Recommended Computer System Learning Materials

Zhenbang You

January 2022

Last Updated: March 5, 2022

The latest version can always be found at https://www.overleaf.com/read/txqjnjxyxqqx

1 Preliminary

- The following materials are list in an order **suitable for learning**.
- The references help you identify the authors.
- CSAPP (3rd Edition) [13] is the starting point of everything below.
- Familiarity with **C/C++** is required.
- For the foundation of **data structures and algorithms**, Introduction to Algorithms (3rd Edition, the 4th Edition is coming in April 2022) [14] is highly recommended.
- The most fundamental components of computer systems are:
  1. Computer Architecture
  2. Operating Systems (OS)
  3. Computer Networks
  4. Compilers
  5. Programming Languages (PL)

  Master them before moving on to other parts.

What if I want more?
Search for advanced courses offered by top universities. Meanwhile, you are likely to find lots of papers there!

2 Computer Architecture

1. Computer Organization and Design: The Hardware-Software Interface
   (5th Edition [29]/RISC-V Edition [28]/ARM Edition [30])
   - The original version is based on MIPS. RISC-V version is recommended; after reading this version, you may proceed on ARM version which is a good textbook for learning ARM.
2. Digital Design and Computer Architecture
   (2nd Edition [18]/RISC-V Edition [19]/ARM Edition [17])
   - You may just read Chap 1-5.
   - The original version is based on MIPS.
3. The RISC-V Reader An Open Architecture Atlas [27]

4. Computer Architecture: A Quantitative Approach (6th Edition) [20]
    - You may leave out appendices the first time you read this book.
    - Difficult as it may be, this book is just "the second book for novices". If you want to have a deep understanding of a specific topic, do go to read official tutorials/documentations such as those of NVIDIA.
5. RISC-V Privileged Architecture (slides)
   https://riscv.org/wp-content/uploads/2018/05/riscv-privileged-BCN.v7-2.pdf
    - A wonderful slide on RISC-V privileged architecture, as well as the core problem "what is the privileged architecture".
    - The video of this lecture can be found at
      https://www.youtube.com/watch?v=fxLXvrLN5jA
    - Most of the textbooks on computer architecture discuss little about **privileged architecture**, resulting in great difficulties understanding the OS kernel. Always keep in mind that **ISA**, i.e., the hardware-software interface, consists of both the unprivileged architecture and the privileged architecture.
6. RISC-V specifications
   https://riscv.org/technical/specifications/
    - Elaborate specifications as they may be, they are indeed excellent textbooks for both neophytes and specialists!
    - Appendix A: RVWMO Explanatory Material of "The RISC-V Instruction Set Manual Volume I: Unprivileged ISA" is a brilliant tutorial for **Memory Consistency**!
7. A New Golden Age for Computer Architecture (a Turing Lecture with full text)
   https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext
    - The famous Turing Lecture by Hennessy and Patterson. What wonderful insights of Masters!

3 Operating Systems (OS)

1. Operating Systems: Three Easy Pieces [7]
   https://pages.cs.wisc.edu/ remzi/OSTEP/
    - Due to the research interest of the authors, this book puts much emphasis on File Systems. You may leave out some chapters of this part the first time you read it.
2. Operating Systems Principles & Practice (2nd Edition)
    - Four volumes:
        - Volume I: Kernels and Processes [5]
        - Volume II: Concurrency [2]
        - Volume III: Memory Management [3]
        - Volume IV: Persistent Storage [4]
3. Operating Systems Concepts (9th Edition) [31]
4. xv6 source and text: https://pdos.csail.mit.edu/6.828/2021/xv6.html
    - Hands-on experiences with a real OS is indispensable, and **xv6** is an excellent starting point!
5. Supplemental textbooks
    (a) Linux Kernel Development (4rd Edition) [34]
    (b) Understanding the Linux Kernel (3rd Edition) [11]

   (c) Linux Device Drivers (3rd Edition) [35]
   (d) Understanding Linux network internals (1st Edition) [9]
   (e) Modern Operating Systems (4th Edition) [44]

6. Prerequisites:
   - Compulsory:
     (a) Computer Architecture: privileged architecture
     (b) Data structures and algorithms
   - PL
     - **Java**: JVM, GC, Thread, Monitor.
     - **Go**: Goroutine, Channel, CSP (Communication Sequential Process), Asynchrony.

