one client thread is waiting for a response from the remote server a different client thread can be parsing a different sentence. It may also be possible for a thread to initiate a parallel RPC even though the response from an earlier RPC has not returned, in which case still more client threads can make progress on other sentences. The first mechanism overlaps parsing with RPC latency, whether caused by the network or the server. The second mechanism overlaps network latency with server activity.

b. Ben implements a multithreaded style checker, and runs a series of experiments with various numbers of threads. He finds that performance does indeed improve when he adds a second thread, and again when he adds a third. But he finds that as he adds more and more threads the additional performance improvement diminishes, and finally adding more threads leads to reduced performance. Give an explanation for this behavior.

> The second thread exploits some, but not necessarily all opportunities for overlapping. The third thread exploits some more. But each additional thread finds fewer opportunities for overlapping, and eventually client parsing, network latency, and server time are overlapped to the maximum extent possible. If still more threads are added they can't help, but they can begin interfering with one another, perhaps because of increased scheduling and thread switching overhead, mutual exclusion and coordination overhead, or increasing virtual memory traffic for the extra thread stacks.

c. Suggest a way of improving the style-checker's performance without introducing threads. (Ben is allowed to change only the client.)

> 1. The client could cache the result of a request to the server, on the chance that a writer will tend to use the same style patterns over and over.
>
> 2. The client could parse several sentences, then send several requests to the server at once by constructing a special client RPC stub that fires off several RPC request packets and then waits till all the responses have returned.
>
> 3. If available, use a non-blocking RPC, and work on the next sentence while awaiting results on the previous sentence.
>
> 4. Use multiple processes (as contrasted with threads) in conjunction with shared memory to coordinate which process does which sentences.
>
> 5. Buy a faster processor for the client.

5. *(25 points)* Suppose the longest packet you can transmit across the Internet can contain 480 bytes of useful data, you are using a lock-step end-to-end protocol, and you are sending data from Boston to California. You have measured the round-trip delay and found that it is about 100 milliseconds. (A lock-step protocol is one where you must wait for the recipient to send an acknowledgement of your previous packet before you can send the next packet.)

a. If there are no lost packets, estimate the maximum data rate you can achieve.

```
(480 bytes)/(0.1 seconds) = 4,800 bytes/second
```

b. Unfortunately, 1% of the packets are getting lost. So you install a resend timer, set to 1000 ms. Estimate the data rate you now expect to achieve.

To send 100 blocks requires that 200 packets traverse the network, of which an average of 2 will be lost, so 2 blocks will require retransmission. The average time to send 100 blocks (48,000 bytes) is thus

```
(98 blocks) x (0.1 seconds) = 9.8 seconds
+(2 blocks) x (1.1 seconds) = 2.2 seconds
                      total = 12.0 seconds; average time is 120 ms.
                     480/.12 = 4,000 bytes/second
```

(This approximation ignores the possibility that 2% of the retransmitted blocks will also be lost. A more precise answer (about 3990 bytes/sec.) isn't worth the effort because the 1% packet loss figure probably isn't that accurate anyway.)

Note that you must average the transmission times, not the transmission rates. The reason is that although 2% of the blocks are retransmitted, much more than 2% of the time is spent on retransmitted blocks.

c. On Tuesdays the phone company routes some westward-bound packets via satellite link, and we notice that 50% of the round trips now take exactly 100 extra milliseconds. What effect does this delay have on the overall data rate when the resend timer is not in use. (Assume the network does not lose any packets.)

If we send 100 blocks on a Tuesday, an average of 50 of them will require 100 ms., and the other 50 will require 200 ms. Total time for 100 blocks is

```
50 x 100 ms =  5 sec
50 x 200 ms = 10 sec
              15 sec, or 150 ms/block
(480 bytes)/(.150 sec) = 3200 bytes/second
```

Again, we must average times, not data rates.

d. Ben turns on the resend timer, but since he hadn't heard about the satellite delays he sets it to 150 ms. What now is the data rate on Tuesdays?(Again, assume the network does not lose any packets.)

If the sender is willing to accept the first packet that is responsive to a request, then the data rate does not change; there are a number of duplicate packets flying around but they always arrive at times when the client or the server are waiting for something else and can discard or rerespond without interfering with useful activity. Rate = 3200 bytes/second.

If the sender insists on ignoring responses that don't match with specific requests, the calculation is harder. The time to send one block will be 100 ms, or 250 ms, or 400 ms, or 550 ms., etc., depending on how many tries are needed to get an undelayed round trip.

At this point, one can either approximate by taking the first few terms of the series or one can compute the exact series sum. Here is an elegant way to get the exact sum:

Let t be the expected delay.
There's a 50% chance the round trip takes 100 ms.
There's a 50% chance a retry will be started after 150 ms.
The retry has the same expected delay, starting from when it was sent.
Therefore, working in milliseconds:

$$t = 0.5 \times 100 + 0.5 \times (150 + t)$$

Solving for t:

$$2t = 100 + 150 + t$$
$$t = 250 \text{ ms}$$

The data rate is (480 bytes)/(0.25 sec) = 1920 bytes/sec

e. Usually, when discussing end-to-end data rate across a network, the first parameter one hears is the data rate of the slowest link in the network. Why wasn't that parameter needed to answer parts a-d of this question?

The data rate of the slowest link is one of the things that contributes to the round trip latency, so it is already included in the given parameters.

6. *(10 points)* OutTel corporation has been delivering j486 microprocessors to the computer industry for some time, and Metoo systems has decided to get into the act by building a microprocessor called the "clone486+", which differs from the j486 by providing twice as many processor registers.

Metoo has simulated many programs and concluded that having twice as many processor registers reduces the number of loads and stores to memory by an average of 30%, and thus improves performance, assuming of course that all programs are recompiled to take advantage of the extra registers.

The most popular operating system for the j486 is microkernel—based. As suggested, the hoped-for performance improvement won't be realized unless all applications, both client and server, and the microkernel itself are recompiled. Point out one other reason why Metoo may find the performance improvement to be less than their simulations of individual programs predicts. If you think of more than one reason, choose the one that is likely to reduce performance the most.

Some likely candidates:

1. On every kernel trap it will be necessary for the kernel to save and restore all the extra registers, which will slow things down. Since the most popular operating system is organized as a micro-kernel, there will be a lot of context switches, so this effect will be significant.

2. The bottleneck in the system may not be memory access rate, but somewhere else such as disk access, the network, or even the processor itself.

3. Many of the targets of loads and stores are likely to be in the processor cache, so reducing the number of loads and stores won't save as much time as Metoo expected.