Concurrent and Distributed Systems
Exercise Sheet

Mike Cachopo

December 25, 2020

Submitting work

Submit your answers to the questions as a single PDF file by email. I am rarely in Cambridge so I can't accept paper submissions. I strongly suggest using LaTeX to typeset your answers — you'll need it next year for your dissertation, and the sooner you start practicing the better. I recommend using the Ti*k*Z package to make diagrams, but hand-drawn and scanned is fine if you are in a rush — you should prioritise your learning over making your supervision work look pretty.

The sooner you submit work the more time I have to mark it and give you detailed feedback. Aim to submit your work 48 hours before the corresponding supervision. If for some reason you can't submit work in advance, please let me know not to wait for it. I am happy to mark work submitted after a supervision in exceptional circumstances, but keep in mind that you'll be missing out on most of the benefit of the supervision.

Questions

Each supervision is split into three sections:

Bookwork

Do not include the answers to these questions in your work!

I will ask you these questions quick-fire style at the start of the supervision, so make you sure you are ready to answer them. The goals of these questions are:

- To make sure you know the key concepts you need to understand and remember to do well in the course. We have limited time in supervisions to cover everything in detail, but we can use these questions to check if there are any large holes in your understanding.
- To give you practice explaining computer systems in a precise and concise way. This is a skill you will need to answer Tripos questions on a variety of courses, if you ever write academic papers on systems research, or if you ever have to discuss system design with colleagues in an engineering role.

Exercises

After using the bookwork questions to check that you understand all of the content, you will have some past Tripos questions to do. Please submit written answers to these according to the instructions above.

I suggest practicing answering these questions as you would under exam conditions. This will help you learn how to answer questions under time pressure, and how to effectively use your time to get as many marks as possible in the exam.

Examiners have to mark 100+ scripts per course in just a few days. They will only spend a couple of minutes looking at your answer, so you should make it as easy as possible for them to see that you know what you're talking about. This means that your answer

needs to not only be correct, but also well structured. Here are some tips on answering systems Tripos questions:

- Start by answering each part of the question with a few key points. This is a good way to pick up 5-10 marks in only a couple of minutes.
- Go back and develop well-reasoned arguments where the question asks you to compare two possible solutions to a problem, or add relevant details to a description of a particular system's implementation. This should put you in the 10-15 mark range with about 10 extra minutes of work.
- Finally, add detail where you think it is relevant or may show deeper understanding. This is a great opportunity to show off some **relevant** further reading you've done, and should put you comfortably in the 15-20 mark range.

This process takes practice, so expect to need a few attempts before you get comfortable with it. In an exam, you may well choose to settle for the 10-15 mark range, or even 5-10, so that you can spend more time on another question. This kind of game of optimisation may seem sub-optimal to maximise your learning, but sadly Tripos exams are not optimal learning tools either, and keeping these strategies in mind will help you maximise your grades.

Extensions

The extension sections contain references to further reading material, which you are welcome to read if you have spare time and/or are particularly interested in the course. This material is not examinable, but understanding it will give you more perspective on the contents of the course, and better prepare you for more advanced systems courses, units of assessment, and eventually a career involving system design.

I do not expect you to write answers to any questions that may appear in these sections, but I am happy to discuss them in supervision after we cover the examinable material. If there are other advanced topics you would like to discuss, please let me know when you submit your work so that I can prepare in advance.

General advice

As an undergraduate in Cambridge, I often felt overwhelmed by all the work to be done. If there is one piece of advice I can pass down to you, is to take care of your physical and mental health first, and only then worry about your work.

If you're struggling, please do not hesitate to reach out to me, your DoS and your tutor. It is our job to make sure you get the most out of your education, and we can often help you if you ask for help.

Lastly, enjoy learning through these questions!

Supervision 1

Concurrency primitives

1.1 Bookwork

The free lunch is over Before trying to solve all the problems of concurrent programming, it's important to understand why concurrency is desirable in the first place. Read [Sutter, 2005]. We will discuss in the supervision.

Lecture 1 concepts Define and explain the following:

- Multicore processors;
- Processes, threads and the differences between them (you may want to mention the memory stacks and heap), as well as how context switches work;
- Concurrency, and the difference between time sharing a single core and true parallelism;
- User-level, kernel-level and hybrid thread scheduling, the advantages and disadvantages of each as well as your favourite language that uses each of them;
- "Mostly correct" code and why it is a problem;
- Critical sections and mutual exclusion;
- Race conditions and atomic memory operations.

