Chapter 1

Introduction

In designing an operating system one needs both theoretical insight and horse sense. Without the former, one designs an ad hoc mess; without the latter one designs an elephant in best Carrara marble (white, perfect, and immobile).

Roger Needham and David Hartley [NH69]

Welcome to Computer Systems! This is a brand-new course, so expect surprises, enthusiasm, experimentation, and occasional glitches as we try new ideas out. We’re going to be covering a wide range of topics in this course, including (in no particular order):

- The operating system kernel
- Processes and threads
- Shared-memory interprocess communication
- Message-passing, client-server communication, and remote procedure call
- Scheduling
- Virtual memory and demand paging
- Files and file systems
- Performance analysis of computer systems
- Virtualization and virtual machine monitors
- Consensus and agreement protocols
- Time synchronization
- Logical time
- Blockchains and cryptocurrencies
- Byzantine fault tolerance
- Reliable distributed storage
– and possibly others . . .

**Definition 1.1** (Definition). . . The script for this course is going to have a lot of definitions. They are all true, but note that they are not exclusive: a definition of a term in this refers to the meaning of the term in the context of the course (see Naming, chapter [4]).

**Remarks:**

- On the rare occasions where we adopt a somewhat idiosyncratic definition, we’ll remark on it below the definition.

**Definition 1.2** (Computer Systems). . . **Computer Systems** (sometimes shortened to just “Systems”) is the field of computer science that studies the design, implementation, and behaviour of real, complete systems of hardware and software. **Real** means that anything that can happen in the real world, in a real deployment, is within the scope of study. **Complete** means the ideas we introduce have to make sense in the context of a whole system (such as your phone, or a Google datacenter, or the totality of Facebook’s infrastructure).

**Remarks:**

- This course is being taught by Timothy Roscoe and Roger Wattenhofer, with help from lots of assistants. We come from different background: the more theoretical and the more practical ends of computer systems, and it will probably become clear during the course which of us is which. However, this mixture is deliberate, and hopefully makes the course more interesting and fun (including for us).

- Computer Systems itself has this mixture of theory and hard pragmatism. In some cases, the subject of computer systems sometimes runs considerably ahead of the applicable theory, which comes along later to formalize existing ideas. In others, systems people borrowed or stole theoretical concepts from other fields of computer science. In a few cases, the theory and the implementation (and deployment) actually did evolve together at the same time.

- Systems is thus as much an approach as a subject area. It’s about embracing the whole chaotic mess of reality and avoiding idealizing problems too much, but also using the right theoretical concepts to approximate reality in useful ways. This is why really good Systems people are the most highly prized technical workers (even above machine learning experts) in large tech companies and other organizations [Mic13].

- This course is also a mixture of traditional operating systems, traditional distributed systems, and a bunch of newer concepts (like blockchain). This is also deliberate: in practice, the boundaries between all these are changing right now, and at least to us it makes perfect sense to mix them up and treat them as a single subject.


1.1 Format

We’ll run the lectures a little bit differently to what you may have experienced in, e.g., Systems Programming.

We will distribute a script before each lecture, and you should read this in advance.

For the most part, there won’t be any slides. Instead, we’ll explain key concepts or examples on a whiteboard or via a laptop interactively. You should turn up to the lecture with questions that you would like us to cover. This is sometimes called a “flipped classroom” model: it’s intended to be highly interactive, and mostly driven by the topics you’d like to go into in detail, rather than a pre-prepared set of slides. Since the core material is covered in the script, we have the freedom to talk about what is most useful in the lectures.

You might be wondering if we have objections to you reading email, watching YouTube, using Twitter, WhatsApp or Snapchat, or perusing Instagram or Facebook during lectures.

Remarkably, people who really understand computers, psychology, and education have actually looked into this and done Real Science, randomized trials and all. The results are quite scary [HG03, LG07, LM11, WZG+11, ARWO12, SWC13, M013, CGW17, RUF17]. It turns out that not only does using a laptop, phone, or tablet in classes reduce your academic performance and grade, but there’s a passive effect as well: using a mobile device in a lecture lowers the grades of other people around you, even if they’re not using the device themselves.

We strongly recommend going offline for the 2 × 45 minutes of each lecture.

If you are desperately waiting for that key Instagram photo to show up, it’s now been established scientifically that it’s nicer to your fellow students if you just leave the lecture or not turn up instead.

1.2 Prerequisites

This is a 3rd-year elective course in the ETH Zurich Bachelor. As a result, we assume you’ve completed (and passed) the following courses:

- Design of Digital Circuits
- Parallel Programming
- Systems Programming and Computer Architecture
- Formal Methods and Functional Programming
- Data Modelling and Databases
- Computer Networks

We’ll build on the concepts in these courses. Given the format of the course, and the compact nature of the notes we’ll distribute beforehand, it’s best to be pretty familiar with this prior material.
CHAPTER 1. INTRODUCTION

1.3 Grading

We’ll have an examination in the Prüfungs session early next year. The course grade will be based on this examination. Examinable material will be anything in the script, plus whatever is covered in these interactive sessions.

In addition, an additional quarter-point (0.25) grade can be obtained as a bonus by creating and submitting to us an original exam question including a (correct) solution and grading scheme for one of the topics in this course.

We reserve the right to use your question as inspiration in the actual exam.

Please follow the following guidelines:

• You must submit your question before the end of semester.
• The question must be non-trivial, that is, not a pure knowledge question.
• Both the question and the solution should be as precise as possible, including how points are awarded. Consider up to 20 points for the question.
• You must come up with your own idea. Do not try to cheat – we know our exercise sheets and old exams, and we check material which obviously comes from third-party sources (other books, Wikipedia, ...). Minimal variations of such copied questions are not acceptable either.
• Your submission should be in PDF format.
• Send your exam question to Manuel Eichelberger. Please use “[CompSys Bonus] lastname firstname” as subject line, name your file lastname_firstname.pdf and indicate the topic of your question.

We’ll post the deadline for your question submission about two to three weeks into the course. There will also be a second chance to submit a revised question before the end of the course – further details will follow.

1.4 Mistakes

We expect the script for the whole course to evolve somewhat over the course of the semester – we’ll update it based on the discussions in the class, and also in response to any errors you find and report to us.

Please send typos, etc to the following email address:

compsys-errata@lists.inf.ethz.ch

Bibliography


