DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.033—Computer System Engineering Handout #25—April 5, 1994

QUIZ #2—Solutions

This quiz contains five problems whose relative weights are indicated below. Please place each answer on the page with the question and hand in this booklet at the end of the examination.

Put your name on this cover sheet AND at the bottom of each page of this booklet. The booklet may be separated for grading so that each problem can be evaluated by one grader. Be sure that you don't put part of the answer to a problem on the back of a sheet for another problem!

Beware: some parts of some problems are much harder than others. Read them all through first and attack them in the order that allows you to make the most progress. Write down any assumptions and show your work, or you risk losing partial credit. Be neat. If we can't figure out your answer we can't give you credit. For problems that require a numerical answer, be sure to indicate the units of your answer.

Median score: 75

 Name _____

Problem 1. *(25 points)* Lucifer is determined to figure out Alice's password by a brute-force attack. From watching her log in he knows that her password is eight characters long and all lower-case letters. He sets out to try all possible combinations of eight lower-case letters.

a. Assuming he has to try about half the possibilities before he runs across the right one, one trial can be done in one machine cycle, and he has a 60 MHz computer available, about how long will the project take?

Since there are 26 lower-case letters, there are 26^8 passwords. It will take

$$T1 = \frac{26^8}{2 \cdot 60 \cdot 10^6} = \frac{(2.6)^8}{1.2} = 1740.22$$ seconds, or about half an hour.

b. How long will it take if Alice chooses an eight-character password that includes upper- and lower-case letters, numbers, and special characters? There are 78 characters available for her to choose from.

This time it will take $$T2 = \frac{78^8}{2 \cdot 60 \cdot 10^6} = \frac{(7.8)^8}{1.2} = 1.14 \cdot 10^7$$ sec, or about 132 days.

c. Suppose processors continue to get faster, improving by a factor of three every two years. How long will it be until Alice's new password can be cracked as easily as her old one?

The ratio of the numbers of passwords is $78^8/26^8 = 3^8$. Thus eight triplings of performance are required, and that will take 16 years.

An equivalent approach is to first calculate how fast a processor would be needed to crack the password in the same time, then calculate how many years it will take until processors are that fast.

A legalistic reading of the question yields the answer, "Never. No matter how fast the processor gets, it will always take 3^8 times as long to crack the new password as it takes to crack the old one."

 Name _____

Problem 2. *(24 points)* Louis Reasoner is fascinated with the discovery that some encryption algorithms are *commutative*.

For this problem we use the usual notation: A message M that is encrypted with key k is written $\{M\}^k$. A commutative scheme has the interesting property that for every message and every pair of keys k1 and k2, $\{\{M\}^{k1}\}^{k2} = \{\{M\}^{k2}\}^{k1}$. That is, you get the same result no matter which order two encryptions with different keys are done.

Louis has proposed that Alice, in San Francisco, and Bob, in Boston, use the following scheme for secure delivery of messages between their computers, which are connected via the Internet:

Alice chooses a never-before-used (nonce) encryption key, Ka, encrypts her message M, and sends the result, $\{M\}^{Ka}$, to Bob. Bob chooses another nonce encryption key, Kb, encrypts the already-encrypted message to produce $\{\{M\}^{Ka}\}^{Kb}$ and sends the doubly-encrypted result back to Alice. By commutativity, this message is identical to $\{\{M\}^{Kb}\}^{Ka}$, which is a message that Alice can decrypt with her nonce Ka. She does so, revealing $\{M\}^{Kb}$. This result she sends back to Bob, who can now decrypt it with his nonce Kb to reveal M.

The appealing thing about this scheme is that Alice and Bob did not have to agree on a secret key in advance. Louis calls this the "No-Prior-Agreement" protocol.

a. Is it possible for a passive intruder (that is, one who just listens to the encrypted messages) to discover M? If so, describe how. If not, explain why not.

No. Assuming the encryption algorithm itself is reasonably good, all that a passive intruder will see is packets encrypted under unknown, never-before-used keys.

b. Is it possible for an active intruder (that is, one who can also insert, delete, or replay messages) to discover M? If so, describe how. If not, explain why not.

Yes. Lucifer need merely trap the first packet $\{M\}^{Ka}$ and send it back to Alice; according to the protocol she will decrypt it and send the plaintext to Bob, which Lucifer can also intercept. If Lucifer wants to ensure that Alice doesn't become suspicious, he can encipher the packet in a key he knows before sending it back to Alice. He can similarly mimic the protocol with Bob to ensure that Bob ends up with M and is no wiser about Lucifer's intervention. The problem with this protocol is that it provides no authentication whatever.

Problem 3. *(10 points)* One way of speeding up a name resolution system is to implement a cache that remembers recently looked-up {name, object} pairs. *Synonyms* are multiple names for the same object. What problems do synonyms pose for cache designers?

The trouble with synonyms is that if *name1* and *name2* refer to the same object, *{name1, object}* is in the cache, and someone refers to *name2*, we are likely to not realize that we already have *object* in the cache, and we will look it up again and add it to the cache as the pair *{name2, object}*. We have wasted time, we have used up valuable cache space, and we have set ourselves up for a disaster if the system permits modifying objects by name. A modification to the copy of *object* associated with *name1* should be propagated to the copy of *object* associated with *name2*.

One can invent various mechanisms to allow synonyms in caches. For example, if each object has a unique ID one could change the cache to hold {name, object, UID} triples. Then, whenever someone modifies an object by name, it is possible to search the cache for any other entries that have the same value for UID in their third field and either modify or invalidate those copies.

