Final Exam Sample Questions
Solutions

SAMPLE PROBLEMS

1. Consider a FAT-based (File Allocation Table) file system. Entries in the table are 16 bits wide. A user wants to install a disk with 131072 512-byte sectors.
   a. What is a potential problem?

      Each 16 byte entry in the table is an address of a sector on the disk. The OS can address 65,536 sectors, and the disk has more than that.

   b. Describe a solution to this problem and explain the trade-offs involved.
      Make each FAT entry access a logical sector that is 2 physical sectors. This trades increased internal fragmentation against maintaining the size of the FAT, and backward compatibility.
      Alternatively, you can increase the size of the FAT to be 131,072 17-bit entries, which increases the table size to 278,528 bytes (at least) and increases the complexity of the decoding process. It also ruins any backward compatibility.

2. Generally we've talked about each operating system component in isolation. This question asks you to think about ways in which they interoperate. For each pair of systems below, give a specific way that they interact (or that they could interact). Writing that the file system and I/O system interact because they both use the disk is not worth more than a point, and may be worth none. Writing that the file system and I/O system interact when they determine the mapping from logical blocks → physical blocks which impacts the size of file system structures, and the efficiency of the disk usage because larger logical blocks imply more internal fragmentation on the disk is a more complete answer.
   a. How does a demand paged, lazy loaded virtual memory system interact with the process scheduling and creation system?

      A demand paging system will be at its worst at process creation, with an empty address space. The two systems have to work together to pre-fetch a reasonable working set without overloading the paging system by bringing in the entire address space. If the VM system supports shared text (or copy on write), the two need to work together to take advantage of it.

   b. Name another way (not the example above) that the file system and the hard disk drivers in the I/O system interact.

      Feature coordination. If the disk implements interleaving, the file system implementation should not do its own interleaving. The result can be to make both systems worse that useless - actually detrimental. Similar negotiations have to be done for features like disk caching or bad block renaming.

3. On some computer, the clock interrupt handler needs 2 msec (including context switch overhead) per clock tick to execute, and the clock runs at 75 Hz. What fraction of the CPU time is devoted to the clock?

   *The clock interrupt handler is run 75 times every second, which takes 75*2 = 150ms. This is 15% of the CPU time.*

4. List the terms that best describe each of the following:

   a. Operating system code executed when an asynchronous device signals the CPU

   Interrupt handler

   b. A type of disk arm scheduling policy

   FIFO, Shortest seek first (SSF), SCAN, C-SCAN, or elevator algorithm

5. If the TCP transmission window for unacked bytes is 1000, the one-way latency of a cross-country network link is 50 milliseconds, and the bandwidth of the link is 100 Megabits/second, then how long does it take TCP to transmit 100,000 bytes across the link? That is, how much time elapses from when the first byte is sent by the sender to when the sender *knows* that the last byte has been received by the receiver? You may assume that no packets are lost for this particular problem (but remember that TCP doesn't know that).

   A TCP transmission window size of 1000 implies that the sender can send 1000 bytes before having to wait for an ACK message from the receiver that will allow it to continue sending again. Assume the ACK is essentially 0 bytes long. Then the sequence of messages sent is:

   1. *1000 bytes from sender to receiver: requires 50 ms for first byte to get there and another 1000/(100 Mbps / 8 bytes/bit) secs for the rest of the 1000 bytes to get there after that.*
   2. *ACK msg from receiver to sender: requires 50 ms to get there. To send 100,000 bytes will require 100 round trips of this kind. So the total time required is:*

      *100 * (2 * 50 ms + 1000/(100,000,000/8)) = 100 * (100 ms + 0.08 ms)*
      = 10008 ms
      = 10.008 seconds

6. Security

a. How would you use public-key encryption to implement secure remote procedure call (RPC) between an arbitrary client A and server B. Assume that A knows B's IP address and public key but that B does *not* know A's IP address and public key. Do not assume the existence of a "public key" server. "Secure RPC" means that no one other than A and B can understand the contents of either the request message or the response message. Please use a combination of English and pseudo-code of the form (*data*)^K to indicate that *data* is encrypted/decrypted using key K to answer the question.

A must send B its IP address and public key so that B can talk back to A. To prevent anyone from being able to understand the RPC request – suppose it is f(x), A encrypts everything with B's public key:

(f(x), IP-A, KA-public)^KB-public

B decrypts this message with it's private key:

((f(x), IP-A, KA-public)^KB-public)^KB-private ➔ f(x), IP-A, KA-public

B can now execute the requested f(x). Suppose that returns a value y. B then sends back the return value by encrypting it with A's public key so that only A can understand the returned message:

(y)^KA-public

B sends this message to IP-A. A decrypts the return message using its private key:

((y)^KA-public)^KA-private ➔ y

b. Assume that there is a "public key" server, S, that stores IP addresses and public keys for everyone, including A and B. Suppose that both A and B know the IP address and public key for S, but do not know the IP address and public key of anyone else. How could A reach B and how could B authenticate A? By authenticate I mean: how could B be sure that an RPC request came from the client that it claims to have come from?

Since A doesn't know B's IP or public key it must get them from the public key server. It does that as in part (a), except that the RPC now goes to the public key server. Note: since the public key server only stores "public" information, we might be tempted to use insecure RPC when talking to it. However, that would offer an additional opportunity for someone who is monitoring network traffic to try and "hijack" messages and replace them with altered ones.

