**Question 1. System Virtualization**

A hot new topic in operating systems is system virtualization. While IBM has been offering its VM system for mainframes for many years, virtualization technologies are now migrating to PC architectures, including PowerPC- and i86-based chips. A second trend is the development of multi-core machines, where a single chip essentially implements a small-scale multiprocessor machine. The result will be that multiple virtual machines (and their guest operating systems) can concurrently run on a single chip.

This question asks you to lay out the principles behind building a scalable hypervisor for a future multi-core chip (assume shared memory multiprocessor for now).

- You must deal with virtualized CPU, memory, disk, and communication resources.
- For each such resource and the hypervisor functionality addressing it, you must consider scalability issues.

Please answer the following specific questions:

a) How can you scale to large multi-core machines, with 16 or 32 processors? That is, for each resource, what are the scalability issues? Define scalability.

b) What design principles will you use to attain scalability, again, for each resource? Please feel free to refer to papers you have read for this part of the question.

c) How can you deal with dynamic VM arrivals and departures, that is, what additional hypervisor and/or guest OS support do you envision to prevent non-scalable overload behavior?

**Questions 2. Overlay Networks**

A common, new technique in distributed systems is to dynamically create and maintain a set of machines on which a certain application runs. The resulting 'overlay network' can be managed and established independently of the applications that use it. A simple application presented in early work on this topic used an overlay network to bypass a low-bandwidth communication link provided by the network for remote login session. In peer-to-peer systems, elaborate 'election' algorithms have been developed to identify suitable vs. unsuitable nodes for remote program execution.

The purpose of an overlay network is to provide an execution environment for a distributed application. This question asks you to consider overlay networks from the QoS perspective. That...
is, the reason for creating some overlay is to provide to an application that uses it some desirable level of QoS. Further, the specific QoS of interest to this question is high availability.

a) Define availability.
b) Describe what information would be needed at the overlay level and shared with the application level about this QoS. Pick a sample application to illustrate your answer to this question.
c) Describe a simple overlay-level algorithm you could use to attain high availability across underlying machines and networks with dynamically varying quality.
d) Discuss the interfaces needed between application and overlay to utilize your newly ‘adaptive’ overlay. Are there any interfaces necessary at all, or could overlay adaptation remain totally transparent to an application? If yes, then, are there any benefits to creating additional ‘awareness’ interfaces? Please be specific here, discussing some specific example and discussing the advantages and disadvantages of application-level awareness.

**Question 3: Distributed Simulation**
(a) Contrast the approaches typically used in distributed virtual environments (e.g., a multi-player gaming application) vs. parallel/distributed analytic simulations (e.g., execution of a simulation modeling a telecommunication network) with respect to event ordering. Explain why different approaches are typically used for these two domains.

(b) Propose an algorithm for performing conservative synchronization of federated analytic simulations using a client/server architecture, here each client is a discrete-event simulator, and the server performs computations necessary to properly synchronize the client simulations. Show pseudo-code for your algorithm. Assume the underlying communication network guarantees ordered delivery of messages, i.e., successive messages sent from a source process to the same destination are delivered in the same order in which they were sent. Argue why your algorithm is correct.

**Question 4: Time Warp**
A well-known problem with optimistic synchronization algorithms such as Time Warp is the execution may spend too much time performing computations that are later rolled back. Propose two different algorithms that attempt to address this problem, and argue the merits and limitations of these approaches. In particular, describe situations where you would expect your first algorithm to outperform the second, and conversely, situations where the second would be expected to outperform the first.

**Question 5: Real-Time Systems**
A dynamic priority scheduler such as earliest deadline first (EDF) and minimum laxity first (MLF) is considered optimal since it is able to maintain feasible schedules up to 100% of CPU utilization. In contrast, a fixed priority dynamic scheduler such as rate monotonic has worst case achievable utilization limits (0.693 in a theoretical analysis and about 0.9 in an average case analysis) that are less than 100%. Construct a schedule that can be scheduled by EDF or MLF (with all jobs completing before deadline), but cannot be scheduled by rate monotonic (with some jobs missing their deadlines).

**Question 6: Message Passing In Parallel Machines**
The IBMs, Dells and Crays of this world are building high-performance clusters with low-latency hardware interconnects with wire speeds of the order of a few microseconds. However, the latency that an application sees for message passing on top of a standard operating system such as Linux is nowhere near these hardware speeds.

a) Why? Enumerate the sources of overheads that hamper the performance for message
passing at the application level. Be as quantitative as possible in presenting your solution. Identify the sources of overheads, and the extent to which they hurt performance.

b) Suggest solution approaches (your own as well what you have read from the papers in the reading list) to overcome some of these overheads. Once again, be quantitative in the suggestions you come up with for overcoming the sources of performance overheads, i.e., the extent to which these solutions help in reducing the overheads.

Question 7:
Consider a fault-tolerant distributed system when node behavior is constantly monitored and the failure of a node is detected within a certain amount of time. All other nodes are immediately informed when the failure of a node is detected. Many fault-tolerant protocols rely on a threshold assumption that states that no more than \( k \) out of \( n \) nodes could be faulty. The overhead of the protocol often depends on \( k \). If the system can implement monitoring to diagnose and detect failures in a timely manner, it may be possible to design more efficient protocols.

Give an example of a fault-tolerant implementation of a service (e.g., a highly available file service) and the protocols deployed by it where the threshold assumption is used. Can the overhead associated with these protocols be reduced if failures can be detected as described above? Explain your answers.

Question 8:
Capabilities represent one mechanism that could be supported by operating systems for protecting resources. Are there examples of capabilities or capability-like mechanisms that are used in widely deployed operating systems? Are there recent research operating systems that use capabilities for protecting resources? How can we explain the relative lack of widespread use of capabilities in operating systems?

Consider an open distributed service where a set of clients can access the common service using a remote procedure call like mechanism. Discuss an implementation of a capability based scheme for controlling access to this distributed service. Provide both the API and the implementation of the calls that you propose to protect the service.

Question 9: Distributed Systems - Remote Procedure Calls
This question explores the ins and outs of remote procedure calls, and how they are used to build distributed services.

a) One key component of any RPC system is stub generation. Describe the process of stub generation, and how it eases the process of distributed application development. What types of problems are avoided through the automation of this process?

b) Naming is an important ingredient in any RPC system. First, describe where the issue of naming arises (i.e., discuss when clients and servers would utilize a name service). Then, describe how a particular RPC system utilizes a naming service (the RPC system as described by Birrel and Nelson, Sun RPC, or Java RMI are all suitable). For this last part of the question, a rough outline of the important steps and interactions is all that is needed.

c) One key question that arises in building a distributed service is how to partition work across the client and the server. First, describe why is partitioning important. Then, present and discuss a general set of guidelines which one could use to partition a service across a client and server.

d) Finally, describe general guidelines that one could use to partition a service over a P2P overlay network.