Concurrent and Distributed Systems
Exercise Sheet

Mike Cachopo

September 25, 2020

Questions

Each supervision is split into three sections:

Bookwork
: This section focuses on the key concepts you need to understand and remember to do well in the course. It's not worth including these definitions in your answers, but they should be on the tip of your tongue if you're asked about them in a supervision! If there is anything here you don't understand please send me an email, so that we can address it in advance and you can take the most out of the supervision.

Exercises
: This is where I expect you to spend most of your time and effort. If you get stuck on a question, move onto the others and come back to it later. If you're still stuck, either send me an email or leave a note to discuss during the supervision.

Extensions
: If you have some spare time or interest after finishing the core exercises, have a look at the extensions. They cover topics that are just beyond the scope of the course, so attempting these questions is optional. However, spending some time here will give you a deeper understanding of the course, which will be helpful later.

Submitting work

Submit your answers to the questions as a single PDF file TODO: email address?. 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.

The sooner you submit work the more time I have to mark it and give detailed feedback. Aim to submit your work 48 hours before the corresponding supervision. If for some reason you can't do that, 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.

General advice

In Cambridge, it is easy to feel overwhelmed by all the work to be done, and like your supervisors expect your life to revolve around work. They don't, and it shouldn't. Take care of your physical and mental health first, and then your work.

If you're struggling, let your relevant supervisors, DoS and Tutor know so that they can help. TODO: Maybe include a link to a student helpline?

Lastly, enjoy learning through these questions!

Contents

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 concurrent programming 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 Answer the following questions:

- 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, but don't spend much time trying to understand it.
- How would you explain CAS to someone who never heard of it?
- How would you explain LL/SC to someone who never heard of it?
- 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 Answer the following questions:

- 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 Answer the following questions:

- 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++ phtreads support?

TODO: processes, threads, interrupts, locks, semaphores, condition synchronisation, lock-free/wait-free, obstruction-free, deadlock, livelock, priority inversion, transactions, serialisability and linearisability

1.2 Exercises

1.3 Extensions

Supervision 2

Transactions

2.1 Bookwork

Lecture 5 concepts Define the following terms:

- 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 Define the following terms:

- 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 Define the following terms:

- 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 Define the following terms:

- 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

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].

TODO: lock-free, wait-free, obstruction-free, slide 10 AND-OR wait-for graphs

Supervision 3

Distributed systems and ordering

3.1 Bookwork

Lecture 9 concepts

- What is a distributed system? Name a few and say what they (try to) accomplish.
- Why is a distributed system inherently a concurrent system? What additional difficulties are there? Why would we want one in the first place, if it is even more complex than a concurrent system?
- What protocols do you implicitly use when accessing this exercise sheet? You don't have to be exhaustive, but I want to see you thinking about this. What important conclusion can you draw from this exercise?
- What is transparency? Why is it useful? How much do we actually want?
- How would you describe a client-server application to your Engling friend?
- What are synchronous and asynchronous requests? What is an appropriate use of each?
- What are examples of errors, faults and failures?
- What are retry; exactly-once; all-or-nothing; at-most-once and at-least-once semantics?
- What is RPC? Why is it a more useful abstraction than request/response? What does "useful" even mean here?
- What is marshalling? What is an IDL? How is all this done in Java? (Hint: this is covered in the Further Java course.)

Lecture 10 concepts

- Make sure you brush up on the file system API in Unix. What are the five procedures to handle a file exposed to a programmer?

- How would you describe NFS to someone who just finished Part IA of the Computer Science Tripos?
- Why is a virtual file system a useful layer of abstraction to use in NFS?
- What does "stateless" mean in the context of NFS? Why would a stateful design make fault recovery more difficult?
- In 60 seconds or less, what are the implications of statelessness on the interface between the NFS client and server?
- The `readdirplus` procedure in NFSv3 is a good use of batching. Why is it useful? Can you think of a situation where batching would be undesirable?
- What is inconsistency in the context of NFS? What feature of NFSv3 causes it? How can programmers work around it? At what granularity does this work?

Lecture 11 concepts

- Object-Oriented Middleware is no longer part of the course, but it is important to have a high-level understanding of it if you ever write a distributed application in an OO language. How is OOM different from RPC?
- REST is also barely covered, but very widely used in industry, so you should know what it is at a high-level. In 30 seconds or less, explain to me what a RESTful service is.
- Why do we care so much about knowing the time? How are physical and logical time different?
- What is UTC, and how is it calculated? How is it different from Unix time?
- What clocks does a typical laptop have? How is each clock used?
- What is the clock synchronisation problem? Why is it a problem? What solutions are there?
- On a whiteboard, draw and explain Cristian's algorithm. What assumptions does it make? Are they fair assumptions to make?
- What is the goal of the Berkeley algorithm? How does it achieve that goal?
- On a whiteboard, draw and explain the communication between a client and a server in the NTP algorithm. How is this an improvement on Cristian's algorithm? How else is NTP more complex than Cristian's algorithm?
- Why can physical time not be used in a distributed system?
- How would you explain a happens-before relation to someone who just finished Part IA of the Computer Science Tripos?

Lecture 12 concepts

- What "rules" are used to define the happens-before relationship shown on the slides?
- By drawing a diagram on a whiteboard, explain how Lamport clocks work. Given the timestamps of two events $\mathbf{L(a)}$ and $\mathbf{L(b)}$, what can we say about the order of **a** and **b**?
- By changing the diagram above, explain how vector clocks work. How can vector clocks be used for ordering? Can you think of a situation in which $\mathbf{V_i} = \mathbf{V_j}$?
- What are the tradeoffs between Lamport clocks and vector clocks? How does each of them scale with the number of processes and the amount of communication?
- What is a consistent cut? (As a point of discussion, is there an error in the definition in the slides?)
- How would you explain process groups to your supervision partner(s)? In what ways can process groups vary?
- What is the difference between "receiving" and "delivering" a message? Why is this distinction useful?
- What are the guarantees provided by FIFO, causal and total ordering? How would you implement each? (Is the "hold-back queue" actually a queue?)

A note on logical clocks Logical clocks are used to attribute timestamps to events so that some ordering can be determined. When using clocks, it's important to keep in mind what events are being "numbered". When clocks are introduced in the lecture slides, the clocks are incremented when a message is sent, received, or when an internal event occurs. In other contexts, such as the implementation of causal ordering of multicast messages in a process group (and the chat application in the Further Java course), the events being tracked are only message "send"s. When implementing an application using a logical clock, make sure you know what you are counting, otherwise you will have bugs that are very difficult to pin down!

3.2 Exercises

3.3 Extensions

TODO: Lamport's relativity argument, gRPC

Supervision 4

Some more distributed stuff

4.1 Bookwork

Lecture 13 concepts Define the following terms:

- Stub

4.2 Exercises

4.3 Extensions

Bibliography

[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.

[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.