University of Glasgow

Tuesday, 7 May 2013
2.00 pm - 4.00 pm
(2 hours)

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

COMPUTING SCIENCE (M): ADVANCED OPERATING SYSTEMS

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.

INSTRUCTIONS TO INVIGILATORS: Please collect all exam question papers and return to the School together with the exam answer scripts

1. **(a)** The rate monotonic algorithm is widely used for scheduling sets of independent, preemptable, periodic tasks on single processor systems. It has been shown, however, that the rate monotonic algorithm is non-optimal. Briefly outline what it means for a real-time scheduling algorithm to be non-optimal.

[2]

(b) One approach to demonstrating the correctness of a rate monotonic schedule for a set of periodic tasks is by using *time demand analysis*. Describe how time demand analysis works, and explain why it is preferable to exhaustive simulation of the system.

[8]

(c) While the rate monotonic algorithm is non-optimal in general, it has been shown to be optimal for sets of *simply periodic* tasks where the relative deadline of each task is greater than or equal to the period of the task. Define what is meant by simply periodic tasks. Prove that the rate monotonic algorithm is optimal for such tasks.

[10]

2. **(a)** Many real-time systems are embedded, and must interact with the wider environment through the use of custom hardware resources. The mechanism by which tasks that run on the system can gain access to those resources is controlled by some form of resource management protocol, for example the *priority inheritance protocol* or the *priority ceiling protocol*. Explain what are the main benefits of the priority ceiling protocol over the priority inheritance protocol. State what extra information about each task is needed for the operation of the priority ceiling protocol.

[4]

(b) The stack-based priority ceiling protocol always allocates a resource to a running task that needs to access that resource. Explain how this protocol ensures there are no resource conflicts between tasks.

[4]

(c) Access to hardware resources is managed by device drivers. Many operating systems use an object-oriented design for their device drivers, with an implementation of the design in C – a language that provides no support for object-oriented programming. The use of C++ in the MacOS X kernel I/O Kit Framework shows that it is possible to use higher-level languages within an operating system kernel, but this idea has not found favour in the industry. Indeed, Linus Torvalds, the inventor of Linux, expressed a common view when he said "Trust me – writing kernel code in C++ is a...stupid idea" in a message to the Linux kernel mailing list in 2004. Is Linus right in his criticism of C++ for kernel development? Discuss the advantages and disadvantages of using a higher-level language, such as C++, to implement device drivers, compared to the more traditional, C-based, device driver framework implemented in Linux, and many other systems.

[12]

3. **(a)** Many modern systems integrate garbage collection into their run time support libraries. The mark-sweep algorithm is one of the simplest garbage collection algorithms. Describe how the mark-sweep algorithm works, and outline three problems that limit the usefulness of this algorithm.

[10]

(b) An alternative to the mark-sweep algorithm is to use a copying garbage collector. Explain what data is copied, and what are the source and the destination of the copy. Discuss why the process of copying the data makes a copying collector more efficient than a mark-sweep collector.

[7]

(c) Explain why a copying collector cannot be used to implement a conservative garbage collector for the C programming language.

[3]

4. **(a)** We discussed the actor model for programming concurrent systems, where the program comprises a set of shared-nothing processes communicating by the exchange of immutable messages. Many actor-based systems adopt a *let-it-crash* approach to error handling, whereby responsibility for handling failures in a task is pushed to a separate supervisor task. Discuss whether the let-it-crash model is an appropriate way of providing robustness in a massively concurrent system, or if in-process error detection and recovery is more suitable. Your answer should include an explicit discussion of the trade-offs between the two approaches.

[10]

(b) The introduction of loosely-coupled message-passing operating systems such as Barrelfish has the potential to blur the boundary between local and remote resources, since it becomes conceptually as easy to message a cloud-based service as a service running on another processor in the same hardware chassis. Does the distinction between local and remote resources matter any more? Discuss, and justify your answer.

[10]

END OF QUESTION PAPER