Suggestions for Further Reading

TABLE OF CONTENTS

INTRODUCTION

The hardware technology that underlies computer systems has improved so rapidly and continuously for more than four decades that the ground rules for system design are constantly subject to change. It takes many years for knowledge and experience to be compiled, digested, and presented in the form of a book, so books about computer systems often seem dated or obsolete by the time they appear in print. Even though some underlying principles are unchanging, the rapid obsolescence of details acts to discourage prospective book authors, and as a result some important ideas are never documented in books. For this reason, an essential part of the study of computer systems is found in current—and, frequently, older—technical papers, professional journal articles, research reports, and occasional, unpublished memoranda that circulate among active workers in the field.

Despite that caveat, there are a few books, relatively recent additions to the literature in computer systems, that are worth having on the shelf. Until the mid-1980s, the books that existed were for the most part commissioned by textbook publishers to fill a market, and they tended to emphasize the mechanical aspects of systems rather than insight into their design. Starting around 1985, however, several very good books started to appear, when professional system designers became inspired to capture their insights. The appearance of these books also suggests that the concepts involved in computer system design are finally beginning to stabilize a bit. (Or it may just be

that computer system technology is beginning to shorten the latencies involved in book publishing.)

The heart of the computer systems literature is found in published papers. Two of the best sources are Association for Computing Machinery (ACM) publications: the journal *ACM Transactions on Computer Systems (TOCS)* and the bi-annual series of conference proceedings, the *ACM Symposium on Operating Systems Principles (SOSP)*. The best papers of each SOSP are published in a following issue of TOCS, and the rest—in recent years all—of the papers of each symposium appear in a special edition of *Operating Systems Review*, an ACM special interest group quarterly that publishes an extra issue in symposium years. Three other regular symposia are also worth following: the *European Conference on Computer Systems (EuroSys)*, the *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, and the USENIX *Symposium on Networked Systems Design and Implementation (NSDI)*. These sources are not the only ones—worthwhile papers about computer systems appear in many other journals, conferences, and workshops. Complete copies of most of the papers listed here, including many of the older ones, can be found on the World Wide Web by an on-line search for an author's last name and a few words of the paper title. Even papers whose primary listing requires a subscription are often posted elsewhere as open resources.

The following pages contain suggestions for further reading about computer systems, both papers and books. The list makes no pretensions of being complete. Instead, the suggestions have been selected from a vast literature to emphasize the best available thinking, best illustrations of problems, and most interesting case studies of computer systems. The readings have been reviewed for obsolescence, but it is often the case that a good idea is still best described by a paper from some time ago, where the idea was developed in a context that no longer seems very interesting. Sometimes that early context is much simpler than today's systems, thus making it easier to see how the idea works. Often, an early author was the first on the scene, so it was necessary to describe things more completely than do modern authors who usually assume significant familiarity with the surroundings and with all of the predecessor systems. Thus the older readings included here provide a very useful complement to current works.

By its nature, the study of the engineering of computer systems overlaps with other areas of computer science, particularly computer architecture, programming languages, databases, information retrieval, security, and data communications. Each of those areas has an extensive literature of its own, and it is often not obvious where to draw the boundary lines. As a general rule, this reading list tries to provide only first-level guidance on where to start in those related areas.

One thing the reader must watch for is that the terminology of the computer systems field is not agreed upon, so the literature is often confusing even to the professional. In addition, the quality level of the literature is quite variable, ranging from the literate through the readable to the barely comprehensible. Although the selections here try to avoid that last category, the reader must still be prepared for some papers, however important in their content, that do not explain their subject as well as they could.

In the material that follows, each citation is accompanied by a comment suggesting why that paper is worth reading—its importance, interest, and relation to other readings. When a single paper serves more than one area of interest, cross-references appear rather than repeating the citation.

1 Systems

As mentioned above, a few wonderful and several really good books about computer systems have recently begun to appear. Here are the must-have items for the reference shelf of the computer systems designer. In addition to these books, the later groupings of readings by topic include other books, generally of narrower interest.

1.1 Wonderful Books About Systems

1.1.1 David A. Patterson and John L. Hennessy. *Computer Architecture: A Quantitative Approach*. Morgan Kaufman, fourth edition, 2007. ISBN: 978–0–12–370490–0. 704 + various pages (paperback). The cover gives the authors' names in the opposite order.

This book provides a spectacular tour-de-force that explores much of the design space of current computer architecture. One of the best features is that each area includes a discussion of misguided ideas and their pitfalls. Even though the subject matter gets very sophisticated, the book is always very readable. The book is opinionated (with a strong bias toward RISC architecture), but nevertheless this is a definitive work on computer organization from the system perspective.

1.1.2 Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991. ISBN: 978–0–471–50336–1. 720 pages.

Much work on performance analysis of computer systems originates in academic settings and focuses on analysis that is mathematically tractable rather than on measurements that matter. This book is at the other end of the spectrum. It is written by someone with extensive industrial experience but an academic flair for explaining things. If you have a real performance analysis problem, it will tell you how to tackle it, how to avoid measuring the wrong thing, and how to step by other pitfalls.

1.1.3 Frederick P. Brooks Jr. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 20th Anniversary edition, 1995. ISBN: 978–0–201–83595–3 (paperback). 336 pages.

Well-written and full of insights, this reading is by far the most significant one on the subject of controlling system development. This is where you learn why adding more staff to a project that is behind schedule will delay it further. Although a few of the chapters are now a bit dated, much of the material here is timeless. Trouble in system development is also timeless, as evidenced by continual reports

of failures of large system projects. Most successful system designers have a copy of this book on their bookshelf, and some claim to reread it at least once a year. Most of the 1995 edition is identical to the first, 1974, edition; the newer edition adds Brooks' *No Silver Bullets* paper (which is well worth reading) and some summarizing chapters.

1.1.4 Lawrence Lessig. *Code and Other Laws of Cyberspace, Version 2.0*. Basic Books, 2006. ISBN: 978-0-465-03914-28 (paperback). 432 pages; 978-0-465-03913-5 (paperback). 320 pages. Also available on-line at `http://codev2.cc/`

This book is an updated version of an explanation by a brilliant teacher of constitutional law of exactly how law, custom, market forces, and architecture together regulate things. In addition to providing a vocabulary to discuss many of the legal issues surrounding technology and the Internet, a central theme of this book is that because technology raises issues that were foreseen neither by law nor custom, the default is that it will be regulated entirely by market forces and architecture, neither of which is subject to the careful and deliberative thought that characterize the development of law and custom. If you have any interest in the effect of technology on intellectual property, privacy, or free speech, this book is required reading.

1.1.5 Jim [N.] Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, San Mateo, California, 1993 (Look for the low-bulk paper edition, which became available with the third printing in 1994). ISBN: 978-1-55860-190-1. 1,070 pages.

All aspects of fault tolerance, atomicity, coordination, recovery, rollback, logs, locks, transactions, and engineering trade-offs for performance are pulled together in this comprehensive book. This is the definitive work on transactions. Though not intended for beginners, given the high quality of its explanations, this complex material is surprisingly accessible. The glossary of terms is excellent, whereas the historical notes are good as far as they go, but are somewhat database-centric and should not be taken as the final word.

1.1.6 Alan F. Westin. *Privacy and Freedom*. Atheneum Press, 1967. 487 pages. (Out of print.)

If you have any interest in privacy, track down a copy of this book in a library or used-book store. It is the comprehensive treatment, by a constitutional lawyer, of what privacy is, why it matters, and its position in the U.S. legal framework.

1.1.7 Ross Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, second edition, 2008. ISBN: 978-0-470-06852-6. 1,040 pages.

This book is remarkable for the range of system security problems it considers, from taxi mileage recorders to nuclear command and control systems. It provides

great depth on the mechanics, assuming that the reader already has a high-level picture. The book is sometimes quick in its explanations; the reader must be quite knowledgeable about systems. One of its strengths is that most of the discussions of how to do it are immediately followed by a section titled "What goes wrong", exploring misimplementations, fallacies, and other modes of failure. The first edition is available on-line.

1.2 Really Good Books About Systems

1.2.1 Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice-Hall, third edition, 2008. ISBN: 978–0–13–600663–3 (hardcover). 952 pages.

This book provides a thorough tutorial introduction to the world of operating systems but with a tendency to emphasize the mechanics. Insight into why things are designed the way they are is there, but in many cases requires teasing out. Nevertheless, as a starting point, it is filled with street knowledge that is needed to get into the rest of the literature. It includes useful case studies of GNU/Linux, Windows Vista, and Symbian OS, an operating system for mobile phones.

1.2.2 Thomas P. Hughes. *Rescuing Prometheus*. Vintage reprint (paperback), originally published in 1998. ISBN: 978–0679739388. 372 pages.

A retired professor of history and sociology explains the stories behind the management of four large-scale, one-of-a-kind system projects: the Sage air defense system, the Atlas rocket, the Arpanet (predecessor of the Internet), and the design phase of the Big Dig (Boston Central Artery/Tunnel). The thesis of the book is that such projects, in addition to unique engineering, also had to develop a different kind of management style that can adapt continuously to change, is loosely coupled with distributed control, and can identify a consensus among many players.

1.2.3 Henry Petroski. *Design Paradigms: Case Histories of Error and Judgment in Engineering*. Cambridge University Press, 1994. ISBN: 978–0–521–46108–5 (hardcover), 978–0–521–46649–3 (paperback). 221 pages.

This remarkable book explores how the mindset of the designers (in the examples, civil engineers) allowed them to make what in retrospect were massive design errors. The failures analyzed range from the transportation of columns in Rome through the 1982 collapse of the walkway in the Kansas City Hyatt Regency Hotel, with a number of famous bridge collapses in between. Petroski analyzes particularly well how a failure of a scaled-up design often reveals that the original design worked correctly, but for a different reason than originally thought. There is no mention of computer systems in this book, but it contains many lessons for computer system designers.

1.2.4 Bruce Schneier. *Applied Cryptography*. John Wiley and Sons, second edition, 1996. ISBN: 978–0–471–12845–8 (hardcover), 978–0–471–11709–4 (paperback). 784 pages.

Here is everything you might want to know about cryptography and cryptographic protocols, including a well-balanced perspective on what works and what doesn't. This book saves the need to read and sort through the thousand or so technical papers on the subject. Protocols, techniques, algorithms, real-world considerations, and source code can all be found here. In addition to being competent, it is also entertainingly written and very articulate. Be aware that a number of minor errors have been reported in this book; if you are implementing code, it would be a good idea to verify the details by consulting reading *1.3.13*.

1.2.5 Radia Perlman. *Interconnections, second edition: Bridges, Routers, Switches, and Internetworking Protocols*. Addison-Wesley, 1999. ISBN: 978-0-201-63448-8. 560 pages.

This book presents everything you could possibly want to know about how the network layer actually works. The style is engagingly informal, but the content is absolutely first-class, and every possible variation is explored. The previous edition was simply titled *Interconnections: bridges and routers*.

1.2.6 Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufman, fourth edition, 2007. ISBN: 978-0-12-370548-8. 848 pages.

This book provides a systems perspective on computer networks. It represents a good balance of why networks are they way they are and a discussion of the important protocols in use. It follows a layering model but presents fundamental concepts independent of layering. In this way, the book provides a good discussion of timeless ideas as well as current embodiments of those ideas.

1.3 Good Books on Related Subjects Deserving Space on the Systems Bookshelf

There are several other good books that many computer system professionals insist on having on their bookshelves. They don't appear in one of the previous categories because their central focus is not on systems or because the purpose of the book is somewhat narrower.

1.3.1 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. McGraw-Hill, second edition, 2001. 1,184 pages. ISBN: 978-0-07-297054-8 (hardcover); 978-0-262-53196-2 (M.I.T. Press paperback, not sold in U.S.A.)

1.3.2 Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufman, 1996. 872 pages ISBN: 978-1-55860-348-6.

Occasionally, a system designer needs an algorithm. Cormen et al. and Lynch's books are the place to find that algorithm, together with the analysis necessary to decide whether or not it is appropriate for the application. In a reading list on

theory, these two books would almost certainly be in one of the highest categories, but for a systems list they are better identified as supplementary.

1.3.3 Douglas K. Smith and Robert C. Alexander. *Fumbling the Future*. William Morrow and Company, 1988. ISBN: 978–0–688–06959–9 (hardcover), 978–1–58348266–7 (iuniverse paperback reprint). 274 pages.

The history of computing is littered with companies that attempted to add general-purpose computer systems to an existing business—for examples, Ford, Philco, Zenith, RCA, General Electric, Honeywell, A. T. & T., and Xerox. None has succeeded, perhaps because when the going gets tough the option of walking away from this business is too attractive. This book documents how Xerox managed to snatch defeat from the jaws of victory by inventing the personal computer, then abandoning it.

1.3.4 Marshall Kirk McKusick, Keith Bostic, and Michael J. Karels. *The Design and Implementation of the 4.4BSD Operating System.* Addison-Wesley, second edition, 1996. ISBN: 978–0–201–54979–9. 606 pages.