4 Computer Networks

1. Computer Networking: A Top Down Approach (8th Edition) [21]
   - I cannot find the reference of the latest version (8th Edition) on Google Scholar, so this citation refers to older version. Note that the authors do not change.
2. Books by W. Richard Stevens (as supplementary materials)
   - UNIX Network Programming
     - Volume 1, Third Edition: The Sockets Networking API [39]
     - Volume 2, Second Edition: Interprocess Communications [32]
   - TCP/IP Illustrated
     - Volume 1: The Protocols (2nd Edition) [15]
     - Volume 2: The Implementation [41]
     - Volume 3: TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols [42]
3. Prerequisites:
   - Compulsory:
     (a) Operating Systems
   - PL
     - **Java**.
     - **Python**: Similar but much simpler socket interface than POSIX.
     - **Go**: RPC.

5 Compilers

Compilers should be associated with **PL** and **Computer Architecture**.

1. Compilers: Principles, Techniques and Tools (2nd Edition) [1]
   - "Dragon Book".
   - Well-known but a little obsolete; still wonderful for new-comers.
2. Modern Compiler Implementation in C (2nd Edition) [6]/Modern Compiler Implementation in Java (2nd Edition)
   - "Tiger Book".
3. Advanced Compiler Design Implementation [26]
   - "Whale Book".

4. Prerequisites:
    - Compulsory:
        (a) TCS (Theoretical Computer Science)
            i. Introduction to the Theory of Computation [38]
        (b) Computer Architecture: ILP (Instruction-Level Parallelism), Memory Hierarchy
        (c) Data structures and algorithms
    - Recommended:
        (a) Computer Architecture: DLP (Data-Level Parallelism), TLP (Thread-Level Parallelism)
        (b) Operating Systems: Thread, Context Switch
    - PL
        - **Java**.

6 Programming Languages (PL)

Abbreviations:

- OOP: Object-Oriented Programming
- FP: Functional Programming

Language List

- **C/C++** (Do learn the latest version of C++, or at least C++17)
    - Tutorials
        1. The C Programming Language (2nd Edition) [33]
            * "**K & R**"
        2. The C++ Programming Language (4th Edition) [43]
    - Documentations
        1. https://cppreference.com/
    - Programming guidelines
        * Scott Meyers "Effective C++" book series
            · Effective C++ [24]
            · Effective Modern C++ [25]
            · Effective STL [23]
        * C++ Core Guidelines:
          https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines
    - Recommended IDE: VSCode on Linux (especially WSL).
    - Compilers (try the latest version!)
        * Currently, **OpenMP** support of **Clang 13** needs to be installed separately, which is not the case for **Clang 12**.
        * **GCC** can be built from source.
        * **Clang** can by downloaded directly from https://releases.llvm.org/download.html
    - Build tool: CMake (also works for CUDA C++).
        * CMake Tutorial: https://cmake.org/cmake/help/latest/guide/tutorial/
- **Java**
  Prerequisite languages: C++.
  Pay attention to the comparison with C++, as well as JVM and JIT.
    - Tutorials
        1. Oracle online tutorial: https://docs.oracle.com/javase/tutorial/

    2. Core Java (10th Edition). Two volumes:
        * Volume I: Fundamentals [16]
        * Volume II: Advanced Features [45]
    3. Java 8 in Action: Lambdas, Streams, and Functional-style Programming [46]
  - Documentations
    1. https://docs.oracle.com/javase/specs/
  - Programming guidelines
    * Effective Java (3rd Edition) [10]
  - Recommended IDE: IntelliJ Idea
  - Build tool (applying to all languages on JVM) : **Maven** or **Gradle**.
    * Maven in 5 Minutes:
      https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html
    * All documentations can be found at:
      https://maven.apache.org/
    * Try **Maven** with **IntelliJ Idea**.

- **Scala**
  Prerequisite languages: Java.
  Notable core features: FP.
  - Tutorials
    1. Tour of Scala: https://docs.scala-lang.org/tour/tour-of-scala.html
    2. Scala Book: https://docs.scala-lang.org/overviews/scala-book/introduction.html
  - Documentations
    * Guides and Overviews: https://docs.scala-lang.org/overviews/index.html
    * All documentations: https://docs.scala-lang.org/
      · **Scala 3** documentations can also be found here!
  - Recommended IDE & build tool: same as **Java**