Lecture 2 concepts

- In a multicore system, would it be possible to achieve mutual exclusion on critical sections by locking (having exclusive access to) the memory bus when entering a critical section? You may find it useful to refer to the slide titled "Computer Architecture Reference Models".
- What can cause a running process/thread to block?
- What are the problems with disabling interrupts to provide atomic execution of a critical section? Don't forget the ones that do not involve other cores or DMA.

- Why do we need special ISA instructions to make an atomic read-and-set? Are these instructions strictly necessary to write correct concurrent programs? You may want to mention Lamport's bakery algorithm in your answer.
- How would you explain CAS to someone who just finished Part IA of the Computer Science Tripos?
- How would you explain LL/SC to someone who just finished Part IA of the Computer Science Tripos?
- What is the difference between deadlock and livelock?
- Be prepared to act out the dining philosophers' problem in the supervision and explain where the problem lies. Can you think of a simple solution?

Lecture 3 concepts

- Why is sleeping preferable to spinning?
- Describe the implementation of a semaphore and its operations. Refer to [Bacon and Harris, 2003, Chapters 10.4–10.6] for more in-depth information than the lecture slides.
- What is an IPI? For this course you don't need to know much about low-level implementation details, but you should be comfortable explaining what they are and how they can be used at a high level.
- How can a semaphore be used to provide mutual exclusion, condition synchronisation and resource allocation? What is the starting value of the semaphore and what are the logical meanings of the operations in each case?
- Explain the producer-consumer problem. What are some real-world applications of this problem? How can you solve it with semaphores? How would you solve it with locks? You may wish to refer to [Bacon and Harris, 2003, Chapter 11.3].

Lecture 4 concepts

- It is safe for multiple threads to concurrently read a shared resource, but writes must have exclusive access. Why? What can go wrong if a read and a write happen concurrently? What can happen if two writes happen concurrently?
- Be ready to sketch out and discuss several RW lock implementations using semaphores in the supervision, focusing on their starvation and fairness properties. This should include the implementation in [Bacon and Harris, 2003, Chapter 11.5].
- What are the advantages of Conditional Critical Regions, and why do we care about them? Why is CCR not a common synchronisation mechanism?

- Where are monitors used in Java, and how do they work? This is covered in the *Further Java* companion course. If you haven't gotten there yet have a look at the Java specification.
- Why are condition variables a useful complement to monitors? What are the similarities and differences with semaphores? How are they used in Java?
- What are the pros and cons of signal-and-wait and signal-and-continue semantics? Which is used in Java? What implications does this have on how we `wait` in Java?
- How would you implement a RW lock using a monitor? (Hint: adapt the implementation using semaphores in [Bacon and Harris, 2003, Chapter 11.5].)
- What concurrency primitives do C++ pthreads support?

1.2 Exercises

Reading and writing 2010 Paper 5 Question 4

Semaphores and monitors 2000 Paper 3 Question 1

1.3 Extensions

Real-life synchronisation Explain to me the solution to [Herlihy and Shavit, 2008, Chapter 1, Exercise 4].

Java concurrency Ask me any question about concurrent programming in Java that I cannot answer. (You will most likely succeed if you try hard enough, but I'm curious to see what you'll come up with.) You may find it useful to refer to the Java specification.

Double-checked locking Read [Bacon et al., 2000]. What is your favourite solution to the double-checked locking problem with the singleton pattern in Java?

Progress guarantees What does it mean for an application to be:

- lock-free;
- wait-free;
- obstruction-free?

What does it mean for a particular operation within an application to have these properties?

Supervision 2

Transactions

2.1 Bookwork

Lecture 5 concepts

- This is repeated from last supervision, but it's worth making sure you remember: What is the difference between deadlock and livelock?
- What are the requirements for the possibility of deadlock? How is each of them shown in the "Resource allocation graph" in the slides? How would you remove each of the requirements?
- If you were writing a concurrent application, how would you deal with deadlock? This question is deliberately very open-ended, we will discuss in supervision.
- One way to prevent hold-and-wait is to request all resources simultaneously. Why can this be a hard thing to do?
- Again, this is a repeat from last supervision, but now you should make sure you can do it: What is a simple solution to the dining philosophers problem?
- TODO: Banker's algorithm?
- What are some pros and cons of the thread scheduling policies on the slides?
- Sketch out an example scenario of priority inversion without using a reference. How would priority inheritance help here?