This book provides a complete picture of the design and implementation of the Berkeley version of the UNIX® operating system. It is well-written and full of detail. The 1989 first edition, describing 4.3BSD, is still useful.

1.3.5 Katie Hafner and John Markoff. *Cyberpunk: Outlaws and Hackers on the Computer Frontier*. Simon & Schuster (Touchstone), 1991, updated June 1995. ISBN: 978–0–671–68322–1 (hardcover), 978–0–684–81862–7 (paperback). 368 pages.

This book is a very readable, yet thorough, account of the scene at the ethical edges of cyberspace: the exploits of Kevin Mitnick, Hans Hubner, and Robert Tappan Morris. It serves as an example of a view from the media, but an unusually well-informed view.

1.3.6 Deborah G. Johnson and Helen Nissenbaum. *Computers, Ethics & Social Values*. Prentice-Hall, 1995. ISBN: 978–0–13–103110–4 (paperback). 714 pages.

A computer system designer is likely to consider reading a treatise on ethics to be a terribly boring way to spend the afternoon, and some of the papers in this extensive collection do match that stereotype. However, among the many scenarios, case studies, and other reprints in this volume are a large number of interesting and thoughtful papers about the human consequences of computer system design. This collection is a good place to acquire the basic readings concerning privacy, risks, computer abuse, and software ownership as well as professional ethics in computer system design.

1.3.7 Carliss Y. Baldwin and Kim B. Clark. *Design Rules: Volume 1, The Power of Modularity*. M.I.T. Press, 2000. ISBN: 978–0–262–02466–2. 471 pages.

This book focuses wholely on modularity (as used by the authors, this term merges modularity, abstraction, and hierarchy) and offers an interesting

representation of interconnections to illustrate the power of modularity and of clean, abstract interfaces. The work uses these same concepts to interpret several decades of developments in the computer industry. The authors, from the Harvard Business School, develop a model of the several ways in which modularity operates by providing design options and making substitution easy. By the end of the book, most readers will have seen more than they wanted to know, but there are some ideas here that are worth at least a quick reading. (Despite the "Volume 1" in the title, there does not yet seem to be a Volume 2.)

1.3.8 Andrew S. Tanenbaum. *Computer Networks*. Prentice-Hall, fourth edition, 2003. ISBN: 978–0–13–066102–9. 813 pages.

This book provides a thorough tutorial introduction to the world of networks. Like the same author's book on operating systems (see reading *1.2.1*), this one also tends to emphasize the mechanics. But again it is a storehouse of up-to-date street knowledge, this time about computer communications, that is needed to get into (or perhaps avoid the need to consult) the rest of the literature. The book includes a selective and thoughtfully annotated bibliography on computer networks. An abbreviated version of this same material, sufficient for many readers, appears as a chapter of the operating systems book.

1.3.9 David L. Mills. *Computer Network Time Synchronization: The Network Time Protocol*. CRC Press/Taylor & Francis, 2006. ISBN: 978–0849358050. 286 pages.

A comprehensive but very readable explanation of the Network Time Protocol (NTP), an under-the-covers protocol of which most users are unaware: NTP coordinates multiple timekeepers and distributes current date and time information to both clients and servers.

1.3.10 Robert G. Gallager. *Principles of Digital Communication*. Cambridge University Press, 2008. ISBN: 978–0–521–87907–1. 422 pages.

This intense textbook focuses on the theory that underlies the link layer of data communication networks. It is not for casual browsing or for those easily intimidated by mathematics, but its an excellent reference source for analysis.

1.3.11 Daniel P. Siewiorek and Robert S. Swarz. *Reliable Computer Systems: Design and Evaluation*. A. K. Peters Ltd., third edition, 1998. ISBN: 978–1–56881–092–8. 927 pages.

This is probably the best comprehensive treatment of reliability that is available, with well-explained theory and reprints of several case studies from recent literature. Its only defect is a slight "academic" bias in that little judgment is expressed on alternative methods, and some examples are without warning of systems that were never really deployed. The first, 1982, edition, with the title *The Theory and Practice of Reliable System Design*, contains an almost completely different (and much older) set of case studies.

1.3.12 Bruce Schneier. *Secrets & Lies/Digital Security in a Networked World*. John Wiley & Sons, 2000. ISBN: 978-0-471-25311-2 (hardcover), 978-0-471-45380-2 (paperback). 432 pages.

This overview of security from a systems perspective provides much motivation, many good war stories (though without citations), and a high-level outline of how one achieves a secure system. Being an overview, it provides no specific guidance on the mechanics, other than to rely on people who know what they are doing. This is excellent book particularly for the manager who wants to go beyond the buzzwords and get an idea of what achieving computer system security involves.

1.3.13 A[lfred] J. Menezes, Paul C. Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997. ISBN: 978-08493-8523-0. 816 pages.

This book is exactly what its title claims: a very complete handbook on putting cryptography to work. It lacks the background and perspective of Reading 1.2.4, and it is extremely technical, which makes parts of it inaccessible to less mathematically inclined readers. But its precise definitions and careful explanations make this by far the best reference book available on the subject.

1.3.14 Johannes A. Buchman. *Introduction to Cryptography*. Springer, second edition, 2004. ISBN: 978-0-387-21156-56 (hardcover). 335 pages.

Buchman provides a nice, concise introduction to number theory for cryptography.

1.3.15 Simson Garfinkel and Gene [Eugene H.] Spafford. *Practical Unix and Internet Security*. O'Reilly & Associates, Sebastopol, California, third edition, 2003. ISBN: 978-59600323-4 (paperback). 986 pages.

This is a really comprehensive guide to how to run a network-attached UNIX system with some confidence that it is relatively safe against casual intruders. In addition to providing practical information for a system manager, it incidentally gives the reader quite a bit of insight into the style of thinking and design needed to provide security.

1.3.16 Simson Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly & Associates, Sebastopol, California, 1995. ISBN: 978-1-56592-098-9 (paperback). 430 pages.

Nominally a user's guide to the PGP encryption package developed by Phil Zimmermann, this book starts out with six very readable overview chapters on the subject of encryption, its history, and the political and licensing environment that surrounds encryption systems. Even the later chapters, which give details on how to use PGP, are filled with interesting tidbits and advice applicable to all encryption uses.

1.3.17 Warwick Ford and Michael S. Baum. *Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption*. Prentice-Hall, second edition, 2000. ISBN: 978-0-13-027276-8. 640 pages.

Although the title implies more generality, this book is about public key infrastructure: certificate authorities, certificates, and their legal status in practice. The authors are a technologist (Ford) and a lawyer (Baum). The book provides very thorough coverage and is a good way to learn a lot about the subject. Because the status of this topic changes rapidly, however, it should be considered a snapshot rather than the latest word.

1.4 Ways of Thinking About Systems

Quite a few books try to generalize the study of systems. They tend to be so abstract, however, that it is hard to see how they apply to anything, so none of them are listed here. Instead, here are five old but surprisingly relevant papers that illustrate ways to think about systems. The areas touched are allometry, aerodynamics, hierarchy, ecology, and economics.

1.4.1 J[ohn] B[urdon] S[anderson] Haldane (1892–1964). On being the right size. In *Possible Worlds and Other Essays*, pages 20–28. Harper and Brothers Publishers, 1928. Also published by Chatto & Windus, London, 1927, and recently reprinted in John Maynard Smith, editor, *On Being the Right Size and Other Essays*, Oxford University Press, 1985. ISBN: 0–19–286045–3 (paperback), pages 1–8.

This is the classic paper that explains why a mouse the size of an elephant would collapse if it tried to stand up. It provides lessons on how to think about incommensurate scaling in all kinds of systems.

1.4.2 Alexander Graham Bell (1847–1922). The tetrahedral principle in kite structure. *National Geographic Magazine 14*, 6 (June 1903), pages 219–251.

This classic paper demonstrates that arguments based on scale can be quite subtle. This paper—written at a time when physicists were still debating the theoretical possibility of building airplanes—describes the obvious scale argument against heavier-than-air craft and then demonstrates that one can increase the scale of an airfoil in different ways and that the obvious scale argument does not apply to all those ways. (This paper is a rare example of unreviewed vanity publication of an interesting engineering result. The *National Geographic* was—and still is—a Bell family publication.)

1.4.3 Herbert A. Simon (1916–2001). The architecture of complexity. *Proceedings of the American Philosophical Society 106*, 6 (December 1962), pages 467–482. Republished as Chapter 4, pages 84–118, of *The Sciences of the Artificial*, M.I.T. Press, Cambridge, Massachusetts, 1969. ISBN: 0–262–191051–6 (hardcover); 0–262–69023–3 (paperback).

This paper is a tour-de-force of how hierarchy is an organizing tool for complex systems. The examples are breathtaking in their range and scope—from watchmaking and biology through political empires. The style of thinking shown in this paper suggests that it is not surprising that Simon later received the 1978 Nobel Prize in economics.

1.4.4 LaMont C[ook] Cole (1916–1978). Man's effect on nature. *The Explorer: Bulletin of the Cleveland Museum of Natural History 11*, 3 (Fall 1969), pages 10–16.

This brief article looks at the Earth as an ecological system in which the actions of humans lead both to surprises and to propagation of effects. It describes a classic example of the propagation of effects: attempts to eliminate malaria in North Borneo led to an increase in the plague and roofs caving in.

1.4.5 Garrett [James] Hardin (1915–). The tragedy of the commons. *Science 162*, 3859 (December 13, 1968), pages 1243–1248. Extensions of "the tragedy of the commons". *Science 280*, 5364 (May 1, 1998), pages 682–683.

This seminal paper explores a property of certain economic situations in which Adam Smith's "invisible hand" works against everyone's interest. It is interesting for its insight into how to predict things about otherwise hard-to-model systems. In revisiting the subject 30 years later, Hardin suggested that the adjective "unmanaged" should be placed in front of "commons". Rightly or wrongly, the Internet is often described as a system to which the tragedy of the (unmanaged) commons applies.

1.5 Wisdom About System Design

Before reading anything else on this topic, one should absorb the book by Brooks, *The Mythical Man-Month*, reading *1.1.3*, and the essay by Simon, "The architecture of complexity", reading *1.4.3*. The case studies on control of complexity in Section 1.9 also are filled with wisdom.

1.5.1 Richard P. Gabriel. Worse is better. Excerpt from LISP: good news, bad news, how to win BIG, *AI Expert 6*, 6 (June 1991), pages 33–35.

This paper explains why doing the thing expediently sometimes works out to be a better idea than doing the thing right.

1.5.2 Henry Petroski. Engineering: History and failure. *American Scientist 80*, 6 (November–December 1992), pages 523–526.

Petroski provides insight along the lines that one primary way that engineering makes progress is by making mistakes, studying them, and trying again. Petroski also visits this theme in two books, the most recent being reading *1.2.3*.

1.5.3 Fernando J. Corbató. On building systems that will fail. *Communications of the ACM 34*, 9 (September 1991), pages 72–81. (Reprinted in the book by Johnson and Nissenbaum, reading *1.3.6*.)

The central idea in this 1991 Turing Award Lecture is that all ambitious systems will have failures, but those that were designed with that expectation are more likely to eventually succeed.

1.5.4 Butler W. Lampson. Hints for computer system design. *Proceedings of the Ninth ACM Symposium on Operating Systems Principles*, in *Operating Systems Review 17*, 5 (October 1983), pages 33–48. Later republished, but with less satisfactory copy editing, in *IEEE Software 1*, 1 (January 1984), pages 11–28.

This encapsulation of insights is expressed as principles that seem to apply to more than one case. It is worth reading by all system designers.

1.5.5 Jon Bentley. The back of the envelope—programming pearls. *Communications of the ACM 27*, 3 (March 1984), pages 180–184.

One of the most important tools of a system designer is the ability to make rough but quick estimates of how big, how long, how fast, or how expensive a design will be. This brief note extols the concept and gives several examples.

1.5.6 Jeffrey C. Mogul. Emergent (mis)behavior vs. complex software systems. *Proceedings of the First European Conference on Computer Systems* (EuroSys 2006, Leuven, Belgium), pages 293–304. ACM Press, 2006, ISBN: 1–59593–322–0. Also in *Operating Systems Review 40*, 4 (October 2006).

This paper explores in depth the concept of emergent properties described in Chapter 1, providing a nice collection of examples and tying together issues and problems that arise throughout computer and network system design. It also suggests a taxonomy of emergent properties, lays out suggestions for future research, and includes a comprehensive and useful bibliography.

1.5.7 Pamela Samuelson, editor. Intellectual property for an information age. *Communications of the ACM 44*, 2 (February 2001), pages 67–103.

This work is a special section comprising several papers about the challenges of intellectual property in a digital world. Each of the individual articles is written by a member of a new generation of specialists who understand both technology and law well enough to contribute some thoughtful insights to both domains.

1.5.8 Mark R. Chassin and Elise C. Becher. The wrong patient. *Annals of Internal Medicine 136* (June 2002), pages 826–833.

