Operating System & Networks

Uwe R. Zimmer

1. Introduction to operating systems

(1) What is the function of an operating system?

(2) In which circumstances is an operating system *not* required?

(3) What are the features of a µkernel based operating system?

(4) Why are many modern operating systems so big in relation to operating systems of the size of a few KB, (also delivering basic OS-functionality)?

2. Hardware basics

2.1. General computer architecture

(5) What is the minimal set of components which would make up a computer system? Which if these components can be integrated on one chip?

(6) What are the possibilities to support the design of an operating system by means of a well designed hardware?

2.2. CPU

(7) What would be a minimal register set for a CPU?

(8) Are all CPU registers of the same size? How wide (in terms of bits) need a universal register, a stack pointer, and the condition code register need to be at least?

(9) What kind of information do you expect to find on the stack? Explain what kind of programming language you are assuming. Do you always need a use a stack for passing parameters to subroutines?

(10) When enhancing the I/O performance of a small 8-bit controller: would you prefer to replace it with a 16-bit version, or to double the CPU-clock frequency? Which additional information would you request?

(11) What is the minimal hardware support you need to expect for interrupt processing? Which parts of your system are going to implement this support?

(12) How would you implement a system, which needs to be very responsive to interrupts, e.g. need to respond to external events in a defined time-span?

(13) What kind of support can you hope for from an operating system, when you need to receive external interrupts at the level of a user-task?

(14) Are interrupts buffered anywhere in the system?

(15) Put signals, traps, exceptions, and interrupts in relation.

(16) Can you implement memory protection without privileged instructions?

(17) How do you execute a privileged instruction while in a user-level task?

2.3. Memory

(18) Why is precise knowledge about the cache configuration essential for an efficient and/or predictable operating system implementation?

(19) In which situations is a cache improving the transfer rates? What level of speed-up could you expect at the best and at the worst?

2.4. I/O systems

(20) Is it possible to construct a computer system without I/O devices?

(21) Assuming your system is also using direct memory access (DMA) to transfer data between your I/O devices and the main memory. Will you give the CPU or the DMA controller the higher priority on the main system bus?

(22) When would you *not* apply interrupt driven programming?

2.5. Some examples of µprocessors

(23) What makes the difference between a µprocessor and a µcontroller?

(24) Power consumption is a major aspect of many µcontroller applications. In which range would you expect to the power consumption to be for a typical modern µcontroller.

3. Processes

3.1. Processes and threads

(25) What is the difference between general multiprocessing and symmetric multiprocessing? What does this difference imply for the operating system?

(26) What are the major issues, which needs to be addressed, if you enhance a single-task operating system to a multi-tasking operating system?

(27) Name some applications, where a multi-threaded (concurrent) implementation improves the overall performance and some where it is required.

(28) What are the alternatives to multi-threading, if you need to keep a certain responsiveness, e.g. for user requests?

(29) After the operating system dispatched the CPU to a process, how does the operating system itself get access to the CPU again at a later stage (assuming that the process is not terminating)?

(30) Can you add or subtract CPUs from a running multiprocessing computer system?

(31) When are the conditions under which the tasks in the 'blocked' state are re-evaluated?

3.2. Shared memory based synchronization

(32) Why is there a need for synchronization in the first place?

(33) What are the principal hassles of synchronization?

(34) Synchronization is performed in computer systems continuously and on all levels. Could you give some examples from nature, where synchronization occurs?

(35) Do you need hardware support for the implementation of semaphores? Why?

(36) What exactly distinguished a semaphore from a flag based synchronization?

(37) Give some reasons why to use semaphores, and some why not to use them.

(38) Implement a solution for the dining philosophers problem (if you do not know it: look it up) by using semaphores or monitors only.

(39) Which kinds of waiting/blocking/suspension can be implemented? Is it possible to avoid busy-waiting on all levels? Is busy-waiting always less efficient?

