University of Glasgow

Wednesday, 15 May 2019
2:00pm - 4:00pm
(2 hours)

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

ADVANCED SYSTEMS PROGRAMMING (M)

Answer 3 out of 4 questions

This examination paper is worth a total of 60 marks.

The use of calculators is not permitted in this examination.

INSTRUCTIONS TO INVIGILATORS: Please collect all exam question papers and exam answer scripts and retain for school to collect. Candidates must not remove exam question papers.

1. **(a)** Programming languages and operating systems separate the stack from the heap, storing each in a different region of virtual memory, and managing the two regions differently. Outline what data is stored on the stack and what data is stored on the heap. Explain why it is necessary to store the stack and the heap in separate regions of the virtual address space. [6]

(b) One approach to automatic management of heap memory uses reference counts attached to each object. The runtime increments the reference count when a new reference to the object is created, and decrements it when a reference is removed. The memory allocated to an object is reclaimed when the reference count of the object is decremented to zero. Briefly outline the main benefits and problems inherent in using reference counting as a means of automatic memory management. [4]

(c) In the safe subset of the Rust programming language, a data item of type `T` can be accessed through one of two kinds of reference, represented as `&T` and `&mut T`. State what is the difference between these two kinds of reference. Describe the rules and restrictions around the existence of multiple references to the same data for each kind of reference. [4]

(d) With the aid of an example, explain why the restrictions on references discussed in part (c) of this question make it impossible to write certain classes of program using the safe subset of Rust. Discuss what is gained as a result of imposing such restrictions, and whether you think such benefits outweigh the costs of the restrictions. [6]

2. **(a)** It's common for programming languages to support concurrency by providing multiple threads of execution within a single address space, along with locks to control access to shared mutable data. This is the model adopted by the C programming language with the *pthreads* library, and by Java, for example. Discuss what are the problems inherent in this programming model, considering in particular correctness of the resulting code and composition of operations. Use pseudocode fragments to provide examples that illustrate key points. [8]

(b) As an alternative to the thread-based programming model, some languages offer support for concurrency via transactions with automatic roll-back and retry, or through message passing. Systems that use message passing can be further subdivided into those that avoid concerns about shared mutable state by making messages immutable, and those that avoid such problems by tracking ownership to prevent shared access of mutable data. Such languages and systems were discussed in the lectures, and in the recommended reading for the course. Discuss the advantages and disadvantages of each of these three approaches for systems programming. Justify your answer, stating which you consider to be the most promising approach for improving systems programming; if you think none are promising, explain why. [12]

3. **(a)** What is meant by the concept of a *strong type system* in programming languages? [2]

(b) Discuss whether it is possible, and a good idea, to write a complex systems program, such as an operating system, low-level device driver, or network protocol stack, *entirely* in a strongly typed programming language. Justify your answer. Give examples of any

operations, features, or behaviours that you believe make it difficult to write an entire system in a strongly typed language. [8]

(c) We discussed the *type-driven development* approach to designing and implementing systems programs. Describe what is meant by this approach. [8]

(d) Type-driven development can be viewed as imposing a relatively high up-front cost, by pushing the programmer to resolve certain design decisions and constraints early in the process. Discuss to what extent you think this is beneficial, or whether it's desirable to leave some decisions until later in the process—even, potentially, at the cost of leaving some inconsistencies unresolved in the system. [2]

4. **(a)** The recommended reading included Shapiro's paper entitled "Programming language challenges in systems codes: why systems programmers still use C, and what to do about it" (Proceedings of the ACM Workshop on Programming Languages and Operating Systems, San Jose, CA, USA, October 2006). This paper suggests that the C programming language is not appropriate for writing systems code, and outlines some new programming languages features that would improve systems programming. The lectures covered other topics in this space, and introduced you to the Rust programming language that attempts to innovate in the field of systems programs.

Do you agree with the thesis of this paper and the lecture discussion? Discuss the extent to which you believe that changing the programming language will help to address the challenges in building secure, high performance, and robust systems programs. Outline what features of a programming language or runtime you consider important for supporting systems programming, and what are harmful. Illustrate your answer using examples from systems programming, including features that might help or hinder their implementation, to help make your argument.

Answers may argue for, or against, the thesis of the paper and lecture discussion. Answers arguing in either way are acceptable provided they are supported by reasoned discussion, arguments, and examples. Of the 20 marks available for this question, [10] marks are available for technical discussion, [6] marks for the examples, and the remaining [4] marks are awarded for quality of written argument (i.e., the ability to structure and present a clear and coherent argument; *not* English spelling or grammar). [20]

END OF QUESTION PAPER