This paper is a good example, first, of how complex systems fail for complex reasons and second, of the value of the "keep digging" principle. The case study presented here centers on a medical system failure in which the wrong patient was operated on. Rather than just identifying the most obvious reason, the case study concludes that there were a dozen or more opportunities in which the error that led to the failure should have been detected and corrected, but for various reasons all of those opportunities were missed.

1.5.9 P[hillip] J. Plauger. Chocolate. *Embedded Systems Programming* 7, 3 (March 1994), pages 81–84.

This paper provides a remarkable insight based on the observation that many failures in a bakery can be remedied by putting more chocolate into the mixture. The author manages, with only a modest stretch, to convert this observation into a more general technique of keeping recovery simple, so that it is likely to succeed.

1.6 Changing Technology and its Impact on Systems

1.6.1 Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics 38*, 8 (April 19, 1965), pages 114–117. Reprinted in *Proceedings of the IEEE 86*, 1 (January 1998), pages 82–85.

This paper defined what we now call Moore's law. The phenomena Moore describes have driven the rate of technology improvement for more than four decades. This paper articulates why and displays the first graph to plot Moore's law, based on five data points.

1.6.2 John L. Hennessy and Norman P. Jouppi. Computer technology and architecture: An evolving interaction. *IEEE Computer 24*, 9 (September 1991), pages 19–29.

Although some of the technology examples are a bit of out of date, the systems thinking and the paper's insights remain relevant.

1.6.3 Ajanta Chakraborty and Mark R. Greenstreet. Efficient self-timed interfaces for crossing clock domains. *Proceedings of the Ninth International Symposium on Asynchronous Circuits and Systems*, IEEE Computer Society (May 2003), pages 78–88. ISBN: 0–7695–1898–2.

This paper addresses the challenge of having a fast, global clock on a chip by organizing the resources on a chip as a number of synchronous islands connected by asynchronous links. This design may pose problems for constructing perfect arbiters (see Section 5.2.8).

1.6.4 Anant Agarwal and Markus Levy. The KILL Rule for multicore. *44th ACM/IEEE Conference on Design Automation* (June 2007), pages 750–753. ISBN: 978–1–59593–627–1.

This short paper looks ahead to multiprocessor chips that contain not just four or eight, but thousands of processors. It articulates a rule for power-efficient designs: Kill If Less than Linear. For example, the designer should increase the chip area devoted to a resource such as a cache only if for every 1% increase in area there is at least a 1% increase in chip performance. This rule focuses attention on those design elements that make most effective use of the chip area and from back-of-the-envelope calculations favors increasing processor count (which the paper assumes to provide linear improvement) over other alternatives.

1.6.5 Stephen P. Walborn et al. Quantum erasure. *American Scientist 91*, 4 (July–August 2003), pages 336–343.

This paper was written by physicists and requires a prerequisite of undergraduate-level modern physics, but it manages to avoid getting into graduate-level quantum mechanics. The strength of the article is its clear identification of what is reasonably well understood and what is still a mystery about these phenomena. That identification seems to be of considerable value both to students of physics, who may be inspired to tackle the parts that are not understood, and to students of cryptography, because knowing what aspects of quantum cryptography are still mysteries may be important in deciding how much reliance to place on it.

1.7 Dramatic Visions

Once in a while a paper comes along that either has a dramatic vision of what future systems might do or takes a sweeping new look at some aspect of systems design that had previously been considered to be settled. The ideas found in the papers listed in reading sections 1.7 and 1.8 often become part of the standard baggage of all future writers in the area, but the reprises rarely do justice to the originals, which are worth reading if only to see how the mind of a visionary (or revisionist) works.

1.7.1 Vannevar Bush. As we may think. A*tlantic Monthly 176*, 1 (July 1945), pages 101–108. Reprinted in Adele J. Goldberg, *A History of Personal Workstations*, Addison-Wesley, 1988, pages 237–247 and also in Irene Greif, ed., *Computer-Supported Cooperative Work: A Book of Readings*, Morgan Kaufman, 1988. ISBN: 0–934613–57–5.

Bush looked at the (mostly analog) computers of 1945 and foresaw that they would someday be used as information engines to augment the human intellect.

1.7.2 John G. Kemeny, with comments by Robert M. Fano and Gilbert W. King. A library for 2000 A.D. In Martin Greenberger, editor, *Management and the Computer of the Future*, M.I.T. Press and John Wiley, 1962, pages 134–178. (Out of print.)

It has taken 40 years for technology to advance far enough to make it possible to implement Kemeny's vision of how the library might evolve when computers are used in its support. Unfortunately, the engineering that is required still hasn't been done, so the vision has not yet been realized, but Google has stated a similar vision and is making progress in realizing it; see reading *3.2.4*.

1.7.3 [Alan C. Kay, with the] Learning Research Group. *Personal Dynamic Media*. Xerox Palo Alto Research Center Systems Software Laboratory Technical Report SSL–76–1 (undated, circa March 1976).

Alan Kay was imagining laptop computers and how they might be used long before most people had figured out that desktop computers might be a good idea. He gave many inspiring talks on the subject, but he rarely paused long enough to write anything down. Fortunately, his colleagues captured some of his thoughts in this technical report. An edited version of this report, with some

pictures accidentally omitted, appeared in a journal in the year following this technical report: Alan [C.] Kay and Adele Goldberg. Personal dynamic media. *IEEE Computer 10*, 3 (March 1977), pages 31–41. This paper was reprinted with omitted pictures restored in Adele J. Goldberg, *A history of personal workstations*, Addison-Wesley, 1988, pages 254–263. ISBN: 0–201–11259–0.

1.7.4 Doug[las] C. Engelbart. *Augmenting Human Intellect: A Conceptual Framework*. Research Report AFOSR–3223, Stanford Research Institute, Menlo Park, California, October 1962. Reprinted in Irene Greif, ed., *Computer-Supported Cooperative Work: A Book of Readings*, Morgan Kaufman, 1988. ISBN: 0–934613–57–5.

In the early 1960s Engelbart saw that computer systems would someday be useful in myriad ways as personal tools. Unfortunately, the technology of his time, multimillion-dollar mainframes, was far too expensive to make his vision practical. Today's personal computers and engineering workstations have now incorporated many of his ideas.

1.7.5 F[ernando] J. Corbató and V[ictor] A. Vyssotsky. Introduction and overview of the Multics system. A*FIPS 1965 Fall Joint Computer Conference 27*, part I (1965), pages 185–196.

Working from a few primitive examples of time-sharing systems, Corbató and his associates escalated the vision to an all-encompassing computer utility. This paper is the first in a set of six about Multics in the same proceedings, pages 185–247.

1.8 Sweeping New Looks

1.8.1 Jack B. Dennis and Earl C. Van Horne. Programming semantics for multiprogrammed computations. *Communications of the ACM 9*, 3 (March 1966), pages 143–155.

This paper set the ground rules for thinking about concurrent activities, both the vocabulary and the semantics.

1.8.2 J. S. Liptay. Structural aspects of the System/360 model 85: II. The cache. *IBM Systems Journal* 7, 1 (1968), pages 15–21.

The idea of a cache, look-aside, or slave memory had been suggested independently by Francis Lee and Maurice Wilkes some time around 1963, but it was not until the advent of LSI technology that it became feasible to actually build one in hardware. As a result, no one had seriously explored the design space options until the designers of the IBM System/360 model 85 had to come up with a real implementation. Once this paper appeared, a cache became a requirement for most later computer architectures.

1.8.3 Claude E. Shannon. The communication theory of secrecy systems. *Bell System Technical Journal 28*, 4 (October 1949), pages 656–715.

This paper provides the underpinnings of the theory of cryptography, in terms of information theory.

1.8.4 Whitfield Diffie and Martin E. Hellman. Privacy and authentication: An introduction to cryptography. *Proceedings of the IEEE 67*, 3 (March 1979), pages 397–427.

This is the first technically competent paper on cryptography since Shannon in the unclassified literature, and it launched modern unclassified study. It includes a complete and scholarly bibliography.

1.8.5 Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory IT–22*, 6 (November 1976), pages 644–654.

Diffie and Hellman were the second inventors of public key cryptography (the first inventor, James H. Ellis, was working on classified projects for the British Government Communications Headquarters at the time, in 1970, and was not able to publish his work until 1987). This is the paper that introduced the idea to the unclassified world.

1.8.6 Charles T. Davies, Jr. Data processing spheres of control. *IBM Systems Journal 17*, 2 (1978), pages 179–198. Charles T. Davies, Jr. Recovery semantics for a DB/DC system. *1973 ACM National Conference 28* (August 1973), pages 136–141.

This pair of papers—vague but thought-provoking—gives a high-level discussion of "spheres of control", a notion closely related to atomicity. Everyone who writes about transactions mentions that they found these two papers inspiring.

1.8.7 Butler W. Lampson and Howard Sturgis. Crash recovery in a distributed data storage system. Working paper, Xerox Palo Alto Research Center, November 1976 and April 1979. (Never published)

Jim Gray called the 1976 version of this paper "an underground classic". The 1979 version presents the first good definition of models of failure. Both describe algorithms for coordinating distributed updates; they are sufficiently different that both are worth reading.

1.8.8 Leonard Kleinrock. *Communication Nets: Stochastic Message Flow and Delay*. McGraw-Hill, 1964. Republished by Dover, 2007. ISBN: 0–486–45880–6. 224 pages.

1.8.9 Paul Baran, S. Boehm, and J. W. Smith. *On Distributed Communications*. A series of 11 memoranda of the RAND Corporation, Santa Monica, California, August 1964.

Since the growth in the Internet's popularity, there has been considerable discussion about who first thought of packet switching. It appears that Leonard Kleinrock, working in 1961 on his M.I.T. Ph.D. thesis on more effective ways

of using wired networks, and Paul Baran and his colleagues at Rand, working in 1961 on survivable communications, independently proposed the idea of packet switching at about the same time; both wrote internal memoranda in 1961 describing their ideas. Neither one actually used the words "packet switching", however; that was left to Donald Davies of the National Physical Laboratory, who coined that label several years later.

1.8.10 Lawrence G. Roberts and Barry D. Wessler. Computer network development to achieve resource sharing. *AFIPS Spring Joint Computer Conference 36* (May 1970), pages 543–549.

This paper and four others presented at the same conference session (pages 543–597) represent the first public description of the ARPANET, the first successful packet-switching network and the prototype for the Internet. Two years later, *AFIPS Spring Joint Computer Conference 40* (1972), pages 243–298, presented five additional, closely related papers. The discussion of priority concerning reading *1.8.8* and reading *1.8.9* is somewhat academic; it was Roberts's sponsorship of the ARPANET that demonstrated the workability of packet switching.

1.8.11 V[inton G.] Cerf et al. Delay-Tolerant Networking Architecture. *Request for Comments RFC 4838*, Internet Engineering Task Force (April 1997).

This document describes an architecture that evolved from a vision for an Interplanetary Internet, an Internet-like network for interplanetary distances. This document introduces several interesting ideas and highlights some assumptions that people make in designing networks without realizing it. NASA performed its first successful tests of a prototype implementation of a delay-tolerant network.

1.8.12 Jim Gray et al. *Terascale Sneakernet. Using Inexpensive Disks for Backup, Archiving, and Data Exchange*. Microsoft Technical Report MS-TR-02-54 (May 2002). http://arxiv.org/pdf/cs/0208011

Sneakernet is a generic term for transporting data by physically delivering a storage device rather than sending it over a wire. Sneakernets are attractive when data volume is so large that electronic transport will take a long time or be too expensive, and the latency until the first byte arrives is less important. Early sneakernets exchanged programs and data using floppy disks. More recently, people have exchanged data by burning CDs and carrying them. This paper proposes to build a sneakernet by sending hard disks, encapsulated in a small, low-cost computer called a storage brick. This approach allows one to transfer by mail terabytes of data across the planet in a few days. By virtue of including a computer and operating system, it minimizes compatibility problems that arise when transferring the data to another computer.

Several other papers listed under specific topics also provide sweeping new looks or have changed the way people think about systems: Simon, The architecture of complexity, reading *1.4.3*; Thompson, Reflections on trusting trust, reading *11.3.3*;

Lampson, Hints for computer system design, reading *1.5.4*; and Creasy's VM/370 paper, reading *5.6.1*.

1.9 Keeping Big Systems Under Control

1.9.1 F[ernando] J. Corbató and C[harles] T. Clingen. A managerial view of the Multics system development. In Peter Wegner, *Research Directions in Software Technology*, M.I.T. Press, Cambridge, Massachusetts, 1979, pages 139–158. ISBN: 0–262–23096–8.

1.9.2 W[illiam A.] Wulf, R[oy] Levin, and C. Pierson. Overview of the Hydra operating system development. *Proceedings of the Fifth ACM Symposium on Operating Systems Principles*, in *Operating Systems Review 9*, 5 (November 1975), pages 122–131.