Problem 4. *(15 points)* Alyssa is trying to organize her notes on virtual memory systems, and it occurred to her that virtual memory systems can usefully be analyzed as naming systems. She has gone through her 6.033 notes and made a numbered list of some technical terms from the lectures on naming systems; that list is on the right, below. She then listed some mechanisms found in virtual memory systems on the left. But she isn't sure which naming concept goes with which mechanism. Help Alyssa out by placing in each blank on the left the number of the term on the right that best applies to this mechanism. Hint: no number needs to be used twice.

6 a. page map

5 b. virtual address

4 c. physical address

7 d. a TLB entry

3 e. the processor register that points to the page map

1. Search path
2. Naming network
3. Closure
4. Object
5. Name
6. Context
7. None of the above

Note that although a TLB entry is involved in a kind of search, it is not itself a "search path", which is a list of places to search.

Problem 5. *(25 points)* The network level of the Internet names hosts with 4-byte *host addresses*. These names are assigned regionally, with all hosts in one major region receiving host addresses for which the first byte is the same. A region is divided into sub-regions in which all host addresses have the same value for the second byte, and so on. This hierarchical naming scheme is used by the routing system to reduce the size of the tables it must maintain.

a. Explain briefly how hierarchical naming of host addresses might reduce the size of routing tables.

Hierarchical naming creates structural regularity that allows us to compress the size of the routing table. From a given point in the network, it is normally the case that the appropriate "next hop" in the network is the same for many destination addresses. When several such destinations happen to be in the same sub-sub-region, sub-region, or region, we can save space by recording the common "next hop" information once for all those addresses. The hierarchical naming makes it likely (but not certain--see below) that we can cleanly group up entire levels of the hierarchy and store a single route entry suitable to large chunks of the network without loss of information.

Note that having hierarchically organized names does not require that the routers themselves be organized hierarchically for this space reduction to be effective; it is sufficient if just the names are hierarchical. Thus there can be direct links between any two routers, even if they are from different regions, and we do not need to require that packets destined to a different region must go up three levels and then down three levels. If we did assume that the network's physical layout were hierarchical, then we would be guaranteed to find the opportunity for space compression that we currently can only utilize when it happens to be available. Another way of thinking about this difference is that we are taking advantage of the hierarchical shape of the names to do a lossless compression. If the network were organized hierarchically, the compression would be greater. If the network is not hierarchical, but we compress as though it were, then we would have a lossy compression scheme, for which the losses would show up as not knowing the best available routes. Curiously, everything would still work "correctly," except that the performance would probably be non-optimal.

b. Ben Bitdiddle is thinking about designing a system for wireless attachment of portable computers to the Internet. He has decided that the basic architecture should consist of a number of fixed base stations, each of which will communicate by radio with those portable computers in its area. He plans to make each base station a sub-sub-region, using the third byte of the Internet address to identify it. Alyssa points out that this plan limits the number of portables that can simultaneously work with one base station. The base station uses up one of the host addresses for its sub-sub-region. How many portables can work with one base station?

A byte being eight bits, there are 256 addresses available. One of these is used up by the sub-sub-region base station, leaving 255 for portables. (Or, if the network reserves one address for broadcast, 254.)

c. Ben proposes not to permanently assign host addresses to portable computers. Instead, when a portable is switched on, it will ask some base station to dynamically assign a host address for it to use. When dynamic address assignment is used, what else has to happen in order for other computers to initiate communication with a portable?

In order for someone to initiate communication with the portable, some persistent name needs to be associated with the dynamically assigned host address. For example, the name service needs to be updated.

d. Ben wants his design to work with the portable computer mounted on the handlebars of his bicycle. The base stations he will use were designed for

cellular phones, so they have the ability at the physical transport level to transfer (or "handoff") a conversation from one base station to another as the signal strength changes. But in Ben's design a handoff means that the host address of the portable must change. Ben wants to be able to maintain connections through handoffs, so that he can transfer long files without disruption while cycling across town. His first idea is to modify the File Transfer Protocol to allow handoff. Whether or not he figures out how to do the modification, what is the main defect in this proposal?

Ignoring safety problems that may arise from Ben watching the computer display instead of traffic, there is a major abstraction violation here. Ben is trying to solve a network layer problem by tinkering with the end-to-end layer. As a result, he is going to have to track down every implementation of FTP he intends to talk to and modify it, and he is also going to have to modify all other end-to-end protocols he wants to use.

e. ***Briefly*** suggest a way of accomplishing handoff entirely within the network layer.

This one is challenging. One way to start is to have the portable, when it realizes it has been handed off, send a message via the network to the previous base station, asking it to forward packets to the new address. All outgoing packets from the portable should continue to claim to come from the old address, to avoid confusing correspondents. And when the portable does disconnect it would be polite of it to send another message to the earlier base station suggesting that it can discard the forwarding pointer and reassign the host address.

An alternate approach might be to create pseudo-sub-sub-regions in which portables can have permanently assigned addresses, and design a special forwarder that receives all packets addressed to a given pseudo-sub-sub-region. When a portable receives a dynamic host address assignment, its first project is to send a message to its pseudo-sub-sub-region forwarder telling it its current address.

We got into this problem to begin with by condensing routing tables to take advantage of the hierarchical nature of the addresses (and, as it happens, to reduce routing table update traffic in the network). If we reversed that decision, then a host address could be physically anywhere in the network, and it would not be necessary to change host addresses when there is a physical level handoff. So in the end we see that our hierarchical routing strategy has actually muddied the distinction between the physical and network layers.