1. [Silberschatz] We have stressed the need for an operating system to make efficient use of the computer hardware. When is it appropriate for the operating system to forsake this principle and waste resources? Why is such a system not truly wasteful?

Answer: This situation arises in many places. It all comes down to what you mean by "waste resources". For example, Windows "wastes" disk space by including many device drivers in their distribution. An individual user won't use all these drivers, but the hassle of looking for a driver is much more wasteful than the relatively cheap disk space. Another example is in the block size a system uses for disk transfer - smaller blocks waste less disk due to fragmentation, but larger blocks result in better transfer rates. A third example is in scheduling: some real-time schedulers can only guarantee meeting deadlines if the processor is not utilized more than roughly two-thirds of the available cycles. Meeting deadlines with an efficient scheduler is often more valuable than squeezing all the cycles out of a processor.

2. [Silberschatz 1.13] When are caches useful? What problems do they solve? What problems do they cause? If a cache can be made as large as the device it is caching for (for example, a cache as large as a disk), why not do so and eliminate the device?

Answer: The usefulness of cache relies on the fundamental principle of program locality, that is, the non-uniform code and data accesses associated with most programs. A widely quoted rule of thumb is that a program spends 90% of its execution time in only 10% of the code. This rule, together with the guideline that smaller hardware is faster, led to the use of caches in storing code and data that probably will be used in the near future.

Caches are used to hide the increasing performance gap between processors and memories. Ideally programmers would want unlimited amounts of fast memory. However, since fast memory is quite expensive, a hierarchy of various storage resources is employed to provide a memory system with cost almost as low as the cheapest level and speed almost as fast as the fastest level.

The levels of the memory hierarchy usually subset one another in that all data in one level are also found in the level below. This organization creates a coherency problem if all copies of the same data are not updated at the same time. This problem becomes more serious in a multitasking environment, where the CPU is switched back and forth among various processes. Another problem caused by cache is that if the data hits in the cache, memory access time can be reduced. However, if the data is not presented in the cache, the main memory needs to be looked up subsequently, thus increasing the worst-case memory access time.

Two fundamental reasons account for not making a cache as large as a disk. Firstly, it is not cost-effective to make a huge cache because caches are more expensive per byte than disks (at least 1000 times expensive). Secondly, the fast access speed of a cache partially attributes to its small size, implying that making a huge cache would result in the loss of its speed advantage.
3. [Silberschatz 1.11] Direct memory access (DMA) is used for high-speed I/O devices in order to avoid increasing the CPU's execution load.
   a. How does the CPU interface with the device to coordinate the transfer?
   b. How does the CPU know when the memory operations are complete?
   c. The CPU is allowed to execute other programs while the DMA controller is transferring data. Does this process interfere with the execution of user programs? If so, describe what forms of interference are caused.

   Answer: a. To initiate a DMA transfer, the CPU first sets up the DMA registers, which contain a pointer to the source of a transfer, a pointer to the destination of the transfer, and a counter of the number of bytes to be transferred. Then the DMA controller proceeds to place addresses on the bus to perform transfers, while the CPU is available to accomplish other work.
   b. Once the entire transfer is finished, the DMA controller interrupts the CPU.
   c. Both the CPU and the DMA controller are bus masters. A problem would be created if both the CPU and the DMA controller want to access the memory at the same time. Accordingly, the CPU should be momentarily prevented from accessing main memory when the DMA controller seizes the memory bus. However, if the CPU is still allowed to access data in its primary and secondary caches, a coherency issue may be created if both the CPU and the DMA controller update the same memory locations.

4. [Tanenbaum] Give one reason why a closed-source proprietary operating system like Windows should have better quality than an open-source operating system like Linux. Now give one reason why an open-source operating system like Linux should have better quality than a closed-source proprietary system like Windows.

   Answer: As a closed-source proprietary system like Windows is not free, it needs to have better quality. Otherwise no one is going to buy it. On the other hand, an open-source operating system like Linux enables users to customize and refine it, thus enabling it to have better quality.

5. [Silberschatz 2.14] What is the main advantage for an operating system designer of using a virtual machine architecture? What is the main advantage for a user?

   Answer: The fundamental goal of a virtual machine architecture is to share the same hardware yet run several different execution environments. Accordingly, from the perspective of an operating system designer, a virtual machine architecture enables a complete protection of the various system resources as each virtual machine is completely isolated from all other virtual machines. From the perspective of a user, a virtual machine architecture enables the concurrent execution of multiple execution environments on a single machine.

6. A question on signals in Unix, which is a software abstraction of interrupts and traps. You may need to look at the man pages on signals to answer this question.
   a. Write a C program that goes into an infinite loop. When the user enters ^C three times, then the program terminates. The first two times that the user enters ^C, however, does not terminate the program. You will do this by writing a signal handler that catches the SIGINT signal, which is generated when a user types ^C. Your program should not be longer than 25 lines of C code. Turn in the listing of your code.
   b. The signal SIGKILL can't be caught with a signal handler. Why is it useful to have such a signal that cannot be caught?
Answer: a.

```c
#include <signal.h>
#include <stdio.h>

int count = 0;

void catchC (int sig) {
    if (++count == 1) printf("Strike one... \n");
    else if (count == 2) printf("Strike two... \n");
    else { printf("Out!\n"); exit(0); }
}

int main (int argc, char** argv) {
    signal(SIGINT, catchC);
    while (1) { }
}
```

b. Suppose that a process could catch all signals (and ignore them). How would one be able to force such a rogue process to terminate? SIGKILL can always be used to kill a process, no matter how carefully it catches and ignores signals.