1.9.3 Thomas R. Horsley and William C. Lynch. Pilot: A software engineering case study. *Fourth International Conference on Software Engineering* (September 1979), pages 94–99.

These three papers are early descriptions of the challenges of managing and developing large systems. They are still relevant and easy to read, and provide complementary insights.

1.9.4 Effy Oz. When professional standards are lax: The CONFIRM failure and its lessons. *Communications of the ACM 37*, 10 (October 1994), pages 30–36.

CONFIRM is an airline/hotel/rental-car reservation system that never saw the light of day despite four years of work and an investment of more than $100M. It is one of many computer system developments that went out of control and finally were discarded without ever having been placed in service. One sees news reports of software disasters of similar magnitude a few times each year. It is difficult to obtain solid facts about system development failures because no one wants to accept the blame, especially when lawsuits are pending. This paper suffers from a shortage of facts and an oversimplistic recommendation that better ethics are all that are needed to solve the problem. (It seems likely that the ethics and management problems simply delayed recognition of the inevitable.) Nevertheless, it provides a sobering view of how badly things can go wrong.

1.9.5 Nancy G. Leveson and Clark S. Turner. An investigation of the Therac-25 accidents. *Computer 26*, 7 (July 1993), pages 18–41. (Reprinted in reading *1.3.6*.)

This is another sobering view of how badly things can go wrong. In this case, the software controller for a high-energy medical device was inadequately designed; the device was placed in service, and lethal injuries ensued. This paper manages to inquire quite deeply into the source of the problems. Unfortunately, similar mistakes have been made since; see, for example, United States Nuclear Regulatory Commission Information Notice 2001–8s1 (June 2001), which describes radiation therapy overexposures in Panama.

1.9.6 Joe Morgenstern. City perils: The fifty-nine-story crisis. *The New Yorker 71*, 14 (May 29, 1995), pages 45–53.

This article discusses how an engineer responded to the realization that a skyscraper he had designed was in danger of collapsing in a hurricane.

1.9.7 Eric S. Raymond. The cathedral and the bazaar. In *The Cathedral and The Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, pages 19–64. O'Reilly Media Inc., 2001. ISBN: 978–0596001087, 241 pages.

The book is based on a white paper of the same title that compares two styles of software development: the Cathedral model, which is used mostly by commercial software companies and some open-source projects such as the BSD operating system; and the Bazaar model, which is exemplified by development of the GNU/Linux operating system. The work argues that the Bazaar model leads to better software because the openness and independence of Bazaar allow anyone to become a participant and to look at anything in the system that seems of interest: "Given enough eyeballs, all bugs are shallow".

1.9.8 Philip M Boffey. Investigators agree N.Y. blackout of 1977 could have been avoided. *Science 201*, 4360 (September 15, 1978), pages 994–996.

This is a fascinating description of how the electrical generation and distribution system of New York's Consolidated Edison fell apart when two supposedly tolerable faults occurred in close succession, recovery mechanisms did not work as expected, attempts to recover manually got bogged down by the system's complexity, and finally things cascaded out of control.

2 Elements of Computer System Organization

To learn more about the basic abstractions of memory and interpreters, the book *Computer Architecture* by Patterson and Hennessy (reading 1.1.1) is one of the best sources. Further information about the third basic abstraction, communication links, can be found in readings section 7.

2.1 Naming Systems

2.1.1 Bruce [G.] Lindsay. Object naming and catalog management for a distributed database manager. *Proceedings of the Second International Conference on Distributed Computing Systems*, Paris, France (April 1981), pages 31–40. Also IBM San Jose Research Laboratory Technical Report RJ2914 (August 1980). 17 pages.

This paper, a tutorial treatment of names as used in database systems, begins with a better-than-average statement of requirements and then demonstrates how those requirements were met in the R* distributed database management system.

2.1.2 Yogen K. Dalal and Robert S. Printis. 48-bit absolute Internet and Ethernet host numbers. *Proceedings of the Seventh Data Communications Symposium*,

Mexico City, Mexico, (October 1981), pages 240–245. Also Xerox Office Products Division Technical Report OPD–T8101 (July 1981), 14 pages.

This paper describes how hardware addresses are handled in the Ethernet local area network.

2.1.3 Theodor Holm Nelson. *Literary Machines, Ed. 87.1*. Project Xanadu, San Antonio, Texas, 1987. ISBN: 0–89347–056–2 (paperback). Various pagings.

Project Xanadu is an ambitious vision of a future in which books are replaced by information organized in the form of a naming network, in the form that today is called "hypertext". The book, being somewhat non-linear, is a primitive example of what Nelson advocates.

2.2 The UNIX® System

The following readings and the book by Marshall McKusick et al., reading *1.3.4*, are excellent sources on the UNIX system to follow up the case study in Section 2.5. A good, compact summary of its main features can be found in Tanenbaum's operating systems book [reading 1.2.1], which also covers Linux.

2.2.1 Dennis M. Ritchie and Ken [L.] Thompson. The UNIX time-sharing system. *Bell System Technical Journal 57*, 6, part 2 (1978), pages 1905–1930.

This paper describes an influential operating system with very low-key, but carefully chosen and hard-to-discover, objectives. The system provides a hierarchical catalog structure and succeeds in keeping naming completely distinct from file management. An earlier version of this paper appeared in the *Communications of the ACM 17*, 7 (July, 1974), pages 365–375, after being presented at the *Fourth ACM Symposium on Operating Systems Principles*. The UNIX system evolved rapidly between 1973 and 1978, so the *BSTJ* version, though harder to find, contains significant additions, both in insight and in technical content.

2.2.2 John Lions. *Lions' Commentary on UNIX 6th Edition with Source Code*. Peer-to-peer communications, 1977. ISBN: 978–1–57398–013–7, 254 pages.

This book contains the source code for UNIX Version 6, with comments to explain how it works. Although Version 6 is old, the book remains an excellent starting point for understanding how the system works from the inside, because both the source code and the comments are short and succinct. For decades, this book was part of the underground literature from which designers learned about the UNIX system, but now it is available to the public.

3 The Design of Naming Schemes

Almost any system has a naming plan, and many of the interesting naming plans can be found in papers that describe a larger system. Any reader interested in naming should study the Domain Name System, reading *4.3*, and the topic of Section 4.4.

3.1 Addressing Architectures

Several early sources still contain some of the most accessible explanations of designs that incorporate advanced naming features directly in hardware.

3.1.1 Jack B. Dennis. Segmentation and the design of multiprogrammed computer systems. *Journal of the ACM 12*, 4 (October 1965), pages 589–602.

This is the original paper outlining the advantages of providing naming support in hardware architecture.

3.1.2 R[obert] S. Fabry. Capability-based addressing. *Communications of the ACM 17*, 7 (July 1974), pages 403–412.

This is the first comprehensive treatment of capabilities, a mechanism introduced to enforce modularity but actually more of a naming feature.

3.1.3 Elliott I. Organick. *Computer System Organization, The B5700/B6700 Series*. Academic Press, 1973. ISBN: 0–12–528250–8. 132 pages.

The Burroughs Descriptor system explained in this book is apparently the only example of a hardware-supported naming system actually implemented before the advent of microprogramming.

3.1.4 Elliott I. Organick. *The Multics System: an Examination of its Structure*. M.I.T. Press, Cambridge, Massachusetts, 1972. ISBN: 0–262–15012–3. 392 pages.

This book explores every detail and ramification of the extensive naming mechanisms of Multics, both in the addressing architecture and in the file system.

3.1.5 R[oger] M. Needham and A[ndrew] D. Birrell. The CAP filing system. *Proceedings of the Sixth ACM Symposium on Operating Systems Principles*, in *Operating Systems Review 11*, 5 (November 1977), pages 11–16.

The CAP file system is one of the few implemented examples of a genuine naming network.

3.2 Examples

3.2.1 Paul J. Leach, Bernard L. Stumpf, James A. Hamilton, and Paul H. Levine. UIDs as internal names in a distributed file system. In *ACM SIGACT–SIGOPS Symposium on Principles of Distributed Computing*, Ottawa, Ontario (August 18–20, 1982), pages 34–41.

The Apollo DOMAIN system supports a different model for distributed function. It provides a shared primary memory called the Single Level Store, which extends transparently across the network. It is also one of the few systems to make substantial use of unstructured unique identifiers from a compact set as object names. This paper focuses on this latter issue.

3.2.2 Rob Pike et al. Plan 9 from Bell Labs. *Computing Systems 8*, 3 (Summer 1995), pages 221–254. An earlier version by Rob Pike, Dave Presotto, Ken Thompson,

and Howard Trickey appeared in *Proceedings of the Summer 1990 UKUUG Conference* (1990), London, pages 1–9.

This paper describes a distributed operating system that takes the UNIX system idea that every resource is a file one step further by using it also for network and window system interactions. It also extends the file idea to a distributed system by defining a single file system protocol for access to all resources, whether they are local or remote. Processes can mount any remote resources into their name space, and to the user these remote resources behave just like local resources. This design makes users perceive the system as an easy-to-use time-sharing system that behaves like a single powerful computer, instead of a collection of separate computers.

3.2.3 Tim Berners–Lee et al. The World Wide Web. *Communications of the ACM 37*, 8 (August 1994), pages 76–82.

Many of the publications about the World Wide Web are available only on the Web, with a good starting point being the home page of the World Wide Web Consortium at <http://w3c.org/>.

3.2.4 Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Proceedings of the 7th WWW Conference*, Brisbane, Australia (April 1998). Also in *Computer Networks 30* (1998), pages 107–117.

This paper describes an early version of Google's search engine. It also introduces the idea of page rank to sort the results to a query in order of importance. Search is a dominant way in which users "name" Web pages.

3.2.5 Bryan Ford et al. Persistent personal names for globally connected mobile devices. *Proceedings of the Seventh USENIX Symposium on Operating Systems Design and Implementation* (November 2006), pages 233–248.

This paper describes a naming system for personal devices. Each device is a root of its own naming network and can use short, convenient names for other devices belonging to the same user or belonging to people in the user's social network. The implementation of the naming system allows devices to be disconnected from the Internet and resolve names of devices that are reachable. The first five pages lay out the basic naming plan. Later sections explain security properties and a security-based implementation, which involves material of Chapter 11 [on-line].

4 Enforcing Modularity with Clients and Services

Many systems are organized in a client/service style. A system that provides a good case study is the Network File System (see Section 4.5). The following papers provide some other examples.

4.1 Remote Procedure Call

4.1.1 Andrew D. Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems 2*, 1 (February 1984), pages 39–59.

A well-written paper that shows first, the simplicity of the basic idea, second, the complexity required to deal with real implementations, and third, the refinements needed for high effectiveness.

4.1.2 Andrew Birrell, Greg Nelson, Susan Owicki, and Edward Wobber. Network objects. *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, in *Operating Systems Review 27*, 5 (December 1993), pages 217–230.

This paper describes a programming language for distributed applications based on remote procedure calls, which hide most "distributedness" from the programmer.

4.1.3 Ann Wollrath, Roger Riggs, and Jim Waldo. A distributed object model for the Java™ system. *Computing Systems 9*, 4 (1996), pages 265–290. Originally published in *Proceedings of the Second USENIX Conference on Object-Oriented Technologies Volume 2* (1996).

This paper presents a remote procedure call system for the Java programming language. It provides a clear description of how an RPC system can be integrated with an object-oriented programming language and the new exception types RPC introduces.

4.2 Client/Service Systems

4.2.1 Daniel Swinehart, Gene McDaniel, and David [R.] Boggs. WFS: A simple shared file system for a distributed environment. *Proceedings of the Seventh ACM Symposium on Operating Systems Principles*, in *Operating Systems Review 13*, 5 (December 1979), pages 9–17.

This early version of a remote file system opens the door to the topic of distribution of function across connected cooperating computers. The authors' specific goal was to keep things simple; thus, the relationship between mechanism and goal is much clearer than in more modern, but more elaborate, systems.

4.2.2 Robert Scheifler and James Gettys. The X Window System. *ACM Transactions on Graphics 5*, 2 (April 1986), pages 79–109.

The X Window System is the window system of choice on practically every engineering workstation in the world. It provides a good example of using the client/service model to achieve modularity. One of the main contributions of the X Window System is that it remedied a defect that had crept into the UNIX system when displays replaced typewriters: the display and keyboard were the only hardware-dependent parts of the UNIX application programming interface. The X Window System allowed display-oriented UNIX applications to be completely

independent of the underlying hardware. In addition, the X Window System interposes an efficient network connection between the application and the display, allowing configuration flexibility in a distributed system.

4.2.3 John H. Howard et al. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems 6*, 1 (February 1988), pages 51–81.

This paper describes experience with a prototype of the Andrew network file system for a campus network and shows how the experience motivated changes in the design. The Andrew file system had strong influence on version 4 of NFS.

4.3 Domain Name System (DNS)

The domain name system is one of the most interesting distributed systems in operation. It is not only a building block in many distributed applications, but is itself an interesting case study, offering many insights for anyone wanting to build a distributed system or a naming system.

