Midterm #2 practice questions
Fall 2000 15-712

The midterm is open book, open notes, and will be held in class. We suggest that you prepare for the exam by answering these review questions, reading your classmates' paper reviews posted to the course webpage, studying the lecture notes and any of your own notes that you have taken, and skimming the assigned papers.

In the actual exam, you should write brief, precise answers. Rambling brain-dump answers may be punished twice — you run out of time writing them and we run out of patience reading them. :–)

1. It has been argued that the nature of both the Internet and UNIX around the time of the Internet worm were both helpful in allowing the worm to disseminate and helpful in defeating the worm. List two characteristics being referred to and indicate how each helped and hurt.

   (a) *Wide-scale network communication: the propagation of the Worm depended upon network communication. Likewise, much propagation of information and fixes relied upon the same network.*

   (b) *Homogeneous systems: many of the attacked systems all had the same security weaknesses, allowing the Worm's simple mechanisms to spread widely. Since the set of weaknesses was small, the set of necessary repairs was also small.*

   (c) *Well-documented holes: the weaknesses exploited by the Worm were all well-known before its release, which made it relatively easy to construct. Likewise, identifying and repairing the weaknesses exploited by the Worm was easier because those weaknesses were already known to exist.*

   (Interestingly, these same characteristics still exist today.)

2. What is priority inversion? Describe how it can be caused by locks and a negative effect that could result.

   Priority inversion is when a high priority thread is unnecessarily blocked by a lower priority thread. Locks can cause priority inversion in the following manner given 3 threads of low, medium, and high priority. The high priority thread could block on a lock held by the low priority thread.

If the medium priority thread wants the CPU, it will run instead of the low-priority thread — unlike the low-priority thread (until it releases the lock), the medium-priority thread's execution should not block the progress of the high-priority thread. One possible result of such priority inversion is failure to meet a high-priority deadline in a real-time system.

3. What makes a fault model Byzantine? Why is this more difficult to deal with than "fail-stop" faults?

   Arbitrary behavior after the fault. Byzantine faults are more difficult exactly because they are unpredictable; this makes them difficult to identify and contain.

4. How many nodes are needed to continue operating in the face of a fail-stop fault? Why? How does this answer change for a Byzantine fault?

   In the face of a fail-stop fault, only one correctly functioning node is needed. This is because the faulty node simply does not respond. Thus, the functioning node's response is sufficient. For a Byzantine fault, 4 total nodes nodes may be necessary, because, as shown by Lamport, if you only have 3 nodes, any one faulty node can prevent the others from reaching consensus.

5. Explain one way that threads can be used to avoid deadlock.

   Using Hauser et al.'s "deadlock avoider" model, a new thread could be initiated for any additional task that would otherwise lead the currently executing thread into a potential deadlock situation; this new thread could start the additional task without any locks held from previous activities.

6. Can priority inheritance be used to avoid priority inversion caused by condition variables? If so, how? If not, why not?

   Not unless additional application-specific information is available. A thread blocked on a condition variable doesn't know which thread should inherit its priority; that is, the blocked thread doesn't know which other process will eventually cause the condition to be satisfied.

7. Schneider claims that it is not possible to implement a "t" fault-tolerant system by using a single voter to combine the outputs of an ensemble of state machine replicas into one output. Instead, the voter should be replicated and whatever reads the output of the system should combine the output of the voters. This "critical voter" is not considered part of the system by Schneider. Why?

   Because if it was part of the system, it too would need to be replicated, thus no solution is possible. In practice, the critical voter could be a human looking at different monitors.