Question 1 (OS issues in system structuring)
Following Liedtke arguments for constructing a microkernel, explain how context switching overhead of a microkernel may be mitigated in modern architectures. Specifically, discuss the difference in approaches to solving this problem for the PowerPC and Intel Pentium architectures. Both architectures support segmentation in hardware. PowerPC segmentation scheme allows mapping a $2^{32}$-byte local address space to a $2^{52}$-byte global address space. Pentium segmentation allows mapping a $2^{32}$-byte local address space to a $2^{32}$-byte global address space. Clearly explain the rationale and the expected performance impact of the approaches you propose.

Question 2 (OS issues in parallel systems)
1. As an OS designer for an SMP, present key objectives you have to pay attention to.
2. Mention properties of a clustered object that help meet the above objectives.
3. We want to design a processor scheduler for an SMP. Demonstrate a concrete use case of clustered objects for designing the scheduler.

Question 3 (High Performance Computing)
1) Collective communications are key to the efficient execution of high performance codes on large scale parallel machines. With the impending explosion of cores in future manycore machines, they may experience a renaissance in their use and importance for future applications and their thread-thread communications.
   - Explain at least two different examples of collectives (other than the obvious one, which is broadcast).
   - Using a specific example from the prior point, explain how you would implement a collective on a manycore machine, assuming that it is used for thread-thread communication. Explicitly comment on the following different aspects of its implementation:
     1) data transfer (sizes, from where to where (buffers?, ...),
     2) control, including signalling (e.g., polling vs. interrupts), and
     3) usage, referring to how you expect a future application to use your collective.
   - Since you made your collective extensible, comment on interesting extensions that would make sense for applications. Be sure to be explicit by mentioning representative applications here.

Question 4 (Distributed Computing)
Describe intuitively, not formally, what Fischer, Lynch, and Patterson are arguing in their paper on 'The Impossibility of Distributed Consensus with one Faulty Process". Note that it's important here to be precise about the computational and failure models to which the
authors are referring (make sure you describe them). Then come up with a change to the computational model that makes it possible to arrive at a consensus in finite time. Describe the change and argue why it works.

**Question 5 (Peer to Peer Systems)**
Initially, peer to peer systems were key to exchanging content across end users, bypassing server systems. This led to the creation of peer to peer file systems, mail systems, etc. An interesting, more recent development is that with ever increasing computational power of end systems, peer to peer systems can be constructed that require rich processing on such end systems, potentially leading to the creation of 'serverless' large scale applications, such as massively multiplayer online games. The point of this question is to have you

- identify the key elements of any peer to peer MMOG (e.g., state space representation, QoS needs, comm. protocols, ...)
- define a specific problem for such a MMOG, concerning how to divide the game space across peers, and suggest a solution,
- then address and solve a solution for the load balancing problem that rises (define the problem, explain how/when it arises given your ecomposition of the state space, provide a possible solution), and finally
- assuming unreliable communications and peers running across heterogeneous networks (from home, in the office, ...), come up with a way to (1) assess fairness in terms of a player's ability to perform better than another, and (2) enforce fairness.

**Question 6 (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 battery-operated thermometers that report temperatures in a building over a wireless network. 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 as fire propagates through the building.

1. Discuss the trade-offs of a short-distance network (e.g., Bluetooth) vs. a medium-distance network (e.g., WiFi) in terms of power consumption and protocol complexity.

2. Discuss the management challenges of monitoring the proper functionality of thermometers (e.g., discovery of malfunctions) and physical management (e.g., battery replacement).

**Question 7 (Real-Time Scheduling)**
Dynamic priority scheduler such as earliest deadline first (EDF) and minimum laxity first (MLF) can reach worst-case achievable utilization (WCAU) of 100%. Fixed priority dynamic schedulers such as rate monotonic have relatively lower WCAU. Compare the dynamic priority schedulers with fixed priority dynamic schedulers in terms of their execution overhead, the WCAU, and ease of implementation. Give a concrete example of real-time task set (with CPU requirement and period) that can be scheduled by EDF, but not guaranteed by rate monotonic.

**Question 8 (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. 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. Compare the performance overhead of the dynamically loaded driver with a statically linked driver, both at load time and at execution time.
3. Are there ways to eliminate the overhead introduced by dynamic loading? Explain a solution, or explain the reasons the overhead cannot be eliminated.
4. 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.

**Question 9 (Replication)**

Redundancy is a foundational concept for fault-tolerance and reliability. Without redundancy, a failure can take out a critical component and consequently an entire system that depended on that component.

1. Compare the differences and similarities in the literature between replication for availability vs. caching for performance. Explain at least two dimensions in this comparison, e.g., what they do when updates happen and variants in their implementation.
2. Compare the differences and similarities, as well as advantages and disadvantages, between transactional update of symmetric replicas vs. primary-secondary asynchronous update propagation.
3. Compare replica update algorithms that use transactions with algorithms that use group communications mechanisms such as atomic broadcast and causal broadcast. Explain at least two dimensions, e.g., commit protocols and performance impact.