Operating Systems and Middleware: Supporting Controlled Interaction

Max Hailperin
Gustavus Adolphus College

Revised Edition 1.1
July 27, 2011

To my family

Contents

Preface

Suppose you sit down at your computer to check your email. One of the messages includes an attached document, which you are to edit. You click the attachment, and it opens up in another window. After you start editing the document, you realize you need to leave for a trip. You save the document in its partially edited state and shut down the computer to save energy while you are gone. Upon returning, you boot the computer back up, open the document, and continue editing.

This scenario illustrates that computations interact. In fact, it demonstrates at least three kinds of interactions between computations. In each case, one computation provides data to another. First, your email program retrieves new mail from the server, using the Internet to bridge space. Second, your email program provides the attachment to the word processor, using the operating system's services to couple the two application programs. Third, the invocation of the word processor that is running before your trip provides the partially edited document to the invocation running after your return, using disk storage to bridge time.

In this book, you will learn about all three kinds of interaction. In all three cases, interesting software techniques are needed in order to bring the computations into contact, yet keep them sufficiently at arms length that they don't compromise each other's reliability. The exciting challenge, then, is supporting controlled interaction. This includes support for computations that share a single computer and interact with one another, as your email and word processing programs do. It also includes support for data storage and network communication. This book describes how all these kinds of support are provided both by operating systems and by additional software layered on top of operating systems, which is known as middleware.

Audience

If you are an upper-level computer science student who wants to understand how contemporary operating systems and middleware products work and why they work that way, this book is for you. In this book, you will find many forms of balance. The high-level application programmer's view, focused on the services that system software provides, is balanced with a lower-level perspective, focused on the mechanisms used to provide those services. Timeless concepts are balanced with concrete examples of how those concepts are embodied in a range of currently popular systems. Programming is balanced with other intellectual activities, such as the scientific measurement of system performance and the strategic consideration of system security in its human and business context. Even the programming languages used for examples are balanced, with some examples in Java and others in C or C++. (Only limited portions of these languages are used, however, so that the examples can serve as learning opportunities, not stumbling blocks.)

Systems Used as Examples

Most of the examples throughout the book are drawn from the two dominant families of operating systems: Microsoft Windows and the UNIX family, including especially Linux and Mac OS X. Using this range of systems promotes the students' flexibility. It also allows a more comprehensive array of concepts to be concretely illustrated, as the systems embody fundamentally different approaches to some problems, such as the scheduling of processors' time and the tracking of files' disk space.

Most of the examples are drawn from the stable core portions of the operating systems and, as such, are equally applicable to a range of specific versions. Whenever Microsoft Windows is mentioned without further specification, the material should apply to Windows NT, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows 2008, and Windows 7. All Linux examples are from version 2.6, though much of the material applies to other versions as well. Wherever actual Linux source code is shown (or whenever fine details matter for other reasons), the specific subversion of 2.6 is mentioned in the end-of-chapter notes. Most of the Mac OS X examples originated with version 10.4, also known as Tiger, but should be applicable to other versions.

Where the book discusses the protection of each process's memory, one

additional operating system is brought into the mix of examples, in order to illustrate a more comprehensive range of alternative designs. The IBM iSeries, formerly known as the AS/400, embodies an interesting approach to protection that might see wider application within current students' lifetimes. Rather than giving each process its own address space (as Linux, Windows, and Mac OS X do), the iSeries allows all processes to share a single address space and to hold varying access permissions to individual objects within that space.

Several middleware systems are used for examples as well. The Oracle database system is used to illustrate deadlock detection and recovery as well as the use of atomic transactions. Messaging systems appear both as another application of atomic transactions and as an important form of communication middleware, supporting distributed applications. The specific messaging examples are drawn from the IBM WebSphere MQ system (formerly MQSeries) and the Java Message Service (JMS) interface, which is part of Java 2 Enterprise Edition (J2EE). The other communication middleware examples are Java RMI (Remote Method Invocation) and web services. Web services are explained in platform-neutral terms using the SOAP and WSDL standards, as well as through a J2EE interface, JAX-RPC (Java API for XML-Based RPC).

Organization of the Text

Chapter 1 provides an overview of the text as a whole, explaining what an operating system is, what middleware is, and what sorts of support these systems provide for controlled interaction.

The next nine chapters work through the varieties of controlled interaction that are exemplified by the scenario at the beginning of the preface: interaction between concurrent computations on the same system (as between your email program and your word processor), interaction across time (as between your word processor before your trip and your word processor after your trip), and interaction across space (as between your email program and your service provider's email server).

