Preface

To the best of our knowledge this textbook is unique in its scope and approach. It provides a broad and in-depth introduction to the main principles and abstractions for engineering computer systems, be it an operating system, a client/service application, a database system, a secure Web site, or a fault-tolerant disk cluster. These principles and abstractions are timeless and are of value to any student or professional reader, whether or not specializing in computer systems. The principles and abstractions derive from insights that have proven to work over generations of computer systems, the authors' own experience with building computer systems, and teaching about them for several decades.

The book teaches a broad set of principles and abstractions, yet it explores them in depth. It captures the core of a concept using pseudocode so that readers can test their understanding of a concrete instance of the concept. Using pseudocode, the book carefully documents the essence of client/service computing, remote procedure calls, files, threads, address spaces, best-effort networks, atomicity, authenticated messages, and so on. This approach continues in the problem sets, where readers can explore the design of a wide range of systems by studying their pseudocode.

This printed textbook is Part I of a two-part publication, containing just the first six chapters. Part II, consisting of Chapters 7–11 and additional supporting materials, is posted on-line as an open educational resource. For details of how and where to find Part II on-line, see "Where to find Part II and other on-line materials" on page xxix.

WHY THIS TEXTBOOK?

Many fundamental ideas concerning computer systems, such as design principles, modularity, naming, abstraction, concurrency, communications, fault tolerance, and atomicity, are common to several of the upper-division electives of the Computer Science and Engineering (CSE) curriculum. A typical CSE curriculum starts with two beginning courses, one on programming and one on hardware. It then branches out, with one of the main branches consisting of systems-oriented electives that carry labels such as

- Operating systems
- Networks
- Database systems
- Distributed systems
- Programming languages
- Software engineering
- Security
- Fault tolerance
- Concurrency
- Architecture

The primary problem with this list is that it has grown over the last three decades, and most students interested in systems do not have the time to take all or even

several of those courses. The typical response is for the CSE curriculum to require either "choose three" or "take Operating Systems plus two more". The result is that most students end up with no background at all in the remaining topics. In addition, none of the electives can assume that any of the other electives have preceded it, so common material ends up being repeated several times. Finally, students who are not planning to specialize in systems but want to have some background have little choice but to go into depth in one or two specialized areas.

This book cuts across all of these courses, identifying common mechanisms and design principles, and explaining in depth a carefully chosen set of cross-cutting ideas. This approach provides an opportunity to teach a core undergraduate course that is accessible to all Computer Science and Engineering students, whether or not they intend to specialize in systems. On the one hand, students who will just be users of systems will take away a solid grounding, while on the other hand those who plan to plan to make a career out of designing systems can learn more advanced material more effectively through electives that have the same names as in the list above but with more depth and less duplication. Both groups will acquire a broad base of what the authors hope are timeless concepts rather than current and possibly short-lived techniques. We have found this course structure to be effective at M.I.T.

The book achieves its extensive range of coverage without sacrificing intellectual depth by focusing on underlying and timeless concepts that will serve the student over an entire professional career, rather than providing detailed expositions of the mechanics of operation of current systems that will soon become obsolete. A pervading philosophy of the book is that pedagogy takes precedence over job training. For example, the text does not teach a particular operating system or rely on a single computer architecture. Instead it introduces models that exhibit the main ideas found in contemporary systems, but in forms less cluttered with evolutionary vestiges. The pedagogical model is that for someone who understands the concepts, the detailed mechanics of operation of any particular system can easily and quickly be acquired from other books or from the documentation of the system itself. At the same time, the text makes concepts concrete using pseudocode fragments, so that students have something specific to examine and to test their understanding of the concepts.

FOR WHOM IS THIS BOOK INTENDED?

The authors intend the book for students and professionals who will

- Design computer systems.
- Supervise the design of computer systems.
- Engineer applications of computer systems to information management.
- Direct the integration of computer systems within an organization.
- Evaluate performance of computer systems.

- Keep computer systems technologically up to date.
- Go on to study individual topics such as networks, security, or transaction management in greater depth.
- Work in other areas of computer science and engineering, but would like to have a basic understanding of the main ideas about computer systems.

Level: This book provides an *introduction* to computer systems. It does not attempt to explore every issue or get to the bottom of those issues it does explore. Instead, its goal is for the reader to acquire insight into the complexities of the systems he or she will be depending on for the remainder of a career as well as the concepts needed to interact with system designers. It provides a solid foundation about the mechanisms that underlie operating systems, database systems, data networks, computer security, distributed systems, fault tolerant computing, and concurrency. By the end of the book, the reader should in principle be able to follow the detailed engineering of many aspects of computer systems, be prepared to read and understand current professional literature about systems, and know what questions to ask and where to find the answers.