- **Kotlin**
  Prerequisite languages: Java, Scala.
  Notable core features: null safety, modest FP.
  - Tutorials
    1. Get started with Kotlin: https://kotlinlang.org/docs/getting-started.html
    2. Kotlin Coroutines:
       https://github.com/Kotlin/KEEP/blob/master/proposals/coroutines.md
       * Wonderful article about the design of stackless coroutines!
  - Documentations
    1. https://kotlinlang.org/docs/
  - Recommended IDE & build tool: same as **Java**

- **Go**
  Prerequisite languages: C++, Java.
  Notable core features: concurrent programming (goroutine + channel + asynchrony), modest OOP.
  - Turorials
    1. A Tour of Go: https://go.dev/tour/welcome/1
    2. Effective Go: https://go.dev/doc/effective_go
    3. The Go Blog: https://go.dev/blog/
       * Lots of wonderful articles about the design of Go!
  - Documentations

    1. https://go.dev/doc/
  - Recommended IDE: GoLand
  - Build tool: https://go.dev/blog/using-go-modules

- **Rust**
  Prerequisite languages: C++17, Go, a functional language (e.g., Scala, OCaml, Haskell).
  Notable core features: ownership, borrow checker, various **safety**, modest OOP.
  - Tutorials
    1. The Rust Programming Language: https://doc.rust-lang.org/book/
       * Also a good book about the design of programming languages!
    2. Rust by Example: https://doc.rust-lang.org/stable/rust-by-example/
    3. Asynchronous Programming in Rust: https://rust-lang.github.io/async-book/
  - Documentations
    1. https://doc.rust-lang.org/beta/
  - Recommended IDE: IntelliJ Idea
  - Build Tool: Cargo.
    * Its tutorial can be found in "The Rust Programming Language".

- **Haskell**
  Reckless as it may be to recommend this language, however, it is so elegant...
  - Tutorials
    1. Get Started: https://www.haskell.org/
  - All the books, courses, tutorials, documentations and various kinds of resources can be found at: https://www.haskell.org/documentation/

- Prerequisites (for understanding the design of programming languages):
  - Compulsory:
    1. Computer Architecture
    2. Operating Systems
    3. Compilers
  - Recommended:
    1. Computer Networks

- **Suggestions**:
  - There is no need to master an entire PL in one shot; instead, study part of it every time you need it.
  - PLs develop rapidly. To catch up with the latest development, refer to online documentations/tutorials/blogs besides books.
  - Be sure you have mastered at least one **modern** language in each of the following paradigms:
    * Procedural
    * Object-Oriented
    * Functional

    Note that modern languages like **Go**, **Scala**, **Kotlin** and **Rust** can be greatly different than old ones like **Java** and **Python**. As a special case, although modern **C++** (C++11 and later) is really modern, but there are inevitably a great number of legacies, so C++ can be considered as a mixture.
  - *Concrete examples* always help a lot for understanding *abstract concepts*, and this is also one of the reasons why you should master several languages.
  - Pay attention to the interoperation between a certain language with C (and languages on JVM with Java).
  - Only IDEs I have used will be listed here, so there may be other wonderful IDEs.

7 Parallel Computing

Special Notes:

1. Do read **Computer Architecture: A Quantitative Approach (CAAQA)** before diving into this, and revisit that masterpiece after having some hands-on experience of parallel computing!

2. **Parallel Computing** without **Memory Optimization** is **ridiculous**!

Platforms

- **CUDA**
  - CUDA by Example [36]
    - A little obsolete, but the ideas are still well-presented. If your foundation is good enough, go to the following two documentations directly, and these two are highly recommended.
  - CUDA C++ Programming Guide
    https://docs.nvidia.com/cuda/cuda-c-programming-guide/
  - CUDA C++ Best Practices Guide
    https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/
  - All documentations can be found at
    https://docs.nvidia.com/cuda/

  CUDA is also fantastic for **Asynchronous Programming** and **Heterogeneous Programming**! You can also have a taste of **Compute Hierarchy** with CUDA!