The first of these three topics is controlled interaction between computations operating at one time on a particular computer. Before such interaction can make sense, you need to understand how it is that a single computer can be running more than one program, such as an email program in one window and a word processing program in another. Therefore, Chapter 2 explains the fundamental mechanism for dividing a computer's attention

between concurrent computations, known as threads. Chapter 3 continues with the related topic of scheduling. That is, if the computer is dividing its time between computations, it needs to decide which ones to work on at any moment.

With concurrent computations explained, Chapter 4 introduces controlled interactions between them by explaining synchronization, which is control over the threads' relative timing. For example, this chapter explains how, when your email program sends a document to your word processor, the word processor can be constrained to read the document only after the email program writes it. One particularly important form of synchronization, atomic transactions, is the topic of Chapter 5. Atomic transactions are groups of operations that take place as an indivisible unit; they are most commonly supported by middleware, though they are also playing an increasing role in operating systems.

Other than synchronization, the main way that operating systems control the interaction between computations is by controlling their access to memory. Chapter 6 explains how this is achieved using the technique known as virtual memory. That chapter also explains the many other objectives this same technique can serve. Virtual memory serves as the foundation for Chapter 7's topic, which is processes. A process is the fundamental unit of computation for protected access, just as a thread is the fundamental unit of computation for concurrency. A process is a group of threads that share a protection environment; in particular, they share the same access to virtual memory.

The next three chapters move outside the limitations of a single computer operating in a single session. First, consider the document stored before a trip and available again after it. Chapter 8 explains persistent storage mechanisms, focusing particularly on the file storage that operating systems provide. Second, consider the interaction between your email program and your service provider's email server. Chapter 9 provides an overview of networking, including the services that operating systems make available to programs such as the email client and server. Chapter 10 extends this discussion into the more sophisticated forms of support provided by communication middleware, such as messaging systems, RMI, and web services.

Finally, Chapter 11 focuses on security. Because security is a pervasive issue, the preceding ten chapters all provide some information on it as well. Specifically, the final section of each chapter points out ways in which security relates to that chapter's particular topic. However, even with that coverage distributed throughout the book, a chapter specifically on security

is needed, primarily to elevate it out of technical particulars and talk about general principles and the human and organizational context surrounding the computer technology.

The best way to use these chapters is in consecutive order. However, Chapter 5 can be omitted with only minor harm to Chapters 8 and 10, and Chapter 9 can be omitted if students are already sufficiently familiar with networking.

Relationship to Computer Science Curriculum 2008

Operating systems are traditionally the subject of a course required for all computer science majors. In recent years, however, there has been increasing interest in the idea that upper-level courses should be centered less around particular artifacts, such as operating systems, and more around cross-cutting concepts. In particular, the *Computing Curricula 2001* (CC2001) and its interim revision, *Computer Science Curriculum 2008* (CS2008), provide encouragement for this approach, at least as one option. Most colleges and universities still retain a relatively traditional operating systems course, however. Therefore, this book steers a middle course, moving in the direction of the cross-cutting concerns while retaining enough familiarity to be broadly adoptable.

The following table indicates the placement within this text of knowledge units from CS2008's computer science body of knowledge. Those knowledge units designated as core units within CS2008 are listed in italics. The book covers all core operating systems (OS) units, as well as one elective OS unit. The overall amount of coverage for each unit is always at least that recommended by CS2008, though sometimes the specific subtopics don't quite correspond exactly. Outside the OS area, this book's most substantial coverage is of Net-Centric Computing (NC); another major topic, transaction processing, comes from Information Management (IM). In each row, the listed chapters contain the bulk of the knowledge unit's coverage, though

some topics may be elsewhere.

Knowledge unit (italic indicates core units in CS2008)	**Chapter(s)**
OS/OverviewOfOperatingSystems	1
OS/OperatingSystemPrinciples	1, 7
OS/Concurrency	2, 4
OS/SchedulingAndDispatch	3
OS/MemoryManagement	6
OS/SecurityAndProtection	7, 11
OS/FileSystems	8
NC/Introduction	9
NC/NetworkCommunication (partial coverage)	9
NC/NetworkSecurity (partial coverage)	9
NC/WebOrganization (partial coverage)	9
NC/NetworkedApplications (partial coverage)	10
IM/TransactionProcessing	5

Your Feedback is Welcome

Comments, suggestions, and bug reports are welcome; please send email to `max@gustavus.edu`. Bug reports in particular can earn you a bounty of $2.56 apiece as a token of gratitude. (The great computer scientist Donald Knuth started this tradition. Given how close to bug-free his publications have become, it seems to work.) For purposes of this reward, the definition of a bug is simple: if as a result of your email the author chooses to make a change, then you have pointed out a bug. The change need not be the one you suggested, and the bug need not be technical in nature. Unclear writing qualifies, for example.

Features of the Text

Each chapter concludes with five standard elements. The last numbered section within the chapter is always devoted to security matters related to the chapter's topic. Next comes three different lists of opportunities for active participation by the student: exercises, programming projects, and exploration projects. Finally, the chapter ends with historical and bibliographic notes.

