Sample Exam
(2 hours)

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

# ADVANCED SYSTEMS PROGRAMMING (H)
# COMPSCI 4089

Answer 3 out of 4 questions

This examination paper is worth a total of 60 marks.

1. **(a)** The Rust programming language uses a region-based approach for memory management. This introduces complexity into the type system, since it must track ownership and borrowing of data, and makes implementation of certain data structures problematic. In return, it ensures predictable and safe memory management. Other programming languages make a different trade-off, offering a simpler type system but providing fewer guarantees around memory safety or less predictable memory management. Do you think Rust makes the right trade-off for its intended uses cases and given the expected skills and experience of developers? Justify your answer. [8]

   **Solution:** Answers may legitimately argue for or against the premise. There is [1 mark] available for clearly stating an answer to the question and [2 marks] for quality of written argument. A further [5 marks] are available for technical justification of the stated answer.

   **(b)** State machines are an important aspect of systems programming. They encode the state of a resource, and describe what operations can be performed on that resource in what context. Using a combination of pseudo-code to illustrate key concepts and English prose, describe *one* way in which a state machine can be represented in Rust. Discuss the advantages and disadvantages of your chosen approach. [12]

   **Solution:** There are several ways to encode state machines in Rust, and any reasonable approach will be accepted.

   There are up to [4 marks] are available for the description of the approach, [4 marks] for discussion of the advantages and disadvantages of the approach, and [4 marks] for clearly illustrating key points using pseudo-code and for quality of written argument.

2. **(a)** Systems programs have generally been written in memory unsafe languages, such as C. It is generally accepted that this is problematic, and that it's desirable to move to memory safe languages where possible. Discuss whether it is possible to write an operating system entirely in a memory safe language, highlighting areas where this would be difficult. [5]

   **Solution:** It is not possible to write an operating system entirely using memory safe code in current languages [1 mark].

   Answers should note that some unsafe assembly code is needed to make system calls, program interrupt registers, etc. [2 marks], and that existing type systems are also insufficiently expressive to validate certain code patterns (e.g., data structures containing cyclic references cannot be implemented in safe Rust [2 marks]).

   **(b)** Is it feasible to write an entire operating system in a garbage collected language? Discuss the advantages and disadvantages of such an approach. [5]

**Solution:** It is feasible and such systems have been implemented [1 mark]. Up to [4 marks] are available for the discussion of the advantages and disadvantages, with well-reason and technically correct justification.

**(c)** Studies have shown that the majority of security vulnerabilities in deployed systems relate to a lack of memory safety. To what extent do you think it is currently feasible to use only memory safe languages for systems programming? Explain what are the costs and benefits inherent in making such a change to the implementation language. Discuss what is stopping the industry from moving entirely to memory safe languages for systems programming. [10]

**Solution:** The benefits of switching to a memory safe language are clear: there will be some reduction is memory-related security vulnerabilities and a general improvement in software correctness [2 marks].

The costs are less clearly defined. Up to [5 marks] are available for discussion of the overheads of memory safety, limitations of designs that can be expressed in safe languages, compatibility of the new safe language and unsafe prior code, and so forth. There is no single correct answer, and marks are assigned for reasoned discussion with technical justification.

Up to [3 marks] are available for discussion of what is stopping the industry from moving entirely to memory safe languages. The discussion likely focus on compatibility, training costs, inertia, and so on. Any reasoned and well justified answer is accepted.

**3. (a)** Programming languages, run-times, and operating systems can allow I/O operations to be performed synchronously or asynchronously. Explain what is the difference between these two models. Discuss why support for asynchronous I/O is frequently considered to be beneficial. [5]

**Solution:** Synchronous I/O operations block until the I/O completes [1 mark], while asynchronous I/O is performed in the background allowing the rest of the program to proceed concurrently [2 marks]. Asynchronous I/O is beneficial because I/O is slow, and waiting which it's performed is inefficient [2 marks].

**(b)** One way of implementing asynchronous I/O operations is to structure the code as a set of *coroutines*. In this model, operations that may block return a `Future<T>` representing a value of type `T` that will become available at some later time. The runtime system for such languages includes an `await` operation, that allows a function to wait for a result to become available without blocking the thread, by passing control to other coroutines while waiting. Discuss whether you think this is a good approach to writing asynchronous code. Explain what are the strengths and weaknesses of this approach to structuring code. [7]

**Solution:** Answers may legitimately argue for or against this approach to writing asynchronous code, with [1 mark] available for stating a preference, and up to [4 marks] available for discussion for the strengths and weakness of the approach. Up to [2 marks] for quality of written argument (logical structure, etc., not spelling and grammar).

**(c)** Message passing systems are often said to avoid many of the problems inherent in lock-based concurrency. However, message passing systems can still deadlock, and race conditions can still occur. Explain how deadlocks and race conditions can occur in message passing systems. Discuss whether you think they are more or less likely to occur than in systems using lock-based synchronisation, justifying your answer. [8]

**Solution:** There are [2 marks] available for explaining how deadlock can occur, and [2 marks] for explaining how race conditions can occur. Up to [4 marks] are available for discussion of whether deadlocks and/or race conditions are more or less likely in message passing systems. Arguments either way are accepted, provided they are well justified.

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

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. [20]

**Solution:** Answers may argue for or against the thesis, and any reasonable and well-justified argument is accepted. There is [1 mark] available for answers that clearly state if they are in agreement with the thesis or not. Up to [9 marks] are available for the discussion of whether changing the programming language will improve systems programming, and for reasoned argument to support the chosen position. Arguments may be technical, social, organisation, or educational, but should be justified and briefly evidenced.

A further [6 marks] are available for examples of what new language and runtime features are helpful or harmful. Any reasonable example will be accepted.

Up to [4 marks] are available for quality of written argument. This includes the ability to effectively state opinions, structure their answer, and justify the arguments made. This is not assessing quality of written English (e.g., spelling, grammar), but rather the ability to structure an argument.