Part 1. Threads and synchronization

(a) Show how to implement mutual exclusion (locks) using semaphores.

(b) Why might different executions of the same program with the same inputs yield different schedules with different synchronization orders?

(c) Suppose two threads each write a value to memory location X. In one execution, the final value of X is the value written by thread A. In another execution, the final value of X is the value written by thread B. Do these writes constitute a race? How could you tell?

(d) One goal of the scheduler activations idea is to reduce the time that multi-threaded programs spend in system calls. How does it achieve that?

(e) What does a system with scheduler activations do when a thread blocks in the kernel? Why?
Part 2. Access control. These questions pertain to access control in Unix and the Flume system.

(a) When a Unix program is launched, its process is tagged with a userID for access control purposes. How does the kernel determine which value to give the userID?

(b) The concept of a lattice is central to information flow control (IFC). Give a few examples of Flume labels to illustrate how they meet the properties of a lattice.

(c) Why might a Flume endpoint have a label that is different from the label of the process that the endpoint is bound to?

(d) How does Flume determine if it is safe for data to flow across a communication channel between two processes?

(e) Why does Flume require that a trusted program have “dual privilege” for a secrecy tag in order to declassify data that is marked with the tag? For example, why must it have privilege to add the tag as well as remove it?
### Part 3. Haven

These questions pertain to the Haven/SGX enclave abstraction. Haven is designed to protect an app from all software outside the enclave, even the kernel. It specifies a procedure for a service owner to deploy a secure service application on a cloud provider, and establish trust in the deployment cryptographically via remote attestation. This question asks you to outline how this procedure uses basic cryptographic concepts (public keys, private keys, session keys, certificates, hashes, etc.) to authenticate the parties, implement secure channels, and defend against spying or subversion by an attacker.

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
</table>
| (a) Why do the authors of Haven believe that is important and useful to protect an app from "trusted" software, e.g., the kernel? Give two reasons. | (1) In some cases the host kernel is controlled by a third party (a cloud provider) who may have an economic incentive to spy on the app or subvert it.  
(2) The host kernel may itself be subverted by an attacker.  
Note that the Haven trust model does not address attacks from other co-hosted apps: this threat is covered by conventional OS isolation (including VMs). |
| (b) How does the service owner verify that the deployment platform is a valid SGX server? | An SGX platform has a keypair whose private key is "baked in" to the processor. The platform’s public key is endorsed by the vendor (Intel) in a certificate signed under a keypair controlled by the vendor. The certificate states: “the vendor certifies that P is a public key belonging to an authentic SGX platform”. |
| (c) How does the service owner verify that the service is deployed correctly in an enclave on the platform? | The platform computes a hash over the enclave code/data and initial parameters. It then generates a quote or attestation containing the hash and signed under the platform’s keypair. The service owner receives this attestation certificate and validates it. |
| (d) How does the service owner communicate securely with the deployed service in its enclave, without risk that the platform’s kernel can inspect or modify the traffic? | The service (shield) generates a keypair after launch. The public key is included in the quote signed by the platform. Therefore the service owner knows the service’s public key. It can generate a session key and pass it to the service, encrypted with the service’s public key. Subsequent communication is encrypted under the session key. (This is a whole lot like SSL.) |
| (e) How does a client of the deployed service communicate securely with the deployed service in its enclave, without risk that the platform’s kernel can inspect or modify the traffic? How does it authenticate the service? | In Haven, the service owner passes the service (shield) a symmetric key to decrypt and access a Virtual Hard Drive (VHD). The VHD contains an SSL certificate for the service domain, and a corresponding private key. When a client contacts the service, the service presents its SSL certificate. An SSL connection is established in the usual way: the client generates a session key, and encrypts it under the public key in the SSL cert. |
Part 4. Consistency

These questions pertain to consistency for networked data stores with replication, based on the papers on DDS, GFS, and “Replicated Data Consistency Explained Through Baseball”.

(a) The “Baseball” paper presumes that all writes execute in the same order at all replicas. How would a replicated store ensure this property?

(b) The “Baseball” paper discusses various models in which a replica may return stale data on a read. Given that all replicas execute the same writes in the same order, why would a replica ever return stale data on a read?

(c) GFS is optimized for appends, but GFS reads may return inconsistent data when append-mode writes fail or cross chunk boundaries. How do GFS applications protect against using this inconsistent data returned by reads?

(d) Leases are useful to preserve consistency when network connectivity is lost. What should a node do if it cannot renew a lease?

(e) GFS has a centralized master server. This centralized master is not a performance bottleneck if the GFS design assumptions are valid. Briefly summarize why.
Part 5. Loose ends.

(a) In Remus, how does increasing the checkpoint interval affect service latency and checkpoint cost? Please draw little graphs to illustrate.

(b) How does Remus determine what data to transmit in a checkpoint?

(c) How does the GFS master ensure that updates to its data made by incoming RPCs are not lost if the master fails?

(d) What was the hardest or most time-consuming part of your solution to the “Hacker lab” (cracking into the buggyserver Web server)?

(e) What was the most interesting or difficult concept that you learned in this class, and which is not represented elsewhere on this exam? Explain it as well as you can in the box.

Have a great summer!