(40) Sketch a proof showing that protected objects, monitors, conditional critical regions and semaphores are all equivalent in terms of their principal synchronization capabilities, e.g. the same class of synchronization problems can be solved by all or none of them.

(41) Discus an implementation of releasing nested monitor calls in POSIX, e.g. if there is a call to a mutual exclusive region from within a mutual exclusive region, the first lock might be released under certain circumstances.

(42) Which solution for the 'multiple-tasks-in-monitor' problem, when using conditional variables would you prefer and why'?

(43) Spinlocks (a semaphore implementation) offer the possibility to busy-wait on the release of a lock, if the process holding the lock is currently running. Explain under which circumstances this can be applied, and what are the potential benefits and problems.

(44) Design an example, where object oriented expansion breaks the consistency of a monitor, e.g. is introducing a deadlock situation.

3.3. Message based synchronization

(45) Sketch a proof that message based synchronization and shared memory based synchronization are equivalent in terms of their principal synchronization capabilities, e.g. the same class of synchronization problems can be solved by all or none of them. One way: try to emulate each of them with the other.

(46) Compare message based synchronization and shared memory based synchronization in terms of speed/overhead.

(47) Give examples when platform independent, typed message passing systems are required and when a non-formatted byte-stream transfers are sufficient.

(48) How can you transfer class-attributes (assuming you are using an object-oriented language) over a message passing system. Hint: Ada is supplying routines for the marshalling and un-marshalling of object oriented (tagged) types. Java is supplying means of referring to a class on another platform (not to pass messages to it) - discuss what it takes to implement these communications (for experts only – skip, if you are not experienced in object oriented programming).

(49) Sketch an implementation which is transferring arbitrary data-structures through a byte-stream channel using C++ and POSIX only.

3.4. Deadlocks

(50) What is the difference between a safe, an unsafe, and a deadlocked system?

(51) On which basis can a system be declared ‘safe’?

(52) What is the difference between a deadlock and a lifelock?

(53) In which systems would you suggest to implement Banker’s algorithm?

(54) What can be done, if a deadlock is detected?

(55) Can a single process be deadlocked?

(56) Determine the computational complexity of Banker’s algorithm.

3.5. Scheduling

(57) Which scheduling scheme would you suggest to implement in your laptop operating system?

(58) Which information about the processes in your laptop could you supply to the scheduler?

(59) How is the scheduler gaining access to the CPU, in order to pre-empt a process, which is currently running?

(60) Why is FPS much more frequently implemented and employed than EDF?

(61) What is the definition of an ‘optimal’ priority ordering scheme in a FPS system?

(62) What does the worst case response time analysis provide (with respect to a simple utilization test)?

(63) Give the computational complexities for the response time analysis in case of EDF and FPS!

(64) What are the possibilities to handle high-priority unpredictable, asynchronous tasks in a hard real-time environment?

(65) List some features of practical task-sets, and describe how these might be addressed by a scheduling algorithm!

(66) Why should aperiodic/sporadic/unpredictable tasks have a higher priority than periodic tasks in some cases, and how do you still guarantee the schedulability of the periodic task-set?

(67) Explain the idea of the immediate ceiling priority protocol.

(68) Why are ceiling priority protocols superior to priority inheritance?

(69) The worst case scenario for FPS is to release all tasks at once. What is the worst case release order for EDF?

4. Memory

(70) What services could be offered by an operating system in terms of memory?

(71) Which kind of memory module would you expect to find in the operating system on your laptop? Which optimization criteria would you apply?

4.1. Hardware structures (MMU)

(72) Do you always need an MMU to implement virtual memory?

4.2. Partitioning, Segmentation, Paging & Virtual memory

(73) Explain internal and external fragmentation.

(74) In which systems is internal or external fragmentation a real problem?

(75) Analyse the timing behaviours of different virtual memory address translation schemes.

4.3. Virtual memory management algorithms

(76) Which information would you collect before designing a set of virtual memory management algorithms?

(77) What are the major possible disasters which could happen, if the virtual memory management algorithms are set up badly?