Mid-Semester Quiz

☞ *to all students of ANU/FEIT/DCS/COMP2310*

General

The questions below do not cover the whole of the material of the course up to the mid-semester break, but are meant to give you feedback about your performance. Do this quiz as soon as you can – do not put it on the stack until just before the final exam (it *will* be too late to catch up then). If you can go through all of those questions and can answer them based on a solid understanding of the material, you are on the right track, and should keep going. If you are hesitant or even lost with some of those questions: start reading again (your lecture notes (it should be all there), and the text books), talk with your fellow students (maybe you missed just the lecture, when this material was presented), talk with the tutors, or talk to one of the lecturers of the course, until you gained enough confidence. To make sure that your confidence is not an illusion: exchange your answers with your fellow students, and post critical ones to the phorum (but please only critical ones). It still doesn't necessarily mean that the answers are correct, but we will interfere on the phorum, if you're thinking in a wrong direction. We will respond to your questions and postings (if we have the impression that you worked through the material in the first place), but we will not supply an example solution. Some questions don't have a single, unique answer, but are meant to make you think for a little while.

1. Basic Concurrency

(1) When does concurrency *not* make sense?

(2) Specify a process

(3) Specify a thread

(4) What are the differences between a user-level thread and a kernel-level-thread

(5) Could you sketch all possible process states and their transitions?

(6) Which programming paradigms are more suitable for concurrent systems, which are less suitable? Which are intrinsically concurrent?

2. Mutual Exclusion

(7) When do you need mutual exclusion?

(8) What are the minimal requirements to be able to implement mutual exclusion between two tasks? / more than two tasks?

(9) The most simple algorithms implementing mutual exclusion between two tasks are Decker's and Peterson's algorithm. Describe both of them and determine what the do differently, but still achieving the same goal.

(10) Specify a semaphore.

(11) Is a binary semaphore sufficient to implement all forms of semaphores? / all forms condition synchronization? Why?

(12) Give examples of specific, realistic hardware-support in the implementation mutual exclusion.

(13) Sketch a simple method to implement mutual exclusion between two tasks based on atomic access to shared variables / or based on semaphores.

(14) What is the difference between starvation and a livelock?

3. Condition Synchronization

(15) When would you chose which kind of synchronization method (assuming you have a free choice in your specific development environment)? Give reasons.

(16) Which potential problem arises, when you try to suspend a process on a condition variable inside a monitor routine? How could you solve this problem?

(17) Does it makes sense, or is it even necessary to release multiple processes, which are waiting at a condition variable at once, when some other process signals this condition.

(18) Are there any differences between a semaphore and a condition variable?

(19) Can you call another monitor-routine from within a monitor-routine?

(20) Why are multiple processes allowed at once inside a protected function?

(21) Can you emulate synchronous message passing by asynchronous message passing? If so: how close can such an emulation come?

4. Non-Determinism

(22) If you have a client-server system at hand, where the server uses a non-deterministic select statement to offer several different services to the clients and you would like to change this part of the server into a deterministic program: what would you need to change? - and can you do this without losing any functionality?

5. Scheduling

(23) Which scheduling scheme (or combinations of them) would you prefer to have on your own computer? Give reasons.

(24) Assuming that you have only one processor in your system, and one of the user-tasks is currently executing: How can/will the scheduler gain control n order to suspend the current task? (If a user-task is currently executing, it also means that the scheduler (or any other part of the OS) is currently not running, right?).

(25) In order to come up with a good scheduling system, you need to include as much knowledge as you can get about your tasks. What do you know (or can you extract) about the tasks in a standard desktop-operating system?

(26) Why is Fixed Priority Scheduling often preferred over Earliest Deadline First scheduling?

(27) What is the difference between a utilization test and a response time analysis?

6. Safety and Liveness

(28) What is the rationale behind linear waiting?, i.e. why isn't the much easier FIFO scheme always used instead?

(29) Is there a deadlock-safe synchronization scheme?

(30) Which of the deadlock conditions can you break the easiest? – and in which kinds of environments?

(31) How can you break/prevent the circular-wait deadlock condition?

(32) What are the minimal requirements which need to be fulfilled if you plan to apply deadlock avoidance?

(33) What does it mean that a system fails 'uncontrolled'?

7. Architectures

(34) What are the differences between a multi-tasking, a multi-programming, and a multiprocessor system?

(35) UNIX systems come in many different flavours and architectures. Name some common features.

(36) What are the drawbacks of a fork-style process creation? What are the advantages?

(37) File-oriented interfaces are the standard in UNIX. Give examples for forms of communication / data-exchange where file-oriented interfaces are not adequate.

(38) Why are micro-kernel operating systems more supportive for distribution (of the operating system itself) than most other of operating systems architectures?