The book can be used in several ways. It can be the basis for a one-semester, two-quarter, or three-quarter series on computer systems. Or one or two selected chapters can be an introduction of a traditional undergraduate elective or a graduate course in operating systems, networks, database systems, distributed systems, security, fault tolerance, or concurrency. Used in this way, a single book can serve a student several times.Another possibility is that the text can be the basis for a graduate course in systems in which students review those areas they learned as undergraduates and fill in the areas they missed.

Prerequisites: The book carefully limits its prerequisites. When used as a textbook, it is intended for juniors and seniors who have taken introductory courses on software design and on computer hardware organization, but it does not require any more advanced computer science or engineering background. It defines new terms as it goes, and it avoids jargon, but nevertheless it also assumes that the reader has acquired some practical experience with computer systems from a summer job or two or from laboratory work in the prerequisite courses. It does not require that the reader be fluent in any particular computer language, but rather be able to transfer general knowledge about computer programming languages to the varied and sometimes *ad hoc* programming language used in pseudocode examples.

Other Readers: Professionals should also find this book useful. It provides a modern and forward-looking perspective on computer system design, based on enforcing modularity. This perspective recognizes that over the last decade or two, the primary design challenge has become that of keeping complexity under control rather than fighting resource constraints. In addition, professionals who in college took only a subset of the classes in computer systems or an operating systems class that focused on resource management will find that this text refreshes them with a modern and broader perspective.

HOW TO USE THIS BOOK

Exercises and Problem Sets: Each chapter of the textbook ends with a few short-answer exercises intended to test understanding of some of the concepts in that chapter. At the end of the book is a much longer collection of problem sets that challenge the reader to apply the concepts to new and different problems similar to those that might be encountered in the real world. In most cases, the problem sets require concepts from several chapters. Each problem set identifies the chapter or chapters on which it is focused, but later problem sets typically draw concepts from all earlier chapters. Answers to the exercises and solutions for the problem sets are available from the publisher in a separate book for instructors.

The exercises and problem sets can be used in several ways:

- *As tools for learning.* In this mode, the answers and solutions are available to the student, who is encouraged to work the exercises and problem sets and come up with answers and solutions on his or her own. By comparing those answers and solutions with the expected ones, the student receives immediate feedback that can correct misconceptions and can raise questions about ambiguities or misunderstandings. One technique to encourage study of the exercises and solutions is to announce that questions identical to or based on one or more of the problem sets will appear on a forthcoming examination.
- *As homework or examination material.* In this mode, exercises and problem sets are assigned as homework, and the student hands in answers that are evaluated and handed back together with copies of the answers and solutions.
- *As the source of ideas for new exercises and problem sets.*

Case Studies and Readings: To complement the text, the reader should supplement it with readings from the professional technical literature and with case studies. Following the last chapter is a selected bibliography of books and papers that offer wisdom, system design principles, and case studies surrounding the study of systems. By varying the pace of introduction and the number and intellectual depth of the readings, the text can be the basis for a one-term undergraduate core course, a two-term or three-quarter undergraduate sequence, or a graduate-level introduction to computer systems.

Projects: Our experience is that for a course that touches many aspects of computer systems, a combination of several lightweight hands-on assignments (for example, experimentally determine the size of the caches of a personal computer or trace asymmetrical routes through the Internet), plus one or two larger paper projects that involve having a small team do a high-level system design (for example, in a 10-page report design a reliable digital storage system for the Library of Congress), make an excellent adjunct to the text. On the other hand, substantial programming projects that require learning the insides of a particular system take so much homework time that when combined with a broad concepts course they create an overload. Courses with programming projects do work well in follow-on specialized electives, for example,

on operating systems, networks, databases, or distributed systems. For this reason, at M.I.T. we assign programming projects in several advanced electives but not in the systems course that is based on this textbook.

Support: Several on-line resources provide support for this textbook. The first of these resources is a set of course syllabi, reading lists, problem sets, videotaped lectures, quizzes, and quiz solutions. A second resource is a Web site of the publisher that is devoted to collecting resources and links of interest to students, professional readers, and instructors. A third resource is a mostly open Web site for communication between instructors of M.I.T. course 6.033, which uses this text, and their current students. It contains announcements, readings, and problem assignments for the current or most recent teaching term. In addition to current class communications, this Web site also holds an archive going back to 1995 that includes

- Design project assignments
- Hands-on assignments
- Examinations and solutions (These overlap the exercises and problem sets of the textbook but they also include exam questions and answers about the outside readings.)
- Lecture and recitation schedules
- Reading assignments and essay questions about the readings

Instructions for finding all of these on-line resources are in the section "Where to find Part II and other on-line materials".