4.3.1 Paul V. Mockapetris and Kevin J. Dunlap. Development of the Domain Name System. *Proceedings of the SIGCOMM 1988 Symposium*, pages 123–133. Also published in *ACM Computer Communications Review 18*, 4 (August 1988), pages 123–133, and republished in *ACM Computer Communications Review 25*, 1 (January 1995), pages 112–122.

4.3.2 Paul [V.] Mockapetris. Domain names—Concepts and facilities. *Request for Comments RFC 1034*, Internet Engineering Task Force (November 1987).

4.3.3 Paul [V.] Mockapetris. Domain names—Implementation and specification. *Request for Comments RFC 1035*, Internet Engineering Task Force (November 1987).

These three documents explain the DNS protocol.

4.3.4 Paul Vixie. DNS Complexity. *ACM Queue 5*, 3 (April 2007), pages 24–29.

This paper uncovers many of the complexities of how DNS, described in the case study in Section 4.4, works in practice. The protocol for DNS is simple, and no complete, precise specification of the system exists. The author argues that the current descriptive specification of DNS is an advantage because it allows various implementations to evolve to include new features as needed. The paper describes many of these features and shows that DNS is one of the most interesting distributed systems in use today.

5 Enforcing Modularity with Virtualization

5.1 Kernels

The readings on the UNIX system (see readings section 2.2) are a good starting point for studying kernels.

5.1.1 Per Brinch Hansen. The nucleus of a multiprogramming system. *Communications of the ACM 13*, 4 (April 1970), pages 238–241.

The RC–4000 was the first, and may still be the best explained, system to use messages as the primary concurrency coordination mechanism. It is also what would today be called a microkernel design.

5.1.2 M. Frans Kaashoek et al. Application performance and flexibility on exokernel systems. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, in *Operating Systems Review 31*, 5 (December 1997), pages 52–65.

The exokernel provides an extreme version of separation of policy from mechanism, sacrificing abstraction to expose (within protection constraints) all possible aspects of the physical environment to the next higher layer, giving that higher layer maximum flexibility in creating abstractions for its preferred programming environment, or tailored to its preferred application.

5.2 Type Extension as a Modularity Enforcement Tool

5.2.1 Butler W. Lampson and Howard E. Sturgis. Reflections on an operating system design. *Communications of the ACM 19*, 5 (May 1976), pages 251–265.

An operating system named CAL, designed at the University of California at Berkeley, appears to be the first system to make explicit use of types in the interface to the operating system. In addition to introducing this idea, Lampson and Sturgis also give good insight into the pros and cons of various design decisions. Documented late, the system was actually implemented in 1969.

5.2.2 Michael D. Schroeder, David D. Clark, and Jerome H. Saltzer. The Multics kernel design project. *Proceedings of the Sixth ACM Symposium on Operating Systems Principles*, in *Operating Systems Review 11*, 5 (November 1977), pages 43–56.

This paper addresses a wide range of issues encountered in applying type extension (as well as microkernel thinking, though it wasn't called that at the time) to Multics in order to simplify its internal organization and reduce the size of its trusted base. Many of these ideas were explored in even more depth in Philippe Janson's Ph.D. Thesis, *Using Type Extension to Organize Virtual Memory Mechanisms*, M.I.T. Department of Electrical Engineering and Computer Science, August 1976. That thesis is also available as M.I.T. Laboratory for Computer Science Technical Report TR–167, September 1976.

5.2.3 Galen C. Hunt and James R. Larus. Singularity: Rethinking the software stack. *Operating Systems Review 41*, 2 (April 2007), pages 37–49.

Singularity is an operating system that uses type-safe languages to enforce modularity between different software modules, instead of relying on virtual-memory hardware. The kernel and all applications are written in a strongly typed programming language with automatic garbage collection. They run in a single address space and are isolated from each other by the language runtime. They

can interact with each other only through communication channels that carry type-checked messages.

5.3 Virtual Processors: Threads

5.3.1 Andrew D. Birrell. *An introduction to programming with threads*. Digital Equipment Corporation Systems Research Center Technical Report #35, January 1989. 33 pages. (Also appears as Chapter 4 of Greg Nelson, editor, *Systems Programming with Modula-3*, Prentice-Hall, 1991, pages 88–118.) A version for the C# programming language appeared as Microsoft Research Report MSR-TR-2005-68.

This is an excellent tutorial, explaining the fundamental issues clearly and going on to show the subtleties involved in exploiting threads correctly and effectively.

5.3.2 Thomas E. Anderson et al. Scheduler activations: Effective kernel support for the user-level management of parallelism. *ACM Transactions on Computer Systems 10*, 1 (February 1992), pages 53–79. Originally published in *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, in *Operating Systems Review 25*, 5 (December 1991), pages 95–109.

The distinction between user threads and kernel threads comes to the fore in this paper, which offers a way of getting the advantages of both by having the right kind of user/kernel thread interface. The paper also revisits the idea of a virtual processor, but in a multiprocessor context.

5.3.3 David D. Clark. The structuring of systems using upcalls. *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, in *Operating Systems Review 19*, 5 (December 1985), pages 171–180.

Attempts to impose modular structure by strict layering sometimes manage to overlook the essence of what structure is most appropriate. This paper describes a rather different intermodule organization that seems to be especially effective when dealing with network implementations.

5.3.4 Jerome H. Saltzer. *Traffic Control in a Multiplexed Computer System*. Ph.D. Thesis, Massachusetts Institute of Technology, Department of Electrical Engineering, June 1966. Also available as Project MAC Technical Report TR-30, 1966.

This work describes what is probably the first systematic virtual processor design and thread package, the multiprocessor multiplexing scheme used in the Multics system. It defines the coordination primitives `BLOCK` and `WAKEUP`, which are examples of binary semaphores assigned one per thread.

5.3.5 Rob Pike et al. Processor sleep and wakeup on a shared-memory multiprocessor. *Proceedings of the EurOpen Conference* (1991), pages 161–166.

This well-written paper does an excellent job of explaining how difficult it is to get preemptive multiplexing, handling interrupts, and implementing coordination primitives correct on shared-memory multiprocessor.

5.4 Virtual Memory

There are few examples of papers that describe a simple, clean design. The older papers (some can be found in reading section 3.1) get bogged down in technology constraints; the more recent papers (some of the them can be found in reading section 6.1 on multilevel memory management) often get bogged down in performance optimizations. The case study on the evolution of enforcing modularity with the Intel x86 (see Section 5.7 of Chapter 5) describes virtual memory support in the most widely used processor and shows how it evolved over time.

5.4.1 A[ndre] Bensoussan, C[harles] T. Clingen, and R[obert] C. Daley. The Multics virtual memory: Concepts and design. *Communications of the ACM 15*, 5 (May 1972), pages 308–318.

This is a good description of a system that pioneered the use of high-powered addressing architectures to support a sophisticated virtual memory system, including memory-mapped files. The design was constrained and shaped by the available hardware technology (0.3 MIPS processor with an 18–bit address space), but the paper is a classic and easy to read.

5.5 Coordination

Every modern textbook covers the topic of coordination but typically brushes past the subtleties and also typically gives the various mechanisms more emphasis than they deserve. These readings either explain the issues much more carefully or extend the basic concepts in various directions.

5.5.1 E[dsger] W. Dijkstra. Co-operating sequential processes. In F. Genuys, editor, *Programming Languages*, NATO Advanced Study Institute, Villard-de-Lans, 1966. Academic Press, 1968, pages 43–112.

This paper introduces semaphores, the synchronizing primitive most often used in academic exercises, and is notable for its very careful, step-by-step development of the requirements for mutual exclusion and its implementation. Many modern treatments ignore the subtleties discussed here as if they were obvious. They aren't, and if you want to understand synchronization you should read this paper.

5.5.2 E[dsger] W. Dijkstra. Solution of a problem in concurrent programming control. *Communications of the ACM 8*, 9 (September 1965), page 569.

In this very brief paper, Dijkstra first reports Dekker's observation that multiprocessor locks can be implemented entirely in software, relying on the hardware to guarantee only that read and write operations have before-or-after atomicity.

5.5.3 Leslie Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems 5*, 1 (February 1987), pages 1–11.

This paper presents a fast version of a software-only implementation of locks and gives an argument as to why this version is optimal.

5.5.4 David P. Reed and Rajendra K. Kanodia. Synchronization with eventcounts and sequencers. *Communications of the ACM 22*, 2 (February 1979), pages 115–123.

This paper introduces an extremely simple coordination system that uses less powerful primitives for sequencing than for mutual exclusion; a consequence is simple correctness arguments.

5.5.5 Butler W. Lampson and David D. Redell. Experience with processes and monitors in Mesa. *Communications of the ACM 23*, 2 (February 1980), pages 105–117.

This is a nice discussion of the pitfalls involved in integrating concurrent activity coordination into a programming language.

5.5.6 Stefan Savage et al. Eraser: A dynamic data race detector for multi-threaded programs. *ACM Transactions on Computer Systems 15*, 4 (November 1997), pages 391–411. Also in the *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles* (October 1997).

This paper describes an interesting strategy for locating certain classes of locking mistakes: instrument the program by patching its binary data references; then watch those data references to see if the program violates the locking protocol.

5.5.7 Paul E. McKenney et al. Read-copy update. *Proceedings of the Ottawa Linux Symposium*, 2002, pages 338–367.

This paper observes that locks can be an expensive mechanism for before-or-after atomicity for data structures that are mostly read and infrequently modified. The authors propose a new technique, read-copy update (RCU), which improves performance and scalability. The Linux kernel uses this mechanism for many of its data structures that processors mostly read.

5.5.8 Maurice Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems* 11, 1 (January 1991), pages 124–149.

This paper introduces the goal of wait-free synchronization, now often called non-blocking coordination, and gives non-blocking, concurrent implementations of common data structures such as sets, lists, and queues.

5.5.9 Timothy L. Harris. A pragmatic implementation of non-blocking linked lists. *Proceedings of the fifteenth International Symposium on Distributed Computing*, (October 2001), pages 300–314.

This paper describes a practical implementation of a linked list in which threads can insert concurrently without blocking.

See also reading *5.1.1*, by Brinch Hansen, which uses messages as a coordination technique, and reading *5.3.1* by Birrell, which describes a complete set of coordination primitives for programming with threads.

5.6 Virtualization

5.6.1 Robert J. Creasy. The origin of the VM/370 time-sharing system. *IBM Journal of Research and Development 25*, 5 (1981), pages 483–490.

This paper is an insightful retrospective about a mid-1960s project to virtualize the IBM 360 computer architecture and the development that led to VM/370, which in the 1970s became a popular virtual machine system. At the time, the unusual feature of VM/370 was its creation of a strict, by-the-book, hardware virtual machine, thus providing the ability to run any system/370 program in a controlled environment. Because it was a pioneer project, the author explained things particularly well, thus providing a good introduction to the concepts and problems in implementing virtual machines.

5.6.2 Edouard Bugnion et al. Disco: running commodity operating systems on scalable multiprocessors. *ACM Transactions on Computer Systems 15*, 14 (November 1997), pages 412–447.

This paper brought virtual machines back as a mainstream way of building systems.

5.6.3 Carl Waldspurger. Memory resource management in VMware ESX server. *Proceedings of the Fifth USENIX Symposium on Operating Systems Design and Implementation* (December 2002), pages 181–194.

This well-written paper introduces a nice trick (a balloon driver) to decide how much physical memory to give to guest operating systems.

5.6.4 Keith Adams and Ole Agesen. A comparison of software and hardware techniques for x86 virtualization. *Proceedings of the Twelfth Symposium on Architectural Support for Programming Languages and Operating Systems* (October 2006). ISBN: 1–59593–451–0. Also in *Operating Systems Review 40*, 5 (December 2006), pages 2–13.

This paper describes how one can virtualize the Intel x86 instruction set to build a high-performance virtual machine. It compares two implementation strategies: one that uses software techniques, such as binary rewriting, to virtualize the instruction set, and one that uses recent hardware additions to the x86 processor to make virtualizing easier. The comparison provides insights about implementing modern virtual machines and operating system support in modern x86 processors.

Also see the paper on the secure virtual machine monitor for the VAX machine, reading *11.3.5*.

6 Performance

6.1 Multilevel Memory Management

An excellent discussion of memory hierarchies, with special attention paid to the design space for caches, can be found in Chapter 5 of the book by Patterson and Hennessy, reading *1.1.1*. A lighter-weight treatment focused more on virtual memory, and including a discussion of stack algorithms, can be found in Chapter 3 of Tanenbaum's computer systems book, reading *1.2.1*.

6.1.1 R[obert] A. Frieburghouse. Register allocation via usage counts. *Communications of the ACM 17*, 11 (November 1974), pages 638–642.

This paper shows that compiler code generators must do multilevel memory management and that they have the same problems as do caches and paging systems.

6.1.2 R[ichard] L. Mattson, J. Gecsei, D[onald] R. Slutz, and I[rving] L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal 9*, 2 (1970), pages 78–117.