Lecture 6 concepts

- Why are higher-level concurrent programming models useful? What higher-level models are there, and what are some examples (languages that use them, libraries, applications, etc) of each?
- Explain the programming model of active objects. What is it similar to? How does it provide mutual exclusion and condition synchronisation?

- Explain message passing. How does it differ from active objects? What is the difference between synchronous and asynchronous message passing?
- Why are "composite operations" important? What are some examples of applications that may use them?
- Explain the problems shown in the composite operation example in the slides.
- What are the ACID properties?
- What is serialisability? How is it different from a serial execution?

Lecture 7 concepts

- What kinds of operation conflicts are there? What are the problems caused by each?
- What is the difference between strict isolation and non-strict isolation?
- What are cascading aborts, and why are they a problem?
- How does two-phase locking work? What algorithmic difference is there between a version of 2PL that offers non-strict isolation and one that offers strict isolation? Can you think of a way to avoid the need to know the locks required in advance?
- Why might a transaction abort?
- What approaches are there to rollback a transaction?
- How does timestamp ordering work? What timestamp is given to a transaction as it begins? At commit time, what needs to be checked? What are its pros and cons compared with 2PL?
- What is optimistic concurrency control? You may want to read [Bacon and Harris, 2003, Chapter 20.6] for this one, because the slides are somewhat misleading. What are its advantages and disadvantages?
- Explain the OCC algorithm used in the slides. What's wrong with the "OCC example (2)" slide? As in, how is it being overly conservative? (Hint: can a write-only transaction conflict with any committed transaction?)
- In an application with a high degree of contention — many concurrent transactions conflict, forcing each other to abort — what is the best synchronisation mechanism?

Lecture 8 concepts

- What is a crash? What can cause it? What assumptions does the fail-stop model make, and how can they be broken?
- Why does "writing all objects to disk on commit" not work? How is write-ahead logging different?

- What is the bare minimum information that entries in a write-ahead log must hold? What does that information let us do when a crash occurs?
- Why is lazy write back desirable? What problems does it introduce, and how are they solved with a well placed flush? Does batching change the guarantees provided by the log?
- What benefits do checkpoints give us? In supervision, draw a diagram and explain the algorithm for restarting from a checkpoint.
- In 60 seconds or less, sell me a transactional system. As in, convince me that transactions are a "good" model to use to write a concurrent application (you have to tell me what "good" means too!)
- What are the pros and cons of lock-free programming? What is linearisability? If I asked you to implement a lock-free data structure, such as a linked list, what would you reply?
- What is transactional memory? Why is it "promising"? Why is it not widely used today?

2.2 Exercises

TSO and OCC 2007 Paper 4 Question 1

TSO details 2011 Paper 5 Question 7

2.3 Extensions

Message passing If you want to know more about message passing models read up on Hoare's Communicating Sequential Processes (CSP) or Milner's Communicating Sequential Systems (CCS). If you would like a reasonably clean introduction to how CSP can be used in a programming language, have a look at A Tour of Go, in particular the section on channels.

BASE What are the BASE properties? Why would they be more desirable than ACID?

Lock-free programming The *Multicore Semantics and Programming* Part II Unit of Assessment has a two hour lecture on lock-free programming. If you want to know more, it's a good place to start.

Transactional memory For a relatively short intro to Software Transactional Memory, check out [Herlihy and Shavit, 2008, Chapter 18]. For a deeper understanding of why STM is a very powerful tool, read [Harris et al., 2005].

Supervision 3

Distributed systems and ordering

3.1 Bookwork

Introduction to distributed systems Let's talk about distributed systems from an informal perspective. Some topics to cover:

- Intuitively, what is a distributed system? Why are some applications distributed? What are some examples?
- You will learn a lot about TCP/IP, DNS, HTTP and network latency and throughput in the Networks course in Lent term. For now, let's just make sure you know the basics needed to understand communication between processes.
- RPC is a very powerful and widely used framework. It is not very relevant at the level of abstraction usually kept in this course, but you will likely encounter it in many software engineering roles. Let's discuss this briefly. Do you see the similarity with active objects, from the concurrent part of the course? What are the key differences?
- Distributed systems are extremely complex, and there are a lot of problems to solve in the area. Let's briefly talk about the specific problems addressed in this course.

