13.8 When multiple interrupts from different devices appear at about the same time, a priority scheme could be used to determine the order in which the interrupts would be serviced. Discuss what issues need to be considered in assigning priorities to different interrupts.

Need to have a scheme which guarantees all interrupts will be serviced at some point. Similar to CPU scheduling, do not want an interrupt to be completely ignored because higher priority interrupts continue to enter the queue. May also have some interrupts which depend on others, in which case interrupt A for example may need to be handled before interrupt B. This is a similar issue to CPU process scheduling. Device interrupts should have higher priority than traps generated by user programs. Certain tasks can always be delayed, interrupts handling device control should always be handled before these. Devices with real-time constraints on data should be given higher priority. Devices with no buffering should also have higher priority, as the data will only be available for a short period of time.

13.12 What are the various kinds of performance overhead associated with servicing an interrupt?

Need to context switch the current running process. Need to switch to kernel mode to handle the interrupt. On completion of the interrupt will need to context switch back into the process.

13.13 Describe three circumstances under which blocking I/O should be used. Describe three circumstances under which non-blocking I/O should be used. Why not just implement non-blocking I/O and have processes busy-wait until their devices are ready?

Blocking I/O is appropriate when the process will be waiting for only one specific event. These could be disk/tape access or keyboard input for a specific application.
Non-blocking is useful for when I/O can come from multiple different locations at a given time and the order of these interrupts is not predetermined. Examples may be network daemons, listening to multiple network sockets, window managers which capture both keyboard and mouse inputs and could receive either interrupts and I/O management programs which need to access a number of different I/O devices.

Non-blocking is much more complicated to implement for programmers. Further busy waiting is less efficient than interrupt-driven I/O so overall system performance would decrease.

14.12 The access-control matrix can be used to determine whether a process can switch from, say, domain A to domain B and enjoy the access privileges of domain B. Is this approach equivalent to including access privileges of domain B in those of domain A?

Yes, these are equivalent. So long as the switch privileges associated with domain B are also copied over to domain A.

14.24 How can systems that implement the principle of least privilege still have protection failures that lead to security violations?

The most obvious is down to user error. An administrator could create an "all-powerful" group with full access to the entire system, and include users in this group. This could be malicious on the administrators part, but more likely issues will stem from laziness.

Further, if a program is given access to a particular system, although the rest of the system is protected, if that program is compromised in some way an attacker could still bring harm to the particular system which the program was given access to. Thus protection has failed in that system even though the rest of the system is safe.