The original reference on stack algorithms and their analysis, this paper is well written and presents considerably more in-depth observations than the brief summaries that appear in modern textbooks.

6.1.3 Richard Rashid et al. Machine-independent virtual memory management for paged uniprocessor and multiprocessor architectures. *IEEE Transactions on Computers 37*, 8 (August 1988), pages 896–908. Originally published in *Proceedings of the Second International Conference on Architectural Support for Programming Languages and Operating Systems* (November 1987), pages 31–39.

This paper describes a design for a sophisticated virtual memory system that has been adopted by several operating systems, including several BSD operating systems and Apple's OS X. The system supports large, sparse virtual address spaces, copy-on-write copying of pages, and memory-mapped files.

6.1.4 Ted Kaehler and Glenn Krasner. LOOM: Large object-oriented memory for Smalltalk–80 systems. In Glenn Krasner, editor, *Smalltalk–80: Bits of History, Words of Advice*. Addison-Wesley, 1983, pages 251–271. ISBN: 0–201–11669–3.

This paper describes the memory-management system used in Smalltalk, an interactive programming system for desktop computers. A coherent virtual memory language support system provides for lots of small objects while taking into account address space allocation, multilevel memory management, and naming in an integrated way.

The paper on the Woodstock File System, by Swinehart et al., reading *4.2.1*, describes a file system that is organized as a multilevel memory management system. Also see reading *10.1.8* for an interesting application (shared virtual memory) using multilevel memory management.

6.2 Remote Procedure Call

6.2.1 Michael D. Schroeder and Michael Burrows. Performance of Firefly RPC. *ACM Transactions on Computer Systems 8*, 1 (February 1990), pages 1–17. Originally published in *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, in *Operating Systems Review 23*, 5 (December 1989), pages 102–113.

As a complement to the abstract discussion of remote procedure call in reading *4.1.1*, this paper gives a concrete, blow-by-blow accounting of the steps required in a particular implementation and then compares this accounting with overall time measurements. In addition to providing insight into the intrinsic costs of remote procedures, this work demonstrates that it is possible to do bottom-up performance analysis that correlates well with top-down measurements.

6.2.2 Brian N. Bershad, Thomas E. Anderson, Edward D. Lazowska, and Henry M. Levy. Lightweight remote procedure call. *ACM Transactions on Computer Systems 8*, 1 (February 1990), pages 37–55. Originally published in *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, in *Operating Systems Review 23*, 5 (December 1989), pages 102–113.

6.2.3 Jochen Liedtke. Improving IPC by kernel design. *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, in *Operating Systems Review 27*, 5 (December 1993), pages 175–187.

These two papers develop techniques to allow local kernel-based client/service modularity to look just like remote client/service modularity to the application designer, while at the same time capturing the performance advantage that can come from being local.

6.3 Storage

6.3.1 Chris Ruemmler and John Wilkes. An introduction to disk drive modeling. *Computer 27*, 3 (March 1994), pages 17–28.

This paper is really two papers in one. The first five pages provide a wonderfully accessible explanation of how disk drives and controllers actually work. The rest of the paper, of interest primarily to performance modeling specialists, explores the problem of accurately simulating a complex disk drive, with measurement data to show the size of errors that arise from various modeling simplifications (or oversimplifications).

6.3.2 Marshall K. McKusick, William N. Joy, Samuel J. Leffler, and Robert S. Fabry. A fast file system for UNIX. *ACM Transactions on Computer Systems 2*, 3 (August 1984), pages 181–197.

The "fast file system" nicely demonstrates the trade-offs between performance and complexity in adding several well-known performance enhancement techniques, such as multiple block sizes and sector allocation based on adjacency, to a file system that was originally designed as the epitome of simplicity.

6.3.3 Gregory R. Ganger and Yale N. Patt. Metadata update performance in file systems. *Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation* (November 1994), pages 49–60.

This paper is an application to file systems of some recovery and consistency concepts originally developed for database systems. It describes a few simple rules (e.g., an inode should be written to the disk after writing the disk blocks to which it points) that allow a system designer to implement a file system that is high performance and always keeps its on-disk data structures consistent in the presence of failures. As applications perform file operations, the rules create dependencies between data blocks in the write-behind cache. A disk driver that knows about these dependencies can write the cached blocks to disk in an order that maintains consistency of on-disk data structures despite system crashes.

6.3.4 Andrew Birrell et al. A design for high-performance flash disks. *ACM Operating Systems Review 41*, 2 (April 2007), pages 88–93. (Also appeared as Microsoft Corporation technical report TR-2005–176.)

Flash (non-volatile) electronic memory organized to appear as a disk has emerged as a more expensive but very low-latency alternative to magnetic disks for durable storage. This short paper describes, in an easy-to-understand way, the challenges associated with building a high-performance file system using flash disks and proposes a design to address the challenges. This paper is a good start for readers who want to explore flash-based storage systems.

6.4 Other Performance-Related Topics

6.4.1 Sharon E. Perl and Richard L. Sites. Studies of Windows NT performance using dynamic execution traces. *Proceedings of the Second USENIX Symposium on Operating Systems Design and Implementation* (October 1996). Also in *Operating System Review 30*, SI (October 1996), pages 169–184.

This paper shows by example that any performance issue in computer systems can be explained. The authors created a tool to collect complete traces of instructions executed by the Windows NT operating system and applications. The authors conclude that pin bandwidth limits the achievable execution speed of applications and that locks inside the operating system can limit applications to scale to more than a moderate number of processors. The paper also discusses the impact of cache-coherence hardware (see Chapter 10 [on-line]) on application performance. All of these issues are increasingly important for multiprocessors on a single chip.

6.4.2 Jeffrey C. Mogul and K. K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. *Transactions on Computer Systems 15*, 3 (August 1997), pages 217–252.

This paper introduces the problem of receive livelock (described in Sidebar 6.7) and presents a solution. Receive livelock is a possible undesirable situation when a

system is temporarily overloaded. It can arise if the server spends too much of its time saying "I'm too busy" and as a result has no time left to serve any of the requests.

6.4.3 Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. *Proceedings of the Sixth USENIX Symposium on Operating Systems Design and Implementation* (December 2004), pages 137–150. Also in *Communications of the ACM 51*, 1 (January 2008), pages 1–10.

This paper is a case study of aggregating arrays (reaching into the thousands) of computers to perform parallel computations on large data sets (e.g., all the pages of the Web). It uses a model that applies when a composition of two serial functions (Map and Reduce) has no side-effects on the data sets. The charm of MapReduce is that for computations that fit the model, the runtime uses concurrency but hides it completely from the programmer. The runtime partitions the input data set, executes the functions in parallel on different parts of the data set, and handles the failures of individual computers.

7 The Network as a System and as a System Component

Proceedings of the IEEE 66, 11 (November 1978), a special issue of that journal devoted to packet switching, contains several papers mentioned under various topics here. Collectively, they provide an extensive early bibliography on computer communications.

7.1 Networks

The book by Perlman on bridges and routers, reading *1.2.5*, explains how the network layer really works.

7.1.1 David D. Clark, Kenneth T. Pogran, and David P. Reed. An introduction to local area networks. *Proceedings of the IEEE 66*, 11 (November 1978), pages 1497–1517.

This basic tutorial on local area network communications characterizes the various modular components of a local area network, both interface and protocols, gives specific examples, and explains how local area networks relate to larger, interconnected networks. The specific examples are now out of date, but the rest of the material is timeless.

7.1.2 Robert M. Metcalfe and David R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM 19*, 7 (July 1976), pages 395–404.

This paper provides the design of what has proven to be the most popular local area network technology.

7.2 Protocols

7.2.1 Louis Pouzin and Hubert Zimmerman. A tutorial on protocols. *Proceedings of the IEEE 66*, 11 (November 1978), pages 1346–1370.

This paper is well written and provides perspective along with the details. The fact that it was written a long time ago turns out to be its major appeal. Because networks were not widely understood at the time, it was necessary to fully explain all of the assumptions and offer extensive analogies. This paper does an excellent job of both, and as a consequence it provides a useful complement to modern texts. While reading this paper, anyone familiar with current network technology will frequently exclaim, "So that's why the Internet works that way."

7.2.2 Vinton G. Cerf and Peter T. Kirstein. Issues in packet-network interconnection. *Proceedings of the IEEE 66*, 11 (November 1978), pages 1386–1408.

At the time this paper was written, an emerging problem was the interconnection of independently administered data communication networks. This paper explores the issues in both breadth and depth, a combination that more recent papers do not provide.

7.2.3 David D. Clark and David L. Tennenhouse. Architectural considerations for a new generation of protocols. *ACM SIGCOMM '91 Conference: Communications Architectures and Protocols*, in *Computer Communication Review 20*, 4 (September 1990), pages 200–208.

This paper captures 20 years of experience in protocol design and implementation and lays out the requirements for the next few rounds of protocol design. The basic observation is that the performance requirements of future high-speed networks and applications will require that the layers used for protocol description not constrain implementations to be similarly layered. This paper is required reading for anyone who is developing a new protocol or protocol suite.

7.2.4 Danny Cohen. On holy wars and a plea for peace. *IEEE Computer 14*, 10 (October 1981), pages 48–54.

This is an entertaining discussion of big-endian and little-endian arguments in protocol design.

7.2.5 Danny Cohen. Flow control for real-time communication. *Computer Communication Review 10*, 1–2 (January/April 1980), pages 41–47.

This brief item is the source of the "servant's dilemma", a parable that provides helpful insight into why flow control decisions must involve the application.

7.2.6 Geoff Huston. Anatomy: A look inside network address translators. *The Internet Protocol Journal* 7, 3 (September 2004), pages 2–32.

Network address translators (NATs) break down the universal connectivity property of the Internet: when NATs are in use, one can no longer assume that every computer in the Internet can communicate with every other computer in the Internet. This paper discusses the motivation for NATs, how they work, and in what ways they create havoc for some Internet applications.

7.2.7 Van Jacobson. Congestion avoidance and control. *Proceedings of the Symposium on Communications Architectures and Protocols* (SIGCOMM '88), pages 314–329. Also in *Computer Communication Review 18*, 4 (August 1988).

Sidebar 7.9 gives a simplified description of the congestion avoidance and control mechanism of TCP, the most commonly used transport protocol in the Internet. This paper explains those mechanisms in full detail. They are surprisingly simple but have proven to be effective.

7.2.8 Jordan Ritter. Why Gnutella can't scale. No, really. Unpublished grey literature. <http://www.darkridge.com/~jpr5/doc/gnutella.html>

This paper offers a simple performance model to explain why the Gnutella protocol (see problem set *20*) cannot support large networks of Gnutella peers. The problem is incommensurate scaling of its bandwidth requirements.

7.2.9 David B. Johnson. Scalable support for transparent mobile host internetworking. *Wireless Networks 1*, 3 (1995), pages 311–321.

Addressing a laptop computer that is connected to a network by a radio link and that can move from place to place without disrupting network connections can be a challenge. This paper proposes a systematic approach based on maintaining a tunnel between the laptop computer's current location and an agent located at its usual home location. Variations of this paper (based on the author's 1993 Ph.D. Thesis at Carnegie-Mellon University and available as CMU Computer Science Technical Report CS–93–128) have appeared in several 1993 and 1994 workshops and conferences, as well as in the book *Mobile Computing*, Tomasz Imielinski and Henry F. Korth, editors, Kluwer Academic Publishers, c. 1996. ISBN: 079239697–9.

One popular protocol, remote procedure call, is covered in depth in reading *4.1.1* by Birrell and Nelson, as well as Section 10.3 of Tanenbaum's *Modern Operating Systems,* reading *1.2.1.*

7.3 Organization for Communication

7.3.1 Leonard Kleinrock. Principles and lessons in packet communications. *Proceedings of the IEEE 66*, 11 (November 1978), pages 1320–1329.

7.3.2 Lawrence G. Roberts. The evolution of packet switching. *Proceedings of the IEEE 66*, 11 (November 1978), pages 1307–1313.

These two papers discuss experience with the ARPANET. Anyone faced with the need to design a network should look over these two papers, which focus on lessons learned and the sources of surprise.

7.3.3 J[erome] H. Saltzer, D[avid]. P. Reed, and D[avid]. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems 2*, 4 (November 1984), pages 277–288. An earlier version appears in the *Proceedings*

of the Second International Conference on Distributed Computing Systems (April 1981), pages 504–512.

This paper proposes a design rationale for deciding which functions belong in which layers of a layered network implementation. It is one of the few papers available that provides a system design principle.

7.3.4 Leonard Kleinrock. The latency/bandwidth trade-off in gigabit networks. *IEEE Communications Magazine 30*, 4 (April 1992), pages 36–40.

Technology has made gigabit/second data rates economically feasible over long distances. But long distances and high data rates conspire to change some fundamental properties of a packet network—latency becomes the dominant factor that limits applications. This paper provides a very good explanation of the problem.

7.4 Practical Aspects

For the complete word on the Internet protocols, check out the following series of books.