- **CPU intrinsics**
  - **x86 intrinsics**
    https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html
  - **ARM SVE2 intrinsics**
    https://developer.arm.com/documentation/102340/0001/Program-with-SVE2
  - **ARM Neon intrinsics**
    https://developer.arm.com/documentation/102467/0100/Why-Neon-Intrinsics-

- **MPI**
  - MPI Tutorial: https://mpitutorial.com/tutorials/
  - YouTube video
    - MPI Basics: https://www.youtube.com/watch?v=c0C9mQaxsD4
    - MPI Advanced: https://www.youtube.com/watch?v=q9OfXis50Rg

- **OpenMP**
  - Tim Mattson's (Intel) "Introduction to OpenMP" (2013) on YouTube
    - Video:
      https://www.youtube.com/playlist?list=PLLX-Q6B8xqZ8n8bwjGdzBJ25X2utwnoEG
    - Slides:
      https://www.openmp.org/wp-content/uploads/Intro_To_OpenMP_Mattson.pdf

Categories
The essence of **parallel computing** is **the lack of dependencies**.

- ILP (Instruction-Level Parallelism)
  Mathematical model of ILP – DAG (Directed Acyclic Graph):
  - Node: a stage of a instruction.
  - Edge: dependency (data dependency, control dependency, name dependency) between a pair of nodes.

Goal: eliminate dependencies (edges), and exploit the lack of dependencies between nodes.

- DLP (Data-Level Parallelism)
  The essence of DLP programming: **Vectorization**.
  - To get some hands-on experiences with this, you may start with **PyTorch**, since this relieves you from some low-level details like remaining elements and memory hierarchy.
    - Official tutorial: https://pytorch.org/tutorials/
    - Also, **PyTorch** is a convenient tool to exploit GPU for parallel computing.
- TLP (Thread-Level Parallelism)
  The essence of TLP programming: **async** (concurrent control flow), **await** (synchronization).
  - Concurrency mechanisms
    1. Thread: C++, Java
    2. Future/Promise: Scala
    3. Stackful Coroutine: Go
    4. Stackless Coroutine: Kotlin, Rust (async/await)

    Aside:

    1. In essence, every kind of *Concurrency* is an encapsulation of *Asynchrony*. Therefore, *Concurrent Programming* can be seen as a subset of *Asynchronous Programming*.
       - Here is a good summary of *Asynchronous programming techniques* provided in the tutorial of *Kotlin*: https://kotlinlang.org/docs/async-programming.html
    2. The essence of *Asynchronous Programming* is **async/await**, as well as **suspension points**.
       - For *Threads* and *Stackful Coroutines*, every point is a suspension point, while for *Stackless Coroutines*, only a fraction of points can be suspension points and they are declared explicitly.
    3. For implementations, figure out what is "**continuation**" and how it varies in *Processes*, *Threads*, *Stackful Coroutines*, and *Stackless Coroutines*. Also think about the relation between *continuation* and *suspension points*.
  - Synchronization mechanisms
    - Shared memory
      1. Mutex, Condition Variable: most languages.
      2. Monitor: Java. For C++, this can be readily emulated by *RAII*.
      3. Atomic variables/operations: most languages.
      4. Barrier: most languages, as well as CUDA.
      5. Read Write Lock: most languages. This is especially useful in DLP.

      Many widely used mechanisms are not listed here.
    - Message passing
      1. Channel + Select: Go, Kotlin.
      2. Actor Model:
         - Akka (with Java/Scala interface):
           https://doc.akka.io/docs/akka/current/typed/guide/introduction.html
         - Kotlin:
           https://kotlinlang.org/docs/shared-mutable-state-and-concurrency.html#actors
    - High-level encapsulations
      1. Thread-safe collections
         - Java: java.util.concurrent:
           https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/concurrent/package-summary.html

- Prerequisites:
    - Compulsory:
        1. Computer Architecture
        2. Compilers
    - Recommended:
        1. Operating Systems

8 Database

1. Database System Concepts (Seventh Edition) [37]
2. CMU 15-445/645 Intro to Database Systems
   https://15445.courses.cs.cmu.edu/fall2019/schedule.html
3. CMU 15-721 Advanced Database Systems
   https://15721.courses.cs.cmu.edu/spring2020/schedule.html
4. Prerequisites:
    - Compulsory
        (a) Data structures and algorithms

9 Distributed Computing/Distributed Systems

1. MIT 6.824: Distributed Systems: https://pdos.csail.mit.edu/6.824/schedule.html
2. Prerequisites:
    - Compulsory:
        (a) Operating Systems
        (b) Computer Networks
    - Recommended:
        (a) Computer Architecture
        (b) Programming Languages

10 Cloud Computing

1. CMU 15-719 Advanced Cloud Computing:
   https://www.cs.cmu.edu/ 15719/old/spring2019/syllabus.html
2. Prerequisites:
    - Compulsory:
        (a) Computer Architecture
        (b) Operating Systems
        (c) Computer Networks
        (d) Distributed Systems
    - Recommended:
        (a) Parallel Computing
        (b) Database

11 Recommended ISA

All of the following ISA can be learned by CSAPP & "Computer Organization and Design: The Hardware/Software Interface". However, the following materials help you go further.

