1. OS Basics
   1. What is an Operating System?
   2. When does it run?
   3. What is the size of the memory address space in a machine with 64-bit architecture, i.e. where each memory address is 64 bits long?
   4. How much is 211 (2 to the power 11) in decimals?
2. Briefly explain the following interfaces in a computer system

* Instruction Set Architecture (ISA)
* User Instruction Set Architecture (User ISA),
* System ISA,
* Application Binary Interface (ABI).

1. Processes and Threads
   1. What is a process?
   2. What is a thread?
   3. When would you (as a programmer) prefer to use multiple threads instead of multiple processes?
   4. When would you prefer to use multiple processes instead of multiple threads (in one process)?
2. Which state transitions (if any) occur in a process lifecycle when a process
   1. Runs too long on the CPU?
   2. Tries to read keyboard input, but no input is available?
   3. Receives a signal?
   4. Attempts to execute a System ISA instruction in user space?
   5. Attempts to perform down on a semaphore that is zero?
3. Consider the two models of threads: user-level and kernel-level (not kernel threads). Answer the following and explain why.
   1. Which model can support Local thread scheduling?
   2. Which model can support Global thread scheduling?
   3. Which model would you prefer for an application that performs mostly concurrent numerical computations, but little I/O?
   4. Which model would you prefer for an application that performs a lot of concurrent I/O?
   5. Which model cannot be used on multi-processor (i.e. SMP, or multi-CPU) machines?
4. Event-driven programming
   1. What is the “event-driven” programming model?
   2. What major problems in event-driven programming are solved by thread-based programming?
   3. When would you prefer event-driven programming over thread-based programming?
5. System calls and Kernel Modules
   1. What is special about system calls compared to normal function calls?
   2. What is a system call table? Why is it needed?
   3. Explain how the CPU privilege changes during a system call.
   4. How do system calls differ from kernel modules?
6. What are the following? How can you prevent them?
   1. Race conditions
   2. Deadlocks
   3. Priority Inversion
7. Synchronization
   1. Explain UP and DOWN operations on a semaphore
   2. Explain WAIT and SIGNAL operations on a condition variable.
   3. What is the producer-consumer problem (NOT the solution) and its three synchronization requirements?
8. [This is an open-ended question to test design/critical thinking skills. There can be many correct answers, but not all answers are correct.]

Traditional operating systems support the fundamental abstractions of Processes/Threads which are stateful, blocking, and (typically) long-running. Most OS services, such as the CPU scheduler, have evolved to manage process transitions across the process lifecycle. Even event-driven programming is currently implemented using a process that monitors different events in a while loop.

Consider an alternative OS design in which there are no processes/threads. Instead the fundamental OS abstraction is “event-driven” Tasks which are stateless, non-blocking, and (typically) short-lived. Describe how this abstraction could be supported by the OS and what would be the key changes in the OS, particularly for CPU scheduler, I/O processing, and concurrency primitives.

﻿