7.4.1 W. Richard Stevens. *TCP/IP illustrated*. Addison-Wesley; v. 1, 1994, ISBN: 0–201–63346–9, 576 pages; v. 2 (with co-author Gary R. Wright,) 1995, ISBN: 0–201–63354–x, 1174 pages.; v. 3, 1996, ISBN: 0–201–63495–3, 328 pages. *Volume 1: The Protocols. Volume 2: The Implementation. Volume 3: TCP for Transactions, HTTP, NNTP, and the UNIX® Domain Protocols.*

These three volumes will tell you more than you wanted to know about how TCP/IP is implemented, using the network implementation of the Berkeley System Distribution for reference. The word "illustrated" refers more to computer printouts—listings of packet traces and programs—than to diagrams. If you want to know how some aspect of the Internet protocol suite is actually implemented, this is the place to look—though it does not often explain why particular implementation choices were made.

8 Fault Tolerance: Reliable Systems from Unreliable Components

A plan for some degree of fault tolerance shows up in many systems. For an example of fault tolerance in distributed file systems, see the paper on Coda by Kistler and Satyanarayanan, reading *10.1.2*. See also the paper on RAID by Katz et al., reading *10.2.2.*

8.1 Fault Tolerance

Chapter 3 of the book by Gray and Reuter, reading *1.1.5*, provides a bedrock text on this subject.

8.1.1 Jim [N.] Gray and Daniel P. Siewiorek. High-availability computer systems. *Computer 24*, 9 (September 1991), pages 39–48.

This is a very nice, easy-to-read overview of how high availability can be achieved.

8.1.2 Daniel P. Siewiorek. Architecture of fault-tolerant computers. *Computer 17*, 8 (August 1984), pages 9–18.

This paper provides an excellent taxonomy, as well as a good overview of several architectural approaches to designing computers that continue running even when a single hardware component fails.

8.2 Software Errors

8.2.1 Dawson Engler et al. Bugs as deviant behavior: A general approach to inferring errors in systems code. *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, 2001, in *Operating Systems Review 35*, 5 (December 2001), pages 57–72.

This paper describes a method for finding possible programming faults in large systems by looking for inconsistencies. For example, if in most cases an invocation of a certain function is preceded by disabling interrupts but in a few cases it is not, there is a good chance that a programming fault is present. The paper uses this insight to create a tool for finding potential faults in large systems.

8.2.2 Michael M. Swift et al. Recovering device drivers. *Proceedings of the Sixth Symposium on Operating Systems Design and Implementation* (December 2004), pages 1–16.

This paper observes that software faults in device drivers often lead to fatal errors that cause operating systems to fail and thus require a reboot. It then describes how virtual memory techniques can be used to enforce modularity between device drivers and the rest of the operating system kernel, and how the operating system can recover device drivers when they fail, reducing the number of reboots.

8.3 Disk Failures

8.3.1 Bianca Schroeder and Garth A. Gibson. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? *Proceedings of the Fifth USENIX Conference on File and Storage Technologies* (2007), pages 1–16.

As explained in Section 8.2, it is not uncommon that data sheets for disk drives specify MTTFs of one hundred years or more, many times the actual observed lifetimes of those drives in the field. This paper looks at disk replacement data for 100,000 disk drives and discusses what MTTF means for those disk drives.

8.3.2 Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz Andre Barroso. Failure trends in a large disk drive population. *Proceedings of the Fifth USENIX Conference on File and Storage Technologies* (2007), pages 17–28.

Recently, outfits such as Google have deployed large enough numbers of off-the-shelf disk drives for a long enough time that they can make their own evaluations of disk drive failure rates and lifetimes, for comparison with the a priori reliability models of the disk vendors. This paper reports data collected from such observations. It analyzes the correlation between failures and several parameters that are generally believed to impact the lifetime of disk and finds some surprises. For example, it reports that temperature is less correlated with disk drive failure than was previously reported, as long as the temperature is within a certain range and stable.

9 Atomicity: All-or-Nothing and Before-or-After

9.1 Atomicity, Coordination, and Recovery

The best source on this topic is reading *1.1.5*, but Gray and Reuter's thousand-page book can be a bit overwhelming.

9.1.1 Warren A. Montgomery. *Robust Concurrency Control for a Distributed Information System*. Ph.D. Thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, December 1978. Also available as M.I.T. Laboratory for Computer Science Technical Report TR–207, January 1979. 197 pages.

This work describes alternative strategies that maximize concurrent activity while achieving atomicity: maintaining multiple values for some variables, atomic broadcast of messages to achieve proper sequence.

9.1.2 D. B. Lomet. Process structuring, synchronization, and recovery using atomic actions. *Proceedings of an ACM Conference on Language Design for Reliable Software* (March 1977), pages 128–137. Published as *ACM SIGPLAN Notices 12*, 3 (March 1977); *Operating Systems Review 11*, 2 (April 1977); and *Software Engineering Notes 2*, 2 (March 1977).

This is one of the first attempts to link atomicity to both recovery and coordination. It is written from a language, rather than an implementation, perspective.

9.2 Databases

9.2.1 Jim [N.] Gray et al. The recovery manager of the system R database manager. *ACM Computing Surveys 13*, 2 (June 1981), pages 223–242.

This paper is a case study of a sophisticated, real, high-performance logging and locking system. It is one of the most interesting case studies of its type because it shows the number of different, interacting mechanisms needed to construct a system that performs well.

9.2.2 C. Mohan et al. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Transactions on Database Systems 17*, 1 (1992), pages 94–162.

This paper describes all the intricate design details of a fully featured, commercial-quality database transaction system that uses write-ahead logging.

9.2.3 C. Mohan, Bruce Lindsey, and Ron Obermarck. transaction management in the R* distributed database management system. *ACM Transactions on Database Systems (TODS) 11*, 4 (December 1986), pages 378–396.

This paper deals with transaction management for distributed databases, and introduces two new protocols (Presumed Abort and Presumed Commit) that optimize two-phase commit (see Section 9.6), resulting in fewer messages and log writes. Presumed Abort is optimized for transactions that perform only read operations, and Presumed Commit is optimized for transactions with updates that involve several distributed databases.

9.2.4 Tom Barclay, Jim Gray, and Don Slutz. Microsoft TerraServer: A spatial data warehouse. *Microsoft Technical Report MS-TR-99–29*. June 1999.

The authors report on building a popular Web site that hosts aerial, satellite, and topographic images of Earth using a off-the-shelf components, including a standard database system for storing the terabytes of data.

9.2.5 Ben Vandiver et al. Tolerating byzantine faults in transaction processing systems using commit barrier scheduling. *Proceedings of the Twenty-first ACM Symposium on Operating Systems Principles*, in *Operating Systems Review 41*, 6 (December 2005), pages 59–79.

This paper describes a replication scheme for handling Byzantine faults in database systems. It issues queries and updates to multiple replicas of unmodified, off-the-shelf database systems, and it compares their responses, thus creating a single database that is Byzantine fault tolerant (see Section 8.6 for the definition of Byzantine).

9.3 Atomicity-Related Topics

9.3.1 Mendel Rosenblum and John K. Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems 10*, 1 (February 1992), pages 26–52. Originally published in *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, in *Operating Systems Review 25*, 5 (December 1991), pages 1–15.

Although it has long been suggested that one could in principle store the contents of a file system on disk in the form of a finite log, this design is one of the few that demonstrates the full implications of that design strategy. The paper also presents a fine example of how to approach a system problem by carefully defining the objective, measuring previous systems to obtain a benchmark, and then comparing performance as well as functional aspects that cannot be measured.

9.3.2 H. T. Kung and John T. Robinson. On optimistic methods for concurrency control. *ACM Transactions on Database Systems 9*, 4 (June 1981), pages 213–226.

This early paper introduced the idea of using optimistic approaches to controlling updates to shared data. An optimistic scheme is one in which a transaction proceeds in the hope that its updates are not conflicting with concurrent updates of another transaction. At commit time, the transaction checks to see if the hope was justified. If so, the transaction commits. If not, the transaction aborts and tries again. Applications that use a database in which contention for particular records is infrequent may run more efficiently with this optimistic scheme than with a scheme that always acquires locks to coordinate updates.

See also the paper by Lampson and Sturgis, reading *1.8.7* and the paper by Ganger and Patt, reading *6.3.3*.

10 Consistency and Durable Storage

10.1 Consistency

10.1.1 J. R. Goodman. Using cache memory to reduce processor-memory traffic. *Proceedings of the 10th Annual International Symposium on Computer Architecture*, pages 124–132 (1983).

The paper that introduced a protocol for cache-coherent shared memory using snoopy caches. The paper also sparked much research in more scalable designs for cache-coherent shared memory.

10.1.2 James J. Kistler and M[ahadarev] Satyanarayanan. Disconnected operation in the Coda file system. *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, in *Operating Systems Review 25*, 5 (December 1991), pages 213–225.

Coda is a variation of the Andrew File System (AFS) that provides extra fault tolerance features. It is notable for using the same underlying mechanism to deal both with accidental disconnection due to network partition and the intentional disconnection associated with portable computers. This paper is very well written.

10.1.3 Jim Gray et al. The dangers of replication and a solution. *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, in *ACM SIGMOD Record 25*, 2 (June 1996), pages 173–182.

This paper describes the challenges for replication protocols in situations where the replicas are stored on mobile computers that are frequently disconnected. The paper argues that trying to provide transactional semantics for an optimistic replication protocol in this setting is unstable because there will be too many reconciliation conflicts. It proposes a new two-tier protocol for reconciling disconnected replicas that addresses this problem.

10.1.4 Leslie Lamport. Paxos made simple. Distributed computing (column), *ACM SIGACT News 32*, 4 (Whole Number 121, December 2001), pages 51–58.

This paper describes an intricate protocol, Paxos, in a simple way. The Paxos protocol allows several computers to agree on a value (e.g., the list of available computers in a replicated service) in the face of network and computer failures. It is an important building block in building fault tolerant services.

10.1.5 Fred Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys 22*, 4 (1990), pages 299–319.

This paper provides a clear description of one of the most popular approaches for building fault tolerant services, the replicated-state machine approach.

10.1.6 Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM 21*, 7 (1978), pages 558–565.

This paper introduces an idea that is now known as Lamport clocks. A Lamport clock provides a global, logical clock for a distributed system that respects the physical clocks of the computers comprising the distributed system and the communication between them. The paper also introduces the idea of replicated state machines.

10.1.7 David K. Gifford. Weighted voting for replicated data. *Proceedings of the Seventh ACM Symposium on Operating Systems Principles*, in *Operating Systems Review 13*, 5 (December 1979), pages 150–162. Also available as Xerox Palo Alto Research Center Technical Report CSL–79–14 (September 1979).

The work discusses a replicated data algorithm that allows the trade-off between reliability and performance to be adjusted by assigning weights to each data copy and requiring transactions to collect a quorum of those weights before reading or writing.

10.1.8 Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems* 7, 4 (November 1989), pages 321–359.

This paper describes a method to create a shared virtual memory across several separated computers that can communicate only with messages. It uses hardware support for virtual memory to cause the results of a write to a page to be observed by readers of that page on other computers. The goal is to allow programmers to write parallel applications on a distributed computer system in shared-memory style instead of a message-passing style.

10.1.9 Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (October 2003), pages 29–43. Also in *Operating Systems Review 37*, 5 (December 2003).

This paper introduces a file system used in many of Google's applications. It aggregates the disks of thousands of computers in a cluster into a single

storage system with a simple file system interface. Its design is optimized for large files and replicates files for fault tolerance. The Google File System is used in the storage back-end of many of Google's applications, including search.

10.1.10 F[ay] Chang et al. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems 26*, 2 article 4 (2008), pages 1–26.

This paper describes a database-like system for storing petabytes of structured data on thousands of commodity servers.

10.2 Durable Storage

10.2.1 Raymond A. Lorie. The long-term preservation of digital information. *Proceedings of the First ACM/IEEE Joint Conference on Digital Libraries* (2001), pages 346–352.

This is a thoughtful discussion of the problems of archiving digital information despite medium and technology obsolescence.

10.2.2 Randy H. Katz, Garth A. Gibson, and David A. Patterson. Disk system architectures for high performance computing. *Proceedings of the IEEE 77*, 12 (December 1989), pages 1842–1857.

The first part of this reference paper on Redundant Arrays of Independent Disks (RAID) reviews disk technology; the important material is the catalog of six varieties of RAID organization.

10.2.3 Petros Maniatis et al. LOCKSS: A Peer-to-peer digital preservation system *ACM Transactions on Computer Systems 23*, 1 (February 2005), pages 2–50.

This paper describes a peer-to-peer system for preserving access to journals and other archival information published on the Web. Its design is based on the mantra "lots of copies keep stuff safe" (LOCKSS). A large number of persistent Web caches keep copies and cooperate to detect and repair damage to their copies using a new voting scheme.