- **x86**
  - Intel® 64 and IA-32 Architectures Software Developer Manuals: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

  Make sure you are able to read both the "Intel" format and the "AT&T" format.

- **RISC-V**
  - RISC-V Specifications: https://riscv.org/technical/specifications/
    - Wonderful textbooks on computer architecture!
- **ARM**
  - ARM® CPU Architecture Key Documents: https://developer.arm.com/architectures/cpu-architecture
  - Arm® Architecture Reference Manual Supplement Armv9, for Armv9-A architecture profile: https://developer.arm.com/documentation/ddi0608/latest
- **PTX**
  - PTX ISA :: CUDA Toolkit Documentation: https://docs.nvidia.com/cuda/parallel-thread-execution/
- **Java Bytecode**
  - Java Language and Virtual Machine Specifications: https://docs.oracle.com/javase/specs/

12 Linux Programming

1. Linux man pages
   - "man" command in Linux shell, like "man fork" or "man 2 fork" where "2" specifies the volume.
   - Linux man pages online: https://man7.org/linux/man-pages/
2. Advanced programming in the UNIX environment [40]

13 Linking and Loading

1. 程序员的自我修养: 链接, 装载与库 [8]
2. Linkers and Loaders [22]

14 Software Engineering

- The Mythical Man-Month: Essays on Software Engineering [12]

15 Tools

1. Git
   - Git Magic: http://www-cs-students.stanford.edu/ blynn/gitmagic/book.pdf
   - Official documentations: https://git-scm.com/doc
2. UNIX Makefile
   - Makefile Tutorial By Example: https://makefiletutorial.com/
3. Docker
   - Official tutorials and documentations: https://docs.docker.com/
4. Shell Script
   - Shell Scripting Tutorial: https://www.shellscript.sh/
   - The following two tutorials are basically the same and are written by the same author, with the former being more newbie-friendly:
     - Bash Scripting Tutorial for Beginners: https://linuxconfig.org/bash-scripting-tutorial-for-beginners
     - Bash Scripting Tutorial: https://linuxconfig.org/bash-scripting-tutorial
   - GNU Bash manual: https://www.gnu.org/software/bash/manual/

16 Coding Style

- Google Style Guides: https://google.github.io/styleguide/
- Clang-Format Style Options: https://clang.llvm.org/docs/ClangFormatStyleOptions.html
  - *VS Code* can automatically format your code with a ".clang-format" file without installing anything!

17 Miscellaneous

1. Intel® Product Specifications: https://ark.intel.com
2. GCC online documentation: https://gcc.gnu.org/onlinedocs/
   - Optimize Options: https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html
   - Option Summary: https://gcc.gnu.org/onlinedocs/gcc/Option-Summary.html
   - Instrumentation Summary (including sanitizer and profiler): https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html
3. GNU Manuals Online: https://www.gnu.org/manual/

References

[1] A. Aho, M. Lam, R. Sethi, J. Ullman, K. Cooper, L. Torczon, and S. Muchnick. Compilers: Principles, techniques and tools. 2007.

[2] T. Anderson and M. Dahlin. Operating systems principles & practice volume ii: Concurrency.

[3] T. Anderson and M. Dahlin. Operating systems principles & practice volume iii: Memory management.

[4] T. Anderson and M. Dahlin. Operating systems principles & practice volume iv: Persistent storage.

[5] T. Anderson and M. Dahlin. *Operating Systems Principles & Practice Volume I: Kernels and Processes.* Recursive books, 2014.

[6] A. W. Appel. *Modern compiler implementation in C.* Cambridge university press, 2004.

[7] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau. *Operating systems: Three easy pieces.* Arpaci-Dusseau Books LLC, 2018.

[8] 俞甲子, 石凡. 程序员的自我修养*:* 链接*,* 装载与库. 电子工业出版社, 2009.

[9] C. Benvenuti. *Understanding Linux network internals.* " O'Reilly Media, Inc.", 2006.

[10] J. Bloch. *Effective java (the java series).* Prentice Hall PTR, 2008.

[11] D. P. Bovet and M. Cesati. *Understanding the Linux Kernel: from I/O ports to process management.* " O'Reilly Media, Inc.", 2005.