Models of distributed systems Now that you have an intuitive feel about distributed systems, let's discuss some ways in which we can model them. Some topics to cover:

- How would you describe the two generals problem to your Engling friend? This is an application of what general problem in distributed systems? Can you think of another application?
- What assumption made in the two generals problem is no longer true in the Byzantine generals problem? Can you think of a real-world application of this problem?
- What are the three types of behaviour we have to model in a distributed system? What are the most common models of each type, and what are their properties? Which can we assume in the real-world?

- We will talk more about availability later, when we talk about consistency, but feel free to ask any questions you have now.
- What are faults and failures? Why is the difference important? How do we detect them in practice?

Time, clocks and ordering In the concurrency part of the course, we saw that being able to order events is quite a neat ability. For example, when we attempt to serialise transactions, it's useful to be able to say that one transaction is serialised *before* another. As it turns out, ordering is much more difficult in a distributed system.

- Read the first two pages of Lamport's paper on time [Lamport, 1978]. Why must orderings in a distributed system be only partial orders?
- What kinds of physical clocks are there? What is each used for? What are the shortcomings of each?
- What representations of physical time are there? How is each of them useful?
- What is the purpose of NTP? How does it achieve this goal? In words, how does an NTP client estimate clock skew from an NTP server? What assumptions are made here? Are they fair assumptions to make?
- Why might it be useful to order messages in a distributed system? Can you think of a real-world example that does not involve discourse about cheese?
- How is a happens-before relationship on the events in a distributed system defined?

Broadcast protocols and logical time Now that we've seen how impractical physical time is in a distributed system, let's focus on logical time. A word of advice: whenever you start thinking about a logical clock, make sure you know what the clock is ordering. Is it ordering messages? Events? Transactions? Forgetting to state this explicitly will lead to misunderstandings and bugs that are difficult to pin down!

- In 60 seconds, explain Lamport clocks. You may want to mention:
  - What is being ordered?
  - What are the steps in the algorithm?
  - How do Lamport clocks relate to a causal happens-before relationship?
  - What does it mean for two events to have the same timestamp?
  - How can a total order be formed from Lamport timestamps?
- As above, but now for vector clocks.
- In 30 seconds, explain the isomorphism between vector clocks and sets of events.

- What are the trade-offs between Lamport clocks and vector clocks? How does each of them scale with the number of processes and the amount of communication? How does each of them handle dynamic process groups?
- What are broadcast protocols, and why are they useful?
- What is the difference between "receiving" and "delivering" a message? At which layer does each of them happen? Why is this distinction useful?
- Let's have a free-form discussion about ordering of broadcast messages. Please come prepared to explain each of the ordering schemes presented in the course, why they might be useful, and how they can be implemented.
- Why is FIFO-total order broadcast strictly stronger than causal broadcast?

3.2 Exercises

Distributed time 2001 Paper 9 Question 1

Vector clocks 2005 Paper 9 Question 4

3.3 Extensions

GRPC If you end up writing parts of a distributed system, there's a good chance that you'll have to use GRPC or a similar framework. Feel free to do some research on it and discuss in supervision.

Catchup In the last supervision we will be covering consensus and distributed consistency, which are both very complex problems. Now is a good time for you to ask any questions you have on the course so far, as we are less likely to have time for them in the final supervision.

Supervision 4

Topics in distributed systems

4.1 Bookwork

In the last four lectures there are a lot of relatively isolated topics, and we won't have time to cover them all in the supervision. Instead of trying to go through all of them sequentially, we will:

- Briefly discuss the motivations of replication;
- You will have time to ask any questions you have about the course;
- You will each choose two topics from the last four lectures and explain them in depth in the supervision, in the style of a short presentation.

Instructions for the topic presentations:

- Agree with your partner beforehand which topics each of you is presenting, so there is no overlap.
- Aim to talk for about 5 minutes per topic, and allow for 1-2 minutes of questions/discussion.
- Prepare what you're going to say in advance. You may find it useful to have some written notes in front of you in the supervision.
- Do some reading outside of the course for each of your topics. One paper or textbook chapter per topic is sufficient. The recommended textbooks and references section of the lecture notes are a good place to start.
- You may want to choose one topic you're comfortable with, to focus on your presentation/technical discussion skills. These often transfer into the ability to write clear Tripos answers.
- You may want to choose one topic you are struggling with, to give you the chance to become more familiar with it when preparing your presentation.