10.2.4 A[lan J.] Demers et al. Epidemic algorithms for replicated database maintenance. *Proceedings of the Sixth Symposium on Principles of Distributed Computing* (August 1987), pages 1–12. Also in *Operating Systems Review 22*, 1 (January 1988), pages 8–32.

This paper describes an epidemic protocol to update data that is replicated on many machines. The essence of an epidemic protocol is that each computer periodically gossips with some other, randomly chosen computer and exchanges information; multiple computers thus learn about all updates in a viral fashion. Epidemic protocols can be simple and robust, yet can spread updates relatively quickly.

10.3 Reconciliation

10.3.1 Douglas B. Terry et al. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proceedings of the 15th Symposium on Operating Systems Principles* (December 1995), in *Operating Systems Review 29*, 5 (December 1995), pages 172–183.

This paper introduces a replication scheme for computers that share data but are not always connected. For example, each computer may have a copy of a calendar, which it can update optimistically. Bayou will propagate these updates, detect conflicts, and attempt to resolve conflicts, if possible.

10.3.2 Trevor Jim, Benjamin C. Pierce, and Jérôme Vouillon. How to build a file synchronizer. (A widely circulated piece of grey literature—dated February 22, 2002 but never published.)

This paper describes the nuts and bolts of Unison, a tool that efficiently synchronizes the files stored on two computers. Unison is targeted to users who have their files stored in several places (e.g., on a server at work, a laptop to carry while traveling, and a desktop at home) and would like to have all the files on the different computers be the same.

11 Information Security

11.1 Privacy

The fundamental book about privacy is reading *1.1.6* by Alan Westin.

11.1.1 Arthur R. Miller. *The Assault on Privacy*. University of Michigan Press, Ann Arbor, Michigan, 1971. ISBN: 0–47265500–0. 333 pages. (Out of print.)

This book articulately spells out the potential effect of computerized data-gathering systems on privacy, and of possible approaches to improving legal protection. Part of the latter is now out of date because of advances in legislation, but most of this book is still of much interest.

11.1.2 Daniel J. Weitzner et al. Information accountability. *Communications of the ACM 51*, 6 (June 2008), pages 82–87.

The paper suggests that in the modern world Westin's definition covers only a subset of privacy. See Sidebar 11.1 for a discussion of the paper's proposed extended definition.

11.2 Protection Architectures

11.2.1 Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE 63*, 9 (September 1975), pages 1278–1308.

After 30 years, this paper (an early version of the current Chapter 11) still provides an effective treatment of protection mechanics in multiuser systems. Its

emphasis on protection inside a single system, rather than between systems connected to a network, is one of its chief shortcomings, along with antique examples and omission of newer techniques of certification such as authentication logic.

11.2.2 R[oger] M. Needham. Protection systems and protection implementations. *AFIPS Fall Joint Conference 41*, Part I (December 1972), pages 571–578.

This paper is probably as clear an explanation of capability systems as one is likely to find. For another important paper on capabilities, see Fabry, reading *3.1.2*.

11.3 Certification, Trusted Computer Systems, and Security Kernels

11.3.1 Butler [W.] Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems 10*, 4 (November 1992), pages 265–310.

This paper, one of a series on a logic that can be used to reason systematically about authentication, provides a relatively complete explication of the theory and shows how to apply it to the protocols of a distributed system.

11.3.2 Edward Wobber, Martín Abadi, Michael Burrows, and Butler W. Lampson. Authentication in the Taos operating system. *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, in *Operating Systems Review 27*, 5 (December 1993), pages 256–269.

This paper applies the authentication logic developed in reading *11.3.1* to an experimental operating system. In addition to providing a concrete example, the explanation of the authentication logic itself is a little more accessible than that in the other paper.

11.3.3 Ken L. Thompson. Reflections on trusting trust. *Communications of the ACM 27*, 8 (August 1984), pages 761–763.

Anyone seriously interested in developing trusted computer systems should think hard about the implications for verification that this paper raises. Thompson demonstrates the ease with which a compiler expert can insert undetectable Trojan Horses into a system. Reading 11.3.4 describes a way to detect a Trojan horse. [The original idea that Thompson describes came from a paper whose identity he could not recall at the time, and which is credited with a footnote asking for help locating it. The paper was a technical report of the United States Air Force Electronic Systems Division at Hanscom Air Force Base. Paul A. Karger and Roger R. Schell. *Multics Security Evaluation: Vulnerability Analysis*. *ESD–TR–74–193, Volume II* (June 1974), page 52.]

11.3.4 David A. Wheeler. Countering trusting trust through diverse double-Compiling. *Proceedings of the 21st Annual Computer Security Applications Conference* (2005), pages 28–40.

This paper proposes a solution that the author calls "diverse double compiling", to detect the attack discussed in Thompson's paper on trusting trust (see reading *11.3.3*). The idea is to recompile a new, untrusted compiler's source code twice: first, using a trusted compiler, and second, using the result of this compilation. If the resulting binary for the compiler is bit-for-bit identical with the untrusted compiler's original binary, then the source code accurately represents the untrusted binary, which is the first step in developing trust in the new compiler.

11.3.5 Paul A. Karger et al. A VMM security kernel for the VAX architecture. *1990 IEEE Computer Society Symposium on Security and Privacy* (May 1990), pages 2–19.

In the 1970's, the U.S. Department of Defense undertook a research effort to create trusted computer systems for defense purposes and in the process created a large body of literature on the subject. This paper distills most of the relevant ideas from that literature into a single, readable case study, and it also provides pointers to other key papers for those seeking more details on these ideas.

11.3.6 David D. Clark and David. R. Wilson. A Comparison of commercial and military computer security policies. *1987 IEEE Symposium on Security and Privacy* (April 1987), pages 184–194.

This thought-provoking paper outlines the requirements for security policy in commercial settings and argues that the lattice model is often not applicable. It suggests that these applications require a more object-oriented model in which data may be modified only by trusted programs.

11.3.7 Jaap-Henk Hoepman and Bart Jacobs. Increased security through open source. *Communications of the ACM 50*, 1 (January 2007), pages 79–83.

It has long been argued that the open design principle (see Section 11.1.4) is important to designing secure systems. This paper extends that argument by making the case that the availability of source code for a system is important in ensuring the security of its implementation.

See also reading *1.3.15* by Garfinkel and Spafford, reading *5.2.1* by Lampson and Sturgis, and reading *5.2.2* by Schroeder, Clark, and Saltzer.

11.4 Authentication

11.4.1 Robert [H.] Morris and Ken [L.] Thompson. Password security: A case history. *Communications of the ACM 22*, 11 (November 1979), pages 594–597.

This paper is a model of how to explain something in an accessible way. With a minimum of jargon and an historical development designed to simplify things for the reader, it describes the UNIX password security mechanism.

11.4.2 Frank Stajano and Ross J. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. *Security Protocols Workshop 1999*, pages 172–194.

This paper discusses the problem of how a new device (e.g., a surveillance camera) can establish a secure relationship with the remote controller of the device's owner, instead of its neighbor's or adversary's. The paper's solution is that a device will recognize as its owner the first principal that sends it an authentication key. As soon as the device receives a key, its status changes from newborn to imprinted, and it stays faithful to that key until its death. The paper illustrates the problem and solution, using a vivid analogy of how ducklings authenticate their mother (see Sidebar 11.5).

11.4.3 David Mazières. *Self-certifying File System*. Ph.D. Thesis, Massachusetts Institute of Technology Department of Electrical Engineering and Computer Science (May 2000).

This thesis proposes a design for a cross-administrative domain file system that separates the file system from the security mechanism using an idea called self-certifying path names. Self-certifying names can be found in several other systems.

See also Sidebar 11.6 on Kerberos and reading *3.2.5*, which uses cryptographic techniques to secure a personal naming system.

11.5 Cryptographic Techniques

The fundamental books about cryptography applied to computer systems are reading *1.2.4*, by Bruce Schneier, and reading *1.3.13*, by Alfred Menezes et al. In light of these two books, the first few papers from the 1970's listed below are primarily of historical interest. There is also a good, more elementary, treatment of cryptography in the book by Simson Garfinkel, reading *1.3.15*. Note that all of these books and papers focus on the application of cryptography, not on crypto-mathematics, which is a distinct area of specialization not covered in this reading list. An accessible crypto-mathematics reference is reading *1.3.14*.

11.5.1 R[onald] L. Rivest, A[di] Shamir, and L[en] Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM 21*, 2 (February 1978), pages 120–126.

This paper was the first to suggest a possibly workable public key system.

11.5.2 Whitfield Diffie and Martin E. Hellman. Exhaustive cryptanalysis of the NBS Data Encryption Standard. *Computer 10*, 6 (June 1977), pages 74–84.

This is the unofficial analysis of how to break the DES by brute force—by building special-purpose chips and arraying them in parallel. Twenty-five years later, brute force still seems to be the only promising attack on DES, but the intervening improvements in hardware technology make special chips unnecessary—an array of personal computers on the Internet can do the job. The Advanced Encryption Standard (AES) is DES's successor (see Section 11.9.3.1).

11.5.3 Ross J. Anderson. Why cryptosystems fail. *Communications of the ACM 37*, 11 (November 1994), pages 32–40.

Anderson presents a very nice analysis of what goes wrong in real-world cryptosystems—secure modules don't necessary lead to secure systems—and the applicability of systems thinking in their design. He points out that merely doing the best possible design isn't enough; a feedback loop that corrects errors in the design following experience in the field is an equally important component that is sometimes forgotten.

11.5.4 David Wagner and Bruce Schneier. Analysis of the SSL 3.0 protocol. *Proceedings of the Second USENIX Workshop on Electronic Commerce, Volume 2* (November 1996), pages 29–40.

This paper is useful not only because it provides a careful analysis of the security of the subject protocol, but it also explains how the protocol works in a form that is more accessible than the protocol specification documents. The originally published version was almost immediately revised with corrections. The revised version is available on the World Wide Web at <http://www.counterpane.com/ssl.html>.

11.5.5 M[ihir] Bellare, R[an] Canetti, and H[ugo] Krawczyk. Keying hash functions for message authentication. *Proceedings of the 16th International Cryptography Conference* (August 1996), pages 1–15. (Also see H. Krawczyk, M. Bellare, and R. Canetti, HMAC: Keyed-hashing for message authentication, *Request For Comments RFC 2104*, Internet Engineering Task Force (February 1997).

This paper and the RFC introduce and define HMAC, a hash function used in widely deployed protocols.

11.5.6 David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM 24*, 2 (February 1981), pages 84–88.

This paper introduces a system design, named mixnet, that allows a sender of a message to hide its true identity from a receiver but still allow the receiver to respond.

11.6 Adversaries (The Dark Side)

Section 11.11 on war stories gives a wide range of examples of how adversaries can break a system's security. This section lists a few papers that provide a longer and more detailed descriptions of attacks. This is a fast-moving area; as soon as designers fend off new attacks, adversaries try to find new attacks. This arms race is reflected in some of the following readings, and although some of the attacks described have become ineffective (or will over time), these papers provide valuable insights. The proceedings of *Usenix Security* and *Computer and Communication Security* often contain papers explaining current attacks, and conferences run by the so-called "black hat" community document the "progress" on the dark side.

11.6.1 Eugene Spafford, Crisis and aftermath. *Communications of the ACM 32*, 6 (June 1989), pages 678–687.

This paper documents how the Morris worm works. It was one of the first worms, as well as one of the most sophisticated.

11.6.2 Jonathan Pincus and Brandon Baker. Beyond stack smashing: Recent advances in exploiting buffer overruns. *IEEE Security and Privacy 2*, 4 (August 2004), pages 20–27.

This paper describes how buffer overrun attacks have evolved since the Morris worm.

11.6.3 Abhishek Kumar, Vern Paxson, and Nicholas Weaver. Exploiting underlying structure for detailed reconstruction of an Internet scale event. *Proceedings of the ACM Internet Measurement Conference* (October 2005), pages 351–364.

This paper describes the Witty worm and how the authors were able to track down its source. The work contains many interesting nuggets of information.

11.6.4 Vern Paxson. An analysis of using reflectors for distributed denial-of-service attacks. *Computer Communications Review 31*, 3 (July 2001), pages 38–47.

This paper describes how an adversary can trick a large set of Internet servers to send their combined replies to a victim and in that way launch a denial-of-service attack on the victim. It speculates on several possible directions for defending against such attacks.

11.6.5 Chris Kanich et al. Spamalytics: An empirical analysis of spam marketing conversion. *Proceedings of the ACM Conference on Computer and Communications Security (CCS)* (October 2008), pages 3–14.

This paper describes the infrastructure that spammers use to send unsolicited e-mail and tries to establish what the financial reward system is for spammers. This paper has its shortcomings, but it is one of the few papers that tries to understand the economics behind spam.

11.6.6 Tom Jagatic, Nathaniel Johnson, Markus Jakobsson, and Filippo Menczer. Social phishing. *Communications of the ACM 50*, 10 (October 2007), pages 94–100.

This study investigates the success rate of individual phishing attacks.