[12] F. P. Brooks Jr. *The mythical man-month: essays on software engineering.* Pearson Education, 1995.

[13] R. E. Bryant and D. R. O'Hallaron. Computer systems: A programmer's perspective, 2015.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms.* MIT press, 2009.

[15] K. R. Fall and W. R. Stevens. *TCP/IP illustrated, volume 1: The protocols.* addison-Wesley, 2011.

[16] I. Gvero. Core java volume i: Fundamentals, by cay s. horstmann and gary cornell. *ACM Sigsoft Software Engineering Notes*, 38(3):33–33, 2013.

[17] S. Harris and D. Harris. Digital design and computer architecture: Arm edition, 2015.

[18] S. L. Harris and D. Harris. *Digital design and computer architecture.* Morgan Kaufmann, 2015.

[19] S. L. Harris and D. Harris. *Digital Design and Computer Architecture, RISC-V Edition.* Morgan Kaufmann, 2021.

[20] J. L. Hennessy and D. A. Patterson. Computer architecture: a quantitative approach, 2018.

[21] J. Kurose, K. Ross, and J. Addison-Wesley. Computer networking: A top down approach.

[22] J. Levine. Linkers and loaders,. 1999.

[23] S. Meyers. *Effective STL: 50 specific ways to improve your use of the standard template library.* Pearson Education, 2001.

[24] S. Meyers. *Effective C++: 55 specific ways to improve your programs and designs.* Pearson Education, 2005.

[25] S. Meyers. *Effective modern C++: 42 specific ways to improve your use of C++ 11 and C++ 14.* " O'Reilly Media, Inc.", 2014.

[26] S. Muchnick et al. *Advanced compiler design implementation.* Morgan kaufmann, 1997.

[27] D. Patterson and A. Waterman. *The RISC-V Reader: an open architecture Atlas.* Strawberry Canyon, 2017.

[28] D. A. Patterson and J. L. Hennessy. Computer organization and design risc-v edition: The hardware software interface (the morgan kaufmann).

[29] D. A. Patterson and J. L. Hennessy. Computer organization and design: The hardware/software interface, 2013.

[30] D. A. Patterson and J. L. Hennessy. *Computer organization and design ARM edition: the hardware software interface.* Morgan kaufmann, 2016.

[31] J. L. Peterson and A. Silberschatz. *Operating system concepts.* Addison-Wesley Longman Publishing Co., Inc., 1985.

[32] W. Richard. Unix network programming, volume 2: Interprocess communications, 1999.

[33] D. M. Ritchie, B. W. Kernighan, and M. E. Lesk. *The C programming language.* Prentice Hall Englewood Cliffs, 1988.

[34] L. Robert. *Linux kernel development.* Pearson Education India, 2018.

[35] A. Rubini and J. Corbet. *Linux device drivers.* " O'Reilly Media, Inc.", 2001.

[36] J. Sanders and E. Kandrot. *CUDA by example: an introduction to general-purpose GPU programming.* Addison-Wesley Professional, 2010.

[37] A. Silberschatz, H. F. Korth, S. Sudarshan, et al. *Database system concepts*, volume 5.

[38] M. Sipser. Introduction to the theory of computation. *ACM Sigact News*, 27(1):27–29, 1996.

[39] W. R. Stevens, B. Fenner, and A. M. Rudoff. *UNIX Network Programming Volume 1.* SMIT-SMU, 2018.

[40] W. R. Stevens, S. A. Rago, and D. M. Ritchie. *Advanced programming in the UNIX environment*, volume 4. Addison-Wesley New York., 1992.

[41] W. R. Stevens and G. R. Wright. *TCP/IP Illustrated: volume 2.* Addison-wesley, 1996.

[42] W. R. Stevens and G. R. Wright. *TCP for transactions, HTTP, NNTP, and the UNIX domain protocols.* Addison-Wesley, 2000.

[43] B. Stroustrup. The c++ programming language (hardcover), 2013.

[44] A. Tanenbaum and H. Bos. Modern operating systems. 2015.

[45] A. B. Tarımcı. Core java® volume ii: advanced features by cay s. horstmann and gary cornell. *ACM SIGSOFT Software Engineering Notes*, 39(3):24–25, 2014.

[46] R.-G. Urma, M. Fusco, and A. Mycroft. *Java 8 in action.* Manning publications, 2014.