University of Glasgow

SAMPLE EXAMINATION
(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)** A running program will reference memory on the stack and on the heap. 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]

> **Solution:** The stack stores function parameters, return addresses, and local variables [3 marks]. The heap holds explicitly allocated memory, such as that allocated using `malloc()` or equivalent [1 marks]. The stack and the heap need to be separate regions of the address space because stack frames are tied to a program scope [1 mark], but heap allocations can outlive the stack frame in which they were allocated [1 mark].

(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]

> **Solution:** The benefits are that reference counting is simple [1 mark] and works in an incremental manner so there is no need to "stop the world" [1 mark]. The main problems are that it cannot reclaim data structures that contain cyclic references [1 mark] and that it has high runtime overhead [1 mark].

(c) The Rust programming languages tracks data lifetimes and uses region-based memory management to automatically manage memory. With reference to lifetimes, ownership, borrowing, and the different types of references, explain how Rust manages memory. [10]

> **Solution:** The lifetime of an item of data is tied to a particular scope [1 mark]. When the data item goes out of scope, the resources it owns, including any allocated memory, are dropped [1 mark]. When a function is called, it takes ownership of its parameters, and they are dropped when the function returns [1 mark] unless otherwise returned. Ownership of function return values is passed to the caller [1 mark]. Passing a reference allows a function to borrow ownership of data [1 mark]. Borrows may be via immutable references or mutable references [2 marks]. There can be many immutable references to a data item, or a single mutable references, but not both [3 marks].

2. **(a)** Modern systems programming languages are increasingly focussing on how to provide effective support for concurrency. Discuss how the data ownership rules in Rust help avoid data races due to concurrency. [6]

> **Solution:** Every data item has a single owner [1 mark]. The only way that multiple threads can reference the same data is if they borrow a reference that data. The borrowing rules enforce that there can be many immutable references to a data item, or

a single mutable references, but not both [3 marks]. This means that there is no way for multiple threads to mutable an object concurrently, since they can't get references allowing mutation, hence there are no data races [2 marks].

(b) The Rust and Erlang programming languages both support message passing via channels to communicate data between threads. Channels in Rust are statically typed and the compiler will check that the messages exchanged via a channel conform to the stated type. Erlang is dynamically typed and any type of message can be sent to any thread. Discuss the trade-off between the two approaches. State which do you think is best for systems programming, and explain why. [6]

Solution: The statically typed approach provides more guarantees, but can be considered to make it harder to change a running system [2 marks]. For infrastructure components of systems that change rarely, the additional checks are likely useful, so it's probably better for systems programming to ensure correctness, but convincing arguments either way will be accepted [2 marks].

A further [2 marks] are available for quality of written argument.

(c) The increased focus on concurrency in programming languages is due to the increasing amount of concurrency present in microprocessors and systems. With reference to Moore's law and Dennard Scaling, explain why processors are moving to designs with increasing numbers of cores. [8]

Solution: Moore's law is the observation that advances in manufacturing technology lead to smaller transistors, such that the number of transistors in a processor can be doubled roughly every two years [1 mark].

The Dennard scaling relation is that power consumption of a processor is proportional to $C \cdot F \cdot V^2$ where C is the capacitance, F is the clock frequency, and V is the voltage [4 marks]. C and V depend on the size of transistors, so Moore's law lowers power consumption, allowing us to increase the clock frequency while maintaining the same power budget [1 marks].

The breakdown in the Dennard scaling relation below a certain transistor size means processor clock rate cannot increase due to power constraints, hence the push to use the transistors for more cores [2 marks].

3. (a) Describe how types and functions can be used to model state machines using a language, such as Rust, that tracks data ownership. Explain how this helps to manage state variables and resources. Discuss why this helps to achieve more robust software. [8]

Solution: Types model states, functions model transitions [2 marks]. The transition

function consumes the object holding the state variables for the state that is being left, ensuring any resources are cleaned-up [2 marks].

Any reasoned answer for why this makes software more robust is accepted. A typical answer would be that it ensures that the system is always in a consistent state because the transition functions clean-up resources and only allow 'sensible' transitions. [4 marks]

(b) State machines, especially those used to model network protocols, can also be represented using asynchronous code and coroutines. Discuss why asynchronous functions that can await intermediate results are considered desirable for network programming. [6]

Solution: They're considered desirable because they allow one to write functions that are structured in a manner that looks like traditional blocking calls [1 mark], but that yields rather than blocking to allow concurrent execution with the aid of a runtime [3 marks]. This allows code to be written in a natural style, without the overhead of creating multiple threads [2 marks].

(c) Discuss the extent to which you believe asynchronous programming succeeds in its goal of simplifying implementation of concurrent state machines for network code. [6]

Solution: Any well argued answer is accepted. A typical response might suggest that it's essentially re-inventing cooperative multitasking, which was well known to result in hard to debug timing and interactivity problems due to tasks either calling blocking functions rather than yielding, or due to long running computations that hog the CPU. Such an answer might argue that reducing the cost of threads is the right long-term solution, but asynchronous code is a useful stop-gap.

Alternatively, the answer might argue that such problems are overblown for typical networked applications, and that asynchronous programming offers a simple abstractions that solves real problems, without the learning curve of event-based solutions. It would likely also note that most attempts to build light-weight threads have failed.

Arguments either way are acceptable, with [1 mark] for expressing a clear opinion, [4 marks] for technical justification of that opinion, and [1 mark] for quality of written argument.

4. (a) The recommended reading for lecture 8 included the paper on "The bugs we have to kill" by Bratus, Patterson, and Shubina (USENIX ;login:, August 2015). This paper suggests that the way we build networked applications must change if we are to reduce security vulnerabilities. The authors suggest that we should move to using strongly typed and memory safe programming languages to improve overall robustness of the code, along with formally-specified grammars and parser generator tools to ensure safe parsing of untrusted data. We also discussed these ideas in the lecture, where it was suggested that

such approaches won't eliminate security vulnerabilities but, if used carefully, they might be able eliminate certain common classes of vulnerability, and make others less likely by making hidden assumptions – and their security consequences – visible.

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 and using automated parser generator tools will help to address the challenges in building secure networked systems. Outline what programming language, runtime, and parsing features you consider important for supporting secure systems programming, and what are harmful, giving examples to illustrate key points. [20]

Solution: Answers may argue for or against the thesis, and any reasonable and well-justified argument is accepted. Up to [2 marks] are available for answers that clearly state if they are in agreement with the thesis or not.

Up to [10 marks] are available for discussion of whether changing the programming language and using parser generator tools will improve the security of networked systems. It is likely this will highlight memory safety and making assumptions explicit through type-based design as reasons why changing the language might help. Similarly, making it explicit what is accepted by the parser and what is not, and eliminating difficult-to-write manual parsing code, could be benefits of using automated parsing tools. Reasons why it might be problematic might include compatibility with existing systems, performance overhead due bounds checking and the inability to effectively optimise high level and expressive code to give good performance. It might also be suggested that formal parser generators are too complex, and the benefits are small once a safe language is used. Other arguments will be accepted, if justified.

Up to [4 marks] are available for giving examples to illustrate the above points. Any reasonable examples are accepted.

Up to [4 marks] are available for quality of written argument. This includes the ability to effectively state opinions, structure the 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.