CS414 SP 2007 Assignment 6

1. Which of these is an example capability system, and which is an ACL-based approach?
(a) You give your friend a key to your apartment
(b) A fancy club has a list of approved guests
(c) Some dorms at Cornell have card-swipe access, where the magnetic code on the card is matched against a list of residents
(d) Your car has a parking permit, listing where you're allowed to park.

Answer:
a) Capability: The key is the capability for the right of opening the door, and can be transfered. There's no list of people with keys.
b) ACL: there's a list of who's authorized to enter
c) This is actually ACL in disguise; there's a list of people who are approved. The card isn't a capability, it's an authentication token.
d) Capability: The pass designates your rights, there's no master list of rights-holders.

2
(a) Discuss the strengths and weaknesses of implementing an access matrix using ACL that are associated with objects.

Answer: The strength of storing an access list with each object is the control that comes from storing the access privileges along with each object, thereby allowing the object to revoke or expand the access privileges in a localized manner. The weakness with associating access lists is the overhead of checking whether the requesting domain appears on the access list. This check would be expensive and needs to be performed every time the object is accessed.

(b) Discuss the strengths and weaknesses of implementing an access matrix using capabilities that are associated with domains.

Answer: Capabilities associated with domains provide substantial flexibility and faster access to objects. When a domain presents a capability, the system just needs to check the authenticity of the capability and that could be performed efficiently. Capabilities could also be passed around from one domain to another domain with great ease allowing for a system with a great amount of flexibility. However, the flexibility comes at the cost of a lack of control; revoking capabilities and restricting the flow of capabilities is a difficult task.

3. Buffer-overflow attacks can be avoided by adopting a better programming methodology or by using special hardware support. Discuss these solutions.

Answer: One form of hardware support that guarantees that a buffer overflow attack does not take place is to prevent the execution of code that is located in the stack segment of a process's address space. Recall that buffer-overflow attacks are performed by overflowing the buffer on a stack frame, overwriting the return address of the function, thereby jumping to another portion of the stack frame that contains malicious executable code, which had been placed there as a result of the buffer overflow. By preventing the execution of code from the stack segment, this problem is eliminated. Approaches that use a better programming methodology are typically built around the use of bounds-checking to guard against buffer overflows. Buffer overflows do not not occur in languages like Java where every array access is guaranteed to be within bounds through a software check. Such approaches require no hardware support but result in runtime costs associated with performing bounds-checking.

4. One mechanism for resisting replay attacks in password authentication is to use one-time passwords. A list of passwords is prepared, and once password[N] has been accepted, the server decrements N and prompts for password[N-1] next time. At N=0 a new list is needed. Outline a mechanism by which the users and the server need only remember one master password *mp* and have available locally a way to compute password[N] = f (*mp*, N). (Hint: make use of one-way hash functions)

Answer: Let g be an one-way hash function (e.g., MD5) and let $f(mp, N) = g^N(mp)$. Intuitively, the first password is given by running the hash function on *mp* N times. The second password is given by running N-1 times. Since the hash is one-way function, given any password in the sequence, it's easy to compute the previous one in the sequence, but impossible to compute the next one.

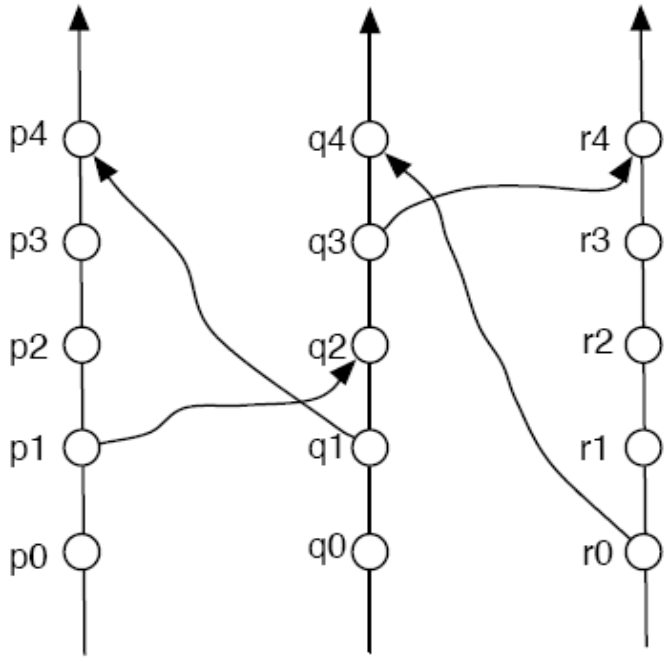
5. In any distributed coordination problem, it is necessary to be able to determine the order in which two events occurred. In a distributed system it is sometimes impossible to say which of two events occurred first. The happened-before relation is used in obtaining a total ordering of events in a distributed system. The happened-before relation (denoted by ->) is defined as follows: first, if A and B are events in the same process, and A was executed before B, then A -> B. Second, if A is the event of sending a message by one process and B is the event of receiving that message by another process, then A -> B. Third, if A -> B and B -> C then A -> C .

(a) In a centralized system with a single core processor, it is always possible to determine the order in which two events occurred, but in a distributed system this is not always possible. Why?

Answer: On a single-core machine, there is one clock, which orders events in the obvious way. In the distributed case,we have multiple clocks, and within the system there is no way to know exactly what the skew is, so no way to tell which happened 'first'.

(b) From the diagram below of relative time for three concurrent processes, what can you say about the relationship between q0 and p2? What can you say about the relationship between p0 and r4? (diagram

attached)



q0 and p2: Nothing. p0 -> r4.
Event A happens-before B if you can find a path from A to B either going down the world-line for one machine, or following the arrows of packets.

6 Consider the following failure model for faulty processors. Processors follow the protocol but might fail at unexpected points in time. When processors fail, they simply stop functioning and do not continue to participate in the distributed system. Given such a failure model, design an algorithm for reaching agreement among a set of processors. Discuss the conditions under which agreement could be reached.

Answer: Assume that each node the following multicast functionality which we refer to as *basic multicast* or *b-multicast*. *b-multicast(v)*: node simply iterates through all of the nodes in the system and sends an unicast message to each node in the system containing v. Also assume that each node performs the following algorithm in a synchronous manner assuming that node *i* starts with the value *vi* .

• At round 1, the node performs b-multicast(*vi*).
• In each round, the node gathers the values received since the previous round, computes the newly received values, and performs a b-multicast of all the newly received values.
• After round $f + 1$, if the number of failures is less than f, then each node has received exactly the same set of values from all of the other nodes in the system. In particular, this set of values includes all values from nodes that managed to send its local value to any of the nodes that do not fail.

The above algorithm works only in a synchronous setting and the message delays are bounded. It also works only when the messages are delivered reliably.