Question 1 (Embedded Systems)
Embedded systems are similar to real-time systems in that they are both constrained by cost and resource limitations, but need to achieve quality of service in their functions. The most important priority of real-time systems is to complete their tasks before the deadline. The most important priority of embedded systems is to complete their tasks while minimizing resource consumption. The next generation of embedded real-time systems will need to complete their tasks before the deadline, but still minimize resource consumption. For concreteness, consider the scenario of dual-use thermometers that report temperatures in a building. During normal operations, the thermometers monitor the room temperatures and help control the central air conditioning system. In a building fire, the thermometers provide an accurate reading of room temperatures to indicate the propagation of fire through the building.

(1) [10%] Discuss the management trade-offs of battery-operated thermometers vs. wired thermostats as done typically today (e.g., in terms of routine maintenance and other relevant factors).

(2) [10%] Discuss the management trade-offs of connecting the thermometers using a wireless network vs. a wired network (e.g., using the electrical wires in a wired thermostat).

(3) [20%] Discuss the technical trade-offs of a short-distance network (e.g., Bluetooth) vs. a medium-distance network (e.g., WiFi), e.g., in terms of power consumption and protocol complexity (and other relevant factors).

(4) [20%] Outline a sketch of the design for such a system, based on the discussion in parts (1), (2), and (3). For concreteness, consider a large building complex with offices, shops, and a hotel (e.g., Atlantic Station). Focus on the sensors and communications, simplifying the control part.

(5) [20%] Discuss the design trade-offs for normal costs (e.g., installation vs. maintenance), and the trade-offs between normal costs vs. performance and reliability during emergencies (e.g., a large scale fire).

(6) [20%] Discuss the potential impact of future technologies on this application, in terms of components (e.g., sensors), communications (e.g., networks), and system integration (e.g., cyber-physical systems).

Question 2 (Dynamic System Reconfiguration)

With typical operating system code base on the order of millions of lines of code, customization is a necessity for most systems. This is particularly the case of device drivers, since a particular system only needs the drivers required for its concrete physical configuration. To achieve some flexibility, standard interfaces such as USB support dynamic loading of device drivers when they are connected to the system for the first time. This is done routinely, for example, when a USB-based memory stick is inserted for the first time.

(1) [25%] Explain in some detail (e.g., using pseudo code) the steps taken by the kernel, from the moment a new memory stick is inserted into a USB slot until it is ready for access. [No more than one page.]
(2) [15%] Compare the performance overhead of the dynamically loaded driver with a statically linked driver, both at load time and at execution time.
(3) [30%] Are there ways to eliminate the overhead introduced by dynamic loading? Using your answer to part (1) as a concrete scenario, explain a solution. Alternatively, explain the reasons the overhead cannot be eliminated.
(4) [30%] In part (1) of this question, you made an assumption about the location of the memory stick driver code (e.g., on disk as part of OS distribution or over the network). Explain the reasons why hackers may be able to make use of the dynamically loaded driver mechanism to penetrate your kernel. Provide security mechanisms to prevent these potential penetrations.

Question 3 (Scalability of applications and OS structure)

This question concerns scalability of parallel applications on a multiprocessor.

a) What are the sources of overhead that limit the scalability of an explicitly parallel application on a multiprocessor?
b) Digging deeper and assuming a parallel machine with cache-coherent shared memory, identify what aspects of these overheads could be attributed to the operating system structure and the abstractions provided by the operating system.
c) Focusing on the multi- many-core era that we are in, explore operating system abstractions that could enable achieving the full potential of the inherent parallelism in an application.

Question 4 (Memory management for supporting interactive applications)

Modern workloads are interactive and often memory intensive (e.g., web browsers with multimedia plugins, and multimedia players). The memory management policies of modern operating systems (Linux, Windows, etc.) are not very good at catering to such interactive applications since these policies have not changed drastically for several decades. This question has two parts:

(a) First, using Linux as an example, explain why its memory management policies may not be conducive to enhancing the performance of interactive applications
(b) Explore novel memory management policies that would help improve the performance of interactive applications.

Question 5 (Resource Management)

Every resource management system needs protection against overuse or abuse of the resource. Three general strategies are "prevention", "avoidance", and "detection & recovery".

(1) Describe these strategies, carefully distinguishing between them, and give an example of each.
(2) Give arguments for and against applying the "detection & recovery" strategy.

Question 6 (Reliability in Distributed Systems)

One solution to the Byzantine Generals problem makes the following assumptions about the underlying system:
(1) Every message that is sent is delivered correctly.
(2) The receiver of a message can reliably determine the identity of the process from which it was (most recently) sent.
(3) The receiver of a signed message can reliably determine whether the message has been modified since it was signed.
(4) Anyone can verify the authenticity of a signature.

Discuss whether each of these assumptions is reasonable in light of the way distributed systems really work. If an assumption is reasonable, describe how a system would support such a feature. Where applicable, provide examples from systems that you know. If an assumption is unreasonable, explain why the feature is difficult to provide.

**Question 7 (Distributed Systems)**

Consider a distributed game in which you are trying to provide functionality that permits players to jointly decide on other players to be removed from the game (e.g., because they are cheating). The particular cheat in question is one in which a player consistently violates certain causal dependencies (e.g., shoot before reloading).

(1) Explain how a game might be instrumented (e.g., additional messages produced and consumed/checked) to catch such cheats.
(2) Explain how a collective decision about whether or not someone has cheated can be made, without resolving to an approach in which a central server is relied upon and assuming an asynchronous system.
(3) Comment on the failure model you can tolerate (you do not need to support byzantine failures).

**Question 8 (Middleware/Distributed Systems)**

Middleware is playing an important role in enterprise systems, starting with CORBA and Java, now moving toward SOA. An important technical issues with middleware is the relative difficulty to make guarantees about or set limits on resource usage, including processor and memory usage. In this question, you address these problems not by changing the middleware, but by embedding it into its own virtual machine, then applying resource limitations and making guarantees for the VM rather than the middleware. Given a standard multi-tier web application like RUBiS, answer the following questions:

(1) What guarantees can you make concerning RUBiS behavior (e.g., request throughput) by simply controlling the resources allocated to VMs running RUBiS (assume one VM per RUBiS process)?
(2) Can you detect and then prevent certain known misbehaviors of Java applications, such as highly variable timing of request execution due to garbage collection? Answer this question technically by talking about how detection/prevention may be done at hypervisor level, if possible.
(3) Going beyond RUBiS, can you deal with arbitrary SOA applications, where a service might decide at runtime which other service to invoke (assuming that you are in control of all such services running on VMs in a single datacenter)? ‘Deal with’ in this context concerns service guarantees like request delay.
(4) Do your solutions scale (and what do they cost?) to larger applications (e.g., consider RUBiS at eBay scale)? Discuss technically by defining scalability.
Question 9 (Low Level OS Kernels)

There is a long tradition of creation OS kernels specialized to certain tasks, such as real-time kernels, kernels suited for certain applications (e.g., specialized to run Java VMs), kernels for embedded systems (e.g., consider Nokia’s Symbian OS), etc. In this question, you are outlining the design for a ‘device driver OS’, which is an operating system focused on high performance data movement as what is needed to run device drivers.

(1) Outline your OS’s functionality - threads/processes? scheduling? memory management? etc.
(2) Discuss the utility of your OS vs. ‘just using Linux’. Be explicit here: where do you expect performance gains and why vs. what issues exist?
(3) Can you add support that assists in dealing with major issues concerning device drivers, such as correctness/misbehavior, etc.? Describe some ideas.
(4) How can you make this OS interoperable with other systems?