HOW THE BOOK IS ORGANIZED

Because not every instructor may want to use every chapter of the textbook, it is presented in what, at least at the time of publication, may be viewed as a somewhat novel way: The first six chapters, which the authors consider to be the core materials for almost any course about computer systems, appear in this printed book. The remaining five chapters are available on-line from the authors and M.I.T. under a Creative Commons license that permits free, unlimited non-commercial use and remixing. The on-line chapters are also available on the Web site of the publisher of this textbook. There are many forward cross-references from the core chapters to the later chapters. Those cross-references are identified as in this example: "This topic is explored in more detail in Section 7.4.1 [on-line]".

Themes: Three themes run through this textbook. First, as suggested by its title, the text emphasizes the importance of systematic design principles. As each design principle is encountered for the first time, it appears in display form with a label and a mnemonic catchphrase. When that design principle is encountered again, it is identified by its name and highlighted with a distinctive print format as a reminder of its wide applicability. The design principles are also summarized on the inside front cover of this book. A second theme is that the text is network-centered, introducing communication and networks in the beginning chapters and building on that base in the

succeeding chapters. A third theme is that it is security-centered, introducing enforced modularity in early chapters and adding successively more stringent enforcement methods in succeeding chapters. The security chapter ends the book, not because it is an afterthought, but because it is the logical culmination of a development based on enforced modularity. Traditional texts and courses teach about threads and virtual memory primarily as a resource allocation problem. This text approaches those topics primarily as ways of providing and enforcing modularity, while at the same time taking advantage of multiple processors and large address spaces.

Terminology and examples: The text identifies and develops concepts and design principles that are common to several specialty fields: software engineering, programming languages, operating systems, distributed systems, networking, database systems, and machine architecture. Experienced computer professionals are likely to find that at least some parts of this text use examples, ways of thinking, and terminology that seem unusual, even foreign to their traditional ways of explaining their favorite topics. But workers from these different specialties will compile different lists of what seems foreign. The reason is that, historically, workers within these specialties have identified what turn out to be identical underlying concepts and design principles, but they have used different language, different perspectives, different examples, and different terminology to explain them.

This text chooses, for each concept, what the authors believe is the most pedagogically effective explanation and examples, adopting widely used terminology wherever possible. In cases where different specialty areas use conflicting terms, glossaries and sidebars provide bridges and discuss terminology collisions. The result is a novel, but in our experience effective, way of teaching new generations of Computer Science and Engineering students what is fundamental about computer system design. With this starting point, when the student reads an advanced book or paper or takes an advanced elective course, he or she should be able to immediately recognize familiar concepts cloaked in the terminology of the specialty. A scientist would explain this approach by saying "The physics is independent of the units of measurement." A similar principle applies to the engineering of computer systems: "The concepts are independent of the terminology".

Citations: The text does not use citations as a scholarly method of identifying the originators of each concept or idea; if it did, the book would be twice as thick. Instead the citations that do appear are pointers to related materials that the authors think are worth knowing about. There is one exception: certain sections are devoted to war stories, which may have been distorted by generations of retelling. These stories include citations intended to identify the known sources of each story, so that the reader has a way to assess their validity.

CHAPTER CONTENT

Relation to ACM/IEEE recommendations: The ACM/IEEE Computer Science and Engineering recommendations of 2001 and 2004 describe two layers. The first layer is a set of modules that constitute an appropriate CSE education. The second layer consists of

several suggested packagings of those modules into term-sized courses. This book may be best viewed as a distinct, modern packaging of the modules, somewhat resembling the ACM/IEEE Computer Science 2001 recommendation CS226c, Operating Systems and Networking (compressed), but with the additional scope of naming, fault tolerance, atomicity, and both system and network security. It also somewhat resembles the ACM/IEEE Computer engineering 2004 recommendation CPE_D203, Operating Systems and Net-Centric computing, with the additional scope of naming, fault tolerance, atomicity, and cryptographic protocols.

Chapter 1: Systems. This chapter lays out the general philosophy of the authors on ways to think about systems, with examples illustrating how computer systems are similar to, and different from, other engineering systems. It also introduces three main ideas: (1) the importance of systematic design principles, (2) the role of modularity in controlling complexity of large systems, and (3) methods of enforcing modularity.

Chapter 2: Elements of Computer System Organization. This chapter introduces three key methods of achieving and taking advantage of modularity in computer systems: abstraction, naming, and layers. The discussion of abstraction lightly reviews computer architecture from a systems perspective, creating a platform on which the rest of the book builds, but without simple repetition of material that readers probably already know. The naming model is fundamental to how computer systems are modularized, yet it is a subject usually left to advanced texts on programming language design. The chapter ends with a case study of the way in which naming, layering, and abstraction are applied in the UNIX file system. Because the case study develops as a series of pseudocode fragments, it provides both a concrete example of the concepts of the chapter and a basis for reference in later chapters.