The distinction between exercises, programming projects, and exploration projects needs explanation. An exercise can be completed with no

outside resources beyond paper and pencil: you need just this textbook and your mind. That does not mean all the exercises are cut and dried, however. Some may call upon you to think creatively; for these, no one answer is correct. Programming projects require a nontrivial amount of programming; that is, they require more than making a small, easily identified change in an existing program. However, a programming project may involve other activities beyond programming. Several of them involve scientific measurement of performance effects, for example; these exploratory aspects may even dominate over the programming aspects. An exploration project, on the other hand, can be an experiment that can be performed with no real programming; at most you might change a designated line within an existing program. The category of exploration projects does not just include experimental work, however. It also includes projects that require you to do research on the Internet or using other library resources.

Supplemental Resources

The author of this text is making supplemental resources available on his own web site. Additionally, the publisher of the earlier first edition commissioned additional resources from independent supplement authors, which may still be available through the publisher's web site and would largely still apply to this revised edition. The author's web site, *http://gustavus.edu/+max/os-book/*, contains at least the following materials:

- Full text of this revised edition
- Source code in Java, C, or C++ for all programs that are shown in the text
- Artwork files for all figures in the text
- An errata list that will be updated on an ongoing basis

About the Revised Edition

Course Technology published the first edition of this book in January of 2006 and in October of 2010 assigned the copyright back to the author, giving him the opportunity to make it freely available. This revised edition closely follows the first edition; rather than being a thorough update, it is aimed at three narrow goals:

- All errata reported in the first edition are corrected.
- A variety of other minor improvements appear throughout, such as clarified explanations and additional exercises, projects, and end-of-chapter notes.
- Two focused areas received more substantial updates:
  - The explanation of Linux's scheduler was completely replaced to correspond to the newer "Completely Fair Scheduler" (CFS), including its group scheduling feature.
  - A new section, 4.9, was added on nonblocking synchronization.

In focusing on these limited goals, a key objective was to maintain as much compatibility with the first edition as possible. Although page numbering changed, most other numbers stayed the same. All new exercises and projects were added to the end of the corresponding lists for that reason. The only newly added section, 4.9, is near the end of its chapter; thus, the only changed section number is that the old Section 4.9 ("Security and Synchronization") became 4.10. Only in Chapter 4 did any figure numbers change.

It is my hope that others will join me in making further updates and improvements to the text. I am releasing it under a Creative Commons license that allows not just free copying, but also the freedom to make modifications, so long as the modified version is released under the same terms. In order to such modifications practical, I'm not just releasing the book in PDF form, but also as a collection of LaTeX source files that can be edited and then run through the `pdflatex` program (along with `bibtex` and `makeindex`). The source file collection also includes PDF files of all artwork figures; Course Technology has released the rights to the artwork they contracted to have redrawn.

If you produce a modified version of this text, the Creative Commons license allows you considerable flexibility in how you make your modified version available. I would urge you to send it back to me (`max@gustavus.edu`) so that I can add your version to the main web site–we will all benefit from having a central repository of progress. Separate materials to supplement the text would also be welcome. One category that occurs to me is animations or screencasts; the static figures in the text are rather limited. Another worthwhile project would be to transform the text into a more contribution-friendly form, such as a wiki.

Acknowledgments

This book was made possible by financial and logistical support from my employer, Gustavus Adolphus College, and moral support from my family. I would like to acknowledge the contributions of the publishing team, especially developmental editor Jill Batistick and Product Manager Alyssa Pratt. I am also grateful to my students for doing their own fair share of teaching. I particularly appreciate the often extensive comments I received from the following individuals, each of whom reviewed one or more chapters: Dan Cosley, University of Minnesota, Twin Cities; Allen Downey, Franklin W. Olin College of Engineering; Michael Goldweber, Xavier University; Ramesh Karne, Towson University; G. Manimaran, Iowa State University; Alexander Manov, Illinois Institute of Technology; Peter Reiher, University of California, Los Angeles; Rich Salz, DataPower Technology; Dave Schulz, Wisconsin Lutheran College; Sanjeev Setia, George Mason University; and Jon Weissman, University of Minnesota, Twin Cities. Although I did not adopt all their suggestions, I did not ignore any of them, and I appreciate them all.

In preparing the revised edition, I took advantage of suggestions from many readers. I would like to thank all of them, even those I've managed to lose track of, to whom I also apologize. Those I can thank by name are Joel Adams, Michael Brackney, Jack Briner, Justin Delegard, Ben Follis, MinChan Kim, Finn Kuusisto, Matt Lindner, Milo Martin, Gabe Schmidt, Fritz Sieker, and Alex Wauck.