1. Rootkits (malicious code in the kernel) can intercept system calls made by processes (all processes or a specific process) and replace the original system call behavior with some other malicious behavior. How would you go about implementing system call interception for such a rootkit? Describe the design but don’t write any code.

Answer: The basic idea is to write a kernel module (the rootkit) that obtains the base address of the system call table. The system call table contains pointers to individual functions that implement each system call. The kernel module then replaces these function pointers in the system call table with pointers to its own malicious functions. These malicious functions can then intercept each system call made by user-level processes, examine the process ID, system call arguments etc. If the rootkit wants, it can replace the original syscall behavior with its own malicious behavior, else it can allow the original system call to proceed as intended.

2. Explain the two key steps in delivering data in packets from a network card to a user-level process reading from a network socket?

Answer:
Step 1: Copying data from the network card to kernel buffers.
Step 2: Copying data from kernel buffers to process in user space code.

3. Explain the difference between (a) signal-driven I/O and asynchronous I/O, (b) blocking I/O and I/O multiplexing.

Answer:
(a) Signal-driven I/O vs. Asynch I/O: For signal-driven I/O, the second stage (copying data to user space) blocks the process till data is copied. For asynchronous I/O, the second stage does not block the process.
(b) Blocking I/O vs. I/O Multiplexing: The blocking I/O model blocks a process indefinitely on only one I/O descriptor at a time. With I/O Multiplexing, a process can block (indefinitely or for a specified time) on multiple I/O descriptors at the same time.

4. (a) What is meant by “Internal” and “External” fragmentation of memory? (b) Which fragmentation is made worse by superpages? Why?

Answer:
(a) Internal fragmentation occurs when part of an allocated memory region is not used by a
process and is thus wasted. External fragmentation occurs when all free memory regions are small and distributed at different non-contiguous locations in the main memory. As a result, an allocation request for a large contiguous memory region cannot be satisfied, even though enough free memory exists.

(b) Both internal and external fragmentation are made worse. Internal fragmentation is worse because pages are large, so there is a higher chance that an allocated memory page may not be used. External fragmentation becomes worse because the operating system may allocate superpages of different sizes. Thus it may become difficult over time to find enough base pages that are contiguous (next to each other) to satisfy a superpage allocation request.

5. (a) How many page tables are there per segment in Multics and Pentium. (b) Which one (Multics/Pentium) is better? Why?

Answer:
Multics has one page table per segment. Pentium has one page table for all segments in a process. Pentium’s design is better for segmentation because switching between uses of different segments in a process doesn’t require the MMU to switch states between two different page tables. This eliminates TLB flushes and cold-start penalty.

6. Suppose that the operating system wanted to track (or intercept) every write performed to a specific memory page by a process. Explain how the OS would achieve this goal?

Answer: The OS would mark the desired pages read-only. When the process attempted to write to the page, an exception would be generated by the hardware giving control to the OS. The OS can then do whatever it wants with the write attempt such as record it and allow the write or deny the write, etc.

7. In the superpage paper, when there is memory pressure, how does the operating system avoid the need to page-out an entire superpage to the disk? What is the overhead in this mechanism, if any?

Answer: When memory pressure increases, OS marks all superpages as read-only. When processes try to write, the OS handles the write exception. At this point, the OS demotes the superpage to base pages around the location of the write operation and to smaller superpages elsewhere. For instance, a write to the 5th base page in a 8-page superpage would result in demoting the 8-page superpage to super gapes of sizes 4, 1, 1, and 2. The overhead of this mechanism is that overwrite operation to a superpage becomes expensive (when there is memory pressure) because all writes are trapped by the OS for superpage demotion.
8. Consider RAID levels 1, 3, 4, and 5 (forget about RAID 0 and 2). Which RAID level provides the best (a) reliability (b) I/O Parallelism. Explain why.

Answer:
(a) RAID 1 provides the best reliability, although at the expense of substantial overhead in extra disk space. It guarantees recovery from a single disk failure. It can also recover from most two-disk failures, except the ones in which both a primary disk and its mirror disk fail.
(b) RAID 5 provides the best I/O parallelism. In the best case, it allows N+1 simultaneous read operations and (N+1)/2 simultaneous write operations.

9. [10 points] Describe the design of a parity-based RAID system that can survive two-disk failures (as opposed to single-disk failure discussed in class). In your design, be sure to explain the following: (a) How your system would compute the parity required for recovery from a two-disk failure? (b) How your system would recover from two-disk and single-disk failures, (c) How much additional space would parity information occupy, compared to data, and (d) What is the maximum level of read and write I/O parallelism in your design?

Answer:
A straightforward solution is to extend RAID 1 with two additional parity disks. (This is equivalent to extending RAID 4 by mirroring every disk, including the parity disk).

Alternatively, extend RAID 5 by mirroring every disk.

(a) Compute XOR-based parity over the primary data disks. And make a mirror of the parity disk.

Alternatively, the parity blocks can be spread out over the primary and mirror disks as in RAID 5.

(b) Two-disk failures: If two unrelated disks (not primary and its mirror) fails, then simply copy over the mirror. If both the primary and its mirror fail, then reconstruct the failed primary by XORing the other primary disks. Then copy over the reconstructed primary disk to its failed mirror disk. Single-disk failure: Simply copy over the corresponding mirror disk.

(c) Extra space : N/2 data disks would require N/2 mirror disks + 2 parity disks.

(d) With first solution (extending RAID 1 or RAID 4), maximum read parallelism = N and maximum write parallelism = 1.

With second solution (extending RAID 5), max read parallelism = N+2 and max write parallelism = (N+2)/2.