1. D is correct; buggy software interlocks were the main problem. A is wrong; although it was initially claimed that the Therac-25 had no software in common with the Therac-20, the authors report that some was later discovered. The paper contradicts statements B and E and it provides no evidence for C.

2. D is correct. A is wrong, because when pages are in the application-level cache, both AMPED and SPED can read them at the same rate. AMPED is a multi-process architecture, while SPED uses a single-process, so SPED is the one that has fewer thread switches, which contradicts C. And since both are event-driven and there is an asynchronous network interface, both AMPED and SPED can process a request from one client at the same time it is sending a page to a different client, which excludes E.

The intention was that answer B be true, because when several AMPED helper processes issue disk reads at about the same time, the disk driver can discover them and schedule those reads in batches to improve performance. Since SPED must wait for one disk read to complete before it can issue another one, it misses this opportunity. But the definition of batching given in the glossary of chapter 3 accidentally got out of sync with this question; that definition requires delaying an operation until a time when it can be combined with another operation. Any disk driver that takes that strategy would probably speed up AMPED and SPED equally well, which would make B false.

3. A describes the main message of the paper. The paper describes all of the other answers as counterproductive.

4. B. The largest useful value for the size of the segment table is equal to the largest segment number that the hardware can handle.

5. A. The index identifies a byte within a segment, counting from the beginning of the segment, so a segment cannot usefully be larger than the largest index that the hardware allows.

6. E. The number of bits in a physical address determines the maximum number of bytes of physical memory that can be attached, but that is unrelated to the size of virtual addresses. The design specifies only the size of the virtual addresses, so there is no requirement.
7. A. The program has issued a virtual address containing an index greater than the segment length, so we should signal an illegal address fault. None of the other answers make any sense.
8. B, C, and D. The compiler is not involved in computing physical addresses, but it does have to insert the correct virtual addresses in programs it compiles, and B, C, and D are three examples.
9. A is correct. A STORE instruction could overwrite an address stored in a variable (or in executable program code). The resulting address could point anywhere, including into the library segment. If the program stores data using the corrupted address, it will then overwrite the library segment. 
   B is also correct. A LONGJMP could jump into the middle of the library executable code. If it happens to land on a STORE instruction that uses register addressing, and the register happens to contain the segment number of the library segment, the library segment could write into itself. 
   C is not correct. Lack of paging can't cause the library segment to be overwritten. 
   D is not correct. An endless loop would not itself cause a write into the library segment.
10. Extension A does not solve any of Ben's problems, since it does not protect the library segment from writes, and it does not prevent incorrect LONGJMPs into the library segment. It might be useful in enforcing some of the other solutions, but by itself it doesn't help. 
   Extension B is a reasonable solution for 9A or 9B, since a write-protect bit can be used to prevent all changes to the library segment. 
   Extension C is a reasonable solution for 9B, since it could be used to prevent the execution of inappropriate instructions in the library segment. 
   Extension D is irrelevant. 
   Extension E doesn't address any of the observed problems.
11. The implementation described for events allows a race following a test of statement A that comes out True. If, just after statement A completes, a preemption allows statements E and F to be executed, the increment by statement E will happen too late to be noticed by statement A, and the call to notify of statement F (which wakes up only those threads that are already in the queue for the event) will happen before statement B has a chance to queue the web server thread. So the sequence that causes the problem is

   ... A, E, F, B, ...

   The web server is now waiting even though a packet has arrived and needs to be processed. When the next packet arrives, the network manager will execute E and F again, and the web server will wake up to find two queued packets. (One might wonder if the network manager would overwrite the first packet with the second one. Since the problem says that the first packet is sometimes handled late, rather than saying it is lost, overwriting must not be a concern.)