Replication

- What is data replication? Why do we want it?
- Explain the complexity introduced by retrying state updates in a distributed system. You should cover:
  - Why do retries exist in the first place?
  - Given that retries are necessary, what types of retry behaviour (semantics) can an application implement?
    You should state these as *If a client sends a request, then the server will* ***deliver*** *that request X times.*
  - What is idempotence? Why is it useful?
  - Can idempotence be implemented using message deduplication? This is an open-ended question; be ready to defend your argument.
- Are exactly-once semantics "enough" in the presence of multiple clients or servers? Why?

Presentation topics As described above, choose two of the following:

Quorums In particular, explain:

- The requirements of a quorum system, and why they are important;
- The performance tradeoffs of different choices of read and write quorum size;
- Different methods of achieving consistency, such as read repair.

Consensus in general I do not want you to explain Raft here, you're already doing that in one of the written exercises! Instead, I want you to focus on some of the following:

- How modern implementations of consensus algorithms differ from the traditional formulation of the problem;
- The PLF result;
- The motivations behind the creation of Raft, and how it differs from Paxos.

Two-phase commit Including:

- A comparison with consensus;
- A discussion of fault tolerance.

Linearisability Including how it relates to happens-before and to physical time.

The CAP theorem In particular, explain:

- What each of the properties is, and how the meaning of consistency differs between contexts;

- Different possible consistency guarantees;
- The real-world implications of the CAP theorem;
- At least one of BASE and PACELC.

Others Any of the following, with the caveat that they are likely not examinable:

- CRDTs;
- A case study of any distributed system, like the one of Spanner in the slides. You may select Spanner or any other system that is relevant to the course.

4.2 Exercises

Third topic Select one of the presentation topics from the bookwork section and explain it. This topic must not be one of the two you picked for the verbal presentation in supervision, but it can overlap with your supervision partner's.
Do this exercise as if you were answering a 10 mark Tripos question part. The purpose of this exercise is for you to practice structuring medium-length Tripos answers, and in particular to select the most relevant information to include in your answer.

Raft Explain in detail the purpose and operation of the Raft algorithm.
Do this exercise as if you were answering a 20 mark Tripos question. The purpose of this exercise is for you to practice structuring long Tripos answers, writing clearly and concisely, and ensuring that even long answers are understandable. You are not expected to remember all the details of Raft in a Tripos exam, so you should answer this question open-book.
You may also find the visualisation of Raft useful as a source of inspiration.

4.3 Extensions

As mentioned in the bookwork section, the last four lectures of the course cover a variety of topics, but each of them only briefly. If you are interested, you can easily learn more about each of them, and incorporate more of your findings in your topic presentations!

Bibliography

[Bacon et al., 2000] Bacon, D., Bloch, J., Bogda, J., Click, C., Haahr, P., Lea, D., May, T., Maessen, J.-W., Mitchell, J., Nilsen, K., et al. (2000). The "double-checked locking is broken" declaration.

[Bacon and Harris, 2003] Bacon, J. and Harris, T. (2003). *Operating Systems: Concurrent and Distributed Software Design*. International computer science series. Addison-Wesley.

[Harris et al., 2005] Harris, T., Marlow, S., Peyton-Jones, S., and Herlihy, M. (2005). Composable memory transactions. In *Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '05, page 48–60, New York, NY, USA. Association for Computing Machinery.

[Herlihy and Shavit, 2008] Herlihy, M. and Shavit, N. (2008). *The Art of Multiprocessor Programming*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Lamport, 1978] Lamport, L. (1978). Time, clocks and the ordering of events in a distributed system. *Communications of the ACM 21, 7 (July 1978), 558-565. Reprinted in several collections, including Distributed Computing: Concepts and Implementations, McEntire et al., ed. IEEE Press, 1984.*, pages 558–565. 2000 PODC Influential Paper Award (later renamed the Edsger W. Dijkstra Prize in Distributed Computing). Also awarded an ACM SIGOPS Hall of Fame Award in 2007.

[Sutter, 2005] Sutter, H. (2005). The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3):202–210.