University of Glasgow

Monday 16 May 2011
2.00 pm – 4.00 pm
(Duration: 2 hours)

DEGREES OF MSci, MEng, BEng, BSc, MA and MA (Social Sciences)

ADVANCED OPERATING SYSTEMS (M)

(Answer 3 out of 4 questions)

This examination paper is worth a total of 60 marks

You must not leave the examination room within the first hour or the last half-hour of the examination.

1. **(a)** It is possible to determine whether a system of n independent periodic tasks, scheduling in a pre-emptive manner on a single processor, can be scheduled using the rate monotonic algorithm using a *maximum schedulable utilization* test. What is the expression for the maximum schedulable utilization in such a system? What are the implications for scheduling the system if its utilisation is greater than the maximum schedulable utilisation?

[3]

(b) How does the maximum schedulable utilisation for a system of rate monotonic tasks change as their relative deadlines increase to be greater than their periods?

[1]

(c) An alternative to the maximum schedulable utilisation test is to perform time demand analysis of the behaviour of a system at its critical instants. Describe when the critical instants of a task occur, outline what time demand analysis is, and discuss how it can be used to determine if a system can be scheduled.

[6]

(d) When sporadic tasks are introduced into a priority-scheduled system of periodic tasks, it becomes necessary to incorporate an *acceptance test* into that system. Describe the purpose of an acceptance test, and why is it important for error handling.

[5]

(e) Are sporadic tasks incompatible with hard real-time systems? Discuss.

[5]

2. **(a)** A system of independent periodic tasks is to be scheduled using the pre-emptive rate monotonic algorithm on a single processor. Those tasks are potentially subject to blocking due to conflicting resource access by other tasks in the system. How should one take into account this potential blocking when determining if the system can be scheduled using a maximum schedulable utilisation test?

[4]

(b) Two ways of managing resource access are the *priority inheritance protocol* and the *priority ceiling protocol*. Describe how each of these resource access control protocols works, outlining the rules by which priority is inherited, and when access to resources is granted.

[12]

(c) How does the maximum blocking time for a job that has a resource conflict with a lower priority job differ between the *priority ceiling protocol* and the *priority inheritance protocol*?

[4]

3. We discussed the following two papers in the lectures and tutorials:

- J. Shapiro. *Programming language challenges in systems codes: why systems programmers still use C, and what to do about it*. Proceedings of the Workshop on Programming Languages and Operating Systems, San Jose, CA, USA, October 2006. ACM.

- E. Brewer, J. Condit, B. McCloskey, and F. Zhou. *Thirty years is long enough: Getting beyond C*. Proceedings of the Workshop on Hot Topics in Operating Systems, Santa Fe, NM, USA, June 2005. USENIX.

These papers outline the authors' opinion on why the C programming language is not appropriate for systems code in modern operating systems, and outline some ways in which its limitations can be mitigated. Outline the key arguments expressed, and discuss the extent to which you believe the arguments outlined in these papers. Should future operating systems be written in C? Justify your answer.

[20]

4. **(a)** The traditional approach to supporting concurrency in programming languages and operating systems has been to provide *threads*, *shared memory*, and *locking*. Examples of such support include the `pthreads` API for POSIX-derived systems implemented in C, and threads with synchronised methods/statements in Java. While this is clearly a sufficient way to provide concurrency, it has been found to have a number of limitations when it comes to building reliable systems. With the aid of examples, describe the limitations inherent in this method of providing support for concurrency, and outline the type of problems it causes in the development of robust software.

[10]

(b) Two alternative abstractions for concurrency are *transactional memory* with automatic rollback and retry; and communication via *message passing*, where copies of immutable data are passed between shared-nothing processes. While one could implement both abstractions in a system, a cleaner architecture would be based around one single abstraction for concurrency. Which of these abstractions do *you* think would be the an appropriate basis for future systems programming? Justify your answer, and explain your rationale for making this design choice.

[10]