In order to convince B that A is really the sender of an RPC, A must "sign" its RPC message with its private key and say something that B can verify, like its identify: "I'm A". When B uses A's public key to decrypt the message and gets a correctly formed message it will know that only A could have signed it. Note that A must still tell B who it is in its RPC request, so that B will know whose public key to ask for from the public key server.

So, the sequence of steps is:

1. *A ➔ S: ("Need info for B", "I'm A")^KS-public*
2. *S: decrypt and get info.*
3. *S ➔ A: (mbox-B, KB-public)^KA-public 4.*
4. *A: decrypt and use info.*
5. *A ➔ B: ((f(x), "I'm A")^KA-private, "I'm A")^KB-public*
6. *B ➔ S: ("Need info for A", "I'm B")^KS-public*
7. *S: decrypt and get info.*
8. *S ➔ B: (mbox-A, KA-public)^KB-public*
9. *B: decrypt and use info.*
10. *B: (((f(x), "I'm A")^KA-private, "I'm A")^KB-public)^KB-private*
11. *B: (((f(x), "I'm A")^KA-private)^KA-public*
12. *B: Check that decrypted message provided by A consists of a properly formed remote procedure call request and the expected declaration of identity: "I'm A".*
13. *B: f(x) ➔ y*
14. *B ➔ A: (y)^KA-public*
15. *A: decrypt*

Note that this solution is subject to replay attacks: someone could replay the message sent from A to B and B would assume that the same RPC is being called a second time.

Consider how you could solve this problem if you had to solve it.

c. Explain how an intruder who can break into the public key server could snoop on and alter all RPC traffic in the system described in (b) without any of the RPC clients and servers being able to tell.

If an intruder compromised the public key server, then the intruder can feed everyone false values for IP addresses and public keys. (We assume that the intruder has also managed to gain the private key of the public key server.) When a client A asks for the IP address and public key of a server B, the corrupted public key server returns a IP address that belongs to it and returns its own public key. A will then unwittingly send its RPC request to the intruder instead of to B. The intruder decrypts the message, reads it, perhaps alters it (e.g. replace f(x) with f(z)), and then sends it on to B as if it were A. When B later asks the intruder for information about A the intruder again returns information that points to itself instead of to the real A. So B will think the RPC request that the intruder forwarded is actually coming from A. The same "waylaying and forwarding" will happen on the RPC reply message.

d. Explain how you would modify the system to prevent the problem identified in (c). Be sure to state any assumptions you make.

If we are worried that someone has corrupted the public key server then we need to create additional sources/repositories of the information that the public key server was maintaining for the system. Suppose there were three public key servers, all running independently, so that an intruder who managed to corrupt one of them couldn't also thereby corrupt the other two.
Clients and servers now talk to all three public key servers when asking for information instead of just to one. They compare the information received from all three and check to see if it all matches up. If at most one public key server can be corrupted then clients and servers will see two correct values for each piece of information and one incorrect value. They discard the incorrect values and use the correct values to proceed to talk to each other.

7. In class, we discussed copy-on-write for memory pages shared among multiple processes. We cannot apply this same concept blindly to process creation using Unix fork(), but instead are forced to copy some parts immediately while other parts can be delayed.
   a. Knowing the components of general processes, which parts must be copied immediately, and which parts can be delayed and copied-on-write?

      This question refers to processes, not caching. When we create a new process, we must copy the stack space and registers, but need not copy the entire address space immediately. This is because often a call to fork creates a new address space into which a new process is immediately loaded, making the initial copy a waste of time and space because it is immediately overwritten. An example of this is when a command shell starts up a new process, first doing a fork to create a new copy of itself, but then replacing that copy with the new process which was executed on the command line.

   b. Why is copy-on-write potentially better than copying the entire process immediately upon creation?

      It can save time and space during the process creation (as mentioned above), avoiding the duplicate effort of making a copy, then immediately overwriting it.

8. Briefly describe the steps taken to read a block of data from the disk to the memory using DMA controlled I/O.

   1. *CPU programs the DMA controller by setting its registers so it knows what to transfer where.*
   2. *The DMA controller initiates the transfer by issuing a read request to the disk controller.*
   3. *The disk controller fetches the next word from its internal buffer and writes it to the memory.*
   4. *When the write is finished, the disk controller sends an acknowledgement to the DMA controller*
   5. *If there are more data to transfer (counter is greater than 0), then the DMA controller repeats steps 2 through 4. If all the data has been transferred, the DMA controller interrupts the CPU to let it know that the transfer is complete*

7. Explain what is symbolic link and list at least two of its drawbacks.

   Symbolic link is a way to achieve file sharing. When we want to share a file, we create a new file, or a link. This file contains the path name of the file that we want to share. Later when we access this link file, the operating system sees that the file being accessed is a link, so it can lookup the shared file and all accesses go to that file. Symbolic link is very useful in file sharing. But it also has some drawbacks. First, it needs some disk space for its i-node and data. Second, when we open a symbolic link, we have to first read the path name in that link, and then follow the link to the shared file. This needs some extra processing time. These two drawbacks are not present in a hard link. Both symbolic link and hard link have another problem, since a file may now have more than one path name, the backup of a file system must be done more carefully.