Chapter 3: Design of Naming Schemes. This chapter continues the discussion of naming in system design by introducing pragmatic engineering considerations and reinforcing the role that names play in organizing a system as a collection of modules. The chapter ends with a case study and a collection of war stories. The case study uses the Uniform Resource Locator (URL) of the World Wide Web to show an example of nearly every naming scheme design consideration. The war stories are examples of failures of real-world naming systems, illustrating what goes wrong when a designer ignores or is unaware of design considerations.

Chapter 4: Enforcing Modularity with Clients and Services. The first three chapters developed the importance of modularity in system design. This chapter begins the theme of enforcing that modularity by introducing the client/service model, which is a powerful and widely used method of allowing modules to interact without interfering with one another. This chapter also begins the network-centric perspective that pervades the rest of the book. At this point, we view the network only as an abstract communication system that provides a strong boundary between client and service. Two case studies again help nail down the concepts. The first is of the Internet Domain Name System (DNS), which provides a concrete illustration of the concepts of both Chapters 3 and 4. The second case study, that of the Sun Network

File System (NFS), builds on the case study of the UNIX file system in Chapter 2 and illustrates the impact of remote service on the semantics of application programming interfaces.

Chapter 5: Enforcing Modularity with Virtualization. This chapter switches attention to enforcing modularity within a computer by introducing virtual memory and virtual processors, commonly called threads. For both memory and threads, the discussion begins with an environment that has unlimited resources. The virtual memory discussion starts with an assumption of many threads operating in an unlimited address space and then adds mechanisms to prevent threads from unintentionally interfering with one another's data—addressing domains and the user/kernel mode distinction. Finally, the text examines limited address spaces, which require introducing virtual addresses and address translation, along with the inter-address-space communication problems that they create.

Similarly, the discussion of threads starts with the assumption that there are as many processors as threads, and concentrates on coordinating their concurrent activities. It then moves to the case where a limited number of real processors are available, so thread management is also required. The discussion of thread coordination uses eventcounts and sequencers, a set of mechanisms that are not often seen in practice but that fit the examples in a natural way. Traditionally, thread coordination is among the hardest concepts for the first-time reader to absorb. Problem sets then invite readers to test their understanding of the principles with semaphores and condition variables.

The chapter explains the concepts of virtual memory and threads both in words and in pseudocode that help clarify how the abstract ideas actually work, using familiar real-world problems. In addition, the discussion of thread coordination is viewed as the first step in understanding atomicity, which is the subject of Chapter 9 [on-line].

The chapter ends with a case study and an application. The case study explores how enforced modularity has evolved over the years in the Intel x86 processor family. The application is the use of virtualization to create virtual machines. The overall perspective of this chapter is to focus on enforcing modularity rather than on resource management, taking maximum advantage of contemporary hardware technology, in which processor chips are multicore, address spaces are 64 bits wide, and the amount of directly addressable memory is measured in gigabytes.

Chapter 6: Performance. This chapter focuses on intrinsic performance bottlenecks that are found in common across many kinds of computer systems, including operating systems, databases, networks, and large applications. It explores two of the traditional topics of operating systems books—resource scheduling and multilevel memory management—but in a context that emphasizes the importance of maintaining perspective on performance optimization in a world where each decade brings a thousand-fold improvement in some underlying hardware capabilities while barely affecting other performance metrics. As an indication of this different perspective, scheduling is illustrated with a disk arm scheduling problem rather than the usual time-sharing processor scheduler.

Chapters 7 through 11 are on-line, in Part II of the book. Their contents are described in the section titled "About Part II" on page 369, and information on how to locate them can be found in "Where to find Part II and other on-line materials".

Suggestions for Further Reading. A selected reading list includes commentary on why each selection is worth reading. The selection emphasis is on books and papers that provide insight rather than materials that provide details.

Problem Sets. The authors use examinations not just as a method of assessment, but also as a method of teaching. Therefore, some of the exercises at the end of each chapter and the problem sets at the end of the book (all of which are derived from examinations administered over the years while teaching the material of this textbook) go well beyond simple practice with the concepts. In working the problems out, the student explores alternative designs, learns about variations of techniques seen in the textbook, and becomes familiar with interesting, sometimes exotic, ideas and methods that have been proposed for or used in real system designs. The problem sets generally have significant setup, and they ask questions that require applying concepts creatively, with the goal of understanding the trade-offs that arise in using these methods.

Glossary. As mentioned earlier, the literature of computer systems derives from several different specialties that have each developed their own dictionaries of system-related concepts. This textbook adopts a uniform terminology throughout, and the Glossary offers definitions of each significant term of art, indicates which chapter introduces the term, and in many cases explains different terms used by different workers in different specialties. For completeness and for easy reference, the Glossary in this book includes terms introduced in Part II.

Index of Concepts. The index tells where to find the defining discussion of every concept. In addition, it lists every application of each of the design principles. (For completeness, it includes concepts that are introduced in Part II, listing just the chapter number.)