Preface to the Second Edition

When we wrote the first edition of this book, it was not very common for college courses on networking to include programming components. That seems difficult to believe now, when the Internet has become so important to our world, and the pedagogical benefits of hands-on programming and real-world protocol examples are so widely accepted. Although there are now other languages that provide access to the Internet, interest in the original C-based *Berkeley Sockets* remains high. The Sockets API (application programming interface) for networking was developed at UC Berkeley in the 1980s for the BSD flavor of UNIX—one of the very first examples of what would now be called an open-source project.

The Sockets API and the Internet both grew up in a world of many competing protocol families—IPX, Appletalk, DECNet, OSI, and SNA in addition to Transmission Control Protocol/Internet Protocal (TCP/IP)—and Sockets was designed to support them all. Fewer protocol families were in common use by the time we wrote the first edition of this book, and the number today is even smaller. Nevertheless, as we predicted in the first edition, the Sockets API remains important for those who want to design and build distributed applications that use the Internet—that is, that use TCP/IP. And the interface has proven robust enough to support the new version of the Internet Protocol (IPv6), which is now supported on virtually all common computing platforms.

Two main considerations motivated this second edition. First, based on our own experience and feedback from others, we found that some topics needed to be presented in more depth and that others needed to be expanded. The second consideration is the increasing acceptance and use of IP version 6, which is now supported by essentially all current end system platforms. At this writing, it is not possible to use IPv6 to exchange messages with a large fraction of hosts on the Internet, but it *is* possible to assign an IPv6 address to many of them. Although it is still too early to tell whether IPv6 will take over the world, it is not too early to start writing applications to be prepared.

Changes from the First Edition

We have updated and considerably expanded most of the material, having added two chapters. Major changes from the first edition include:

- IP version 6 coverage. We now include three kinds of code: IPv4-specific, IPv6-specific, and generic. The code in the later chapters is designed to work with either protocol version on dual-stack machines.
- An additional chapter on socket programming in C++ (contributed by David B. Sturgill). The PracticalSocket library provides wrappers for basic socket functionality. These allow an instructor to teach socket programming to students without C programming background by giving them a library and then gradually peeling back the layers. Students can start developing immediately after understanding addresses/ports and client/server. Later they can be shown the details of socket programming by peeking inside the wrapper code. Those teaching a subject that uses networking (e.g., OS) can use the library and only selectively peel back the cover.
- Enhanced coverage of data representation issues and strategies for organizing code that sends and receives messages. In our instructional experience, we find that students have less and less understanding of how data is actually stored in memory,[1] so we have attempted to compensate with more discussion of this important issue. At the same time, internationalization will only increase in importance, and thus we have included basic coverage of wide characters and encodings.
- Omission of the reference section. The descriptions of most of the functions that make up the Sockets API have been collected into the early chapters. However, with so many online sources of reference information—including "man pages"—available, we chose to leave out the complete listing of the API in favor of more code illustrations.
- Highlighting important but subtle facts and caveats. Typographical devices call out important concepts and information that might otherwise be missed on first reading.

Although the scope of the book has expanded, we have not included everything that we might have (or even that we were asked to include); examples of topics left for more comprehensive texts (or the next edition) are raw sockets and programming with WinSock.

Intended Audience

We originally wrote this book so that we would have something to hand our students when we wanted them to learn socket programming, so we would not have to take up valuable class time

[1] We speculate that this is due to the widespread use of C++ and Java, which hide such details from the programmer, in undergraduate curricula.

teaching it. In the years since the first edition, we have learned a good deal about the topics that students need lots of help on, and those where they do not need as much handholding. We also found that our book was appreciated at least as much by practitioners who were looking for a gentle introduction to the subject. Therefore, this book is aimed simultaneously at two general audiences: students in introductory courses in computer networks (graduate or undergraduate) with a programming component, and practitioners who want to write their own programs that communicate over the Internet. For students, it is intended as a supplement, not as a primary text about networks. Although this second edition is significantly bigger in size and scope than the first, we hope the book will still be considered a good value in that role. For practitioners who just want to write some useful code, it should serve as a standalone *introduction*—but readers in that category should be warned that this book will not make them experts. Our philosophy of learning by doing has not changed, nor has our approach of providing a concise tutorial sufficient to get one started learning on one's own, and leaving the comprehensive details to other authors. For both audiences, our goal is to take you far enough so that you can start experimenting and learning on your own.

Assumed Background

We assume basic programming skills and experience with C and UNIX. You are expected to be conversant with C concepts such as pointers and type casting, and you should have a basic understanding of the binary representation of data. Some of our examples are factored into files that should be compiled separately; we assume that you can deal with that.

Here is a little test: If you can puzzle out what the following code fragment does, you should have no problem with the code in this book:

```
typedef struct {
  int a;
  short s[2];
} MSG;

MSG *mp, m = {4, 1, 0};
char  *fp, *tp;
mp = (MSG *) malloc(sizeof(MSG));
for (fp = (char *)m.s, tp = (char *)mp->s; tp < (char *)(mp+1);)
   *tp++ = *fp++;
```

If you do not understand this fragment, do not despair (there is nothing quite so convoluted in our code), but you might want to refer to your favorite C programming book to find out what is going on here.

You should also be familiar with the UNIX notions of process/address space, command-line arguments, program termination, and regular file input and output. The material in Chapters 4 and 6 assumes a somewhat more advanced grasp of UNIX. Some prior exposure to networking concepts such as protocols, addresses, clients, and servers will be helpful.

Platform Requirements and Portability

Our presentation is UNIX-based. When we were developing this book, several people urged us to include code for Windows as well as UNIX. It was not possible to do so for various reasons, including the target length (and price) we set for the book.

For those who only have access to Windows platforms, please note that the examples in the early chapters require minimal modifications to work with WinSock. (You have to change the include files and add a setup call at the beginning of the program and a cleanup call at the end.) Most of the other examples also require very slight additional modifications. However, some are so dependent on the UNIX programming model that it does not make sense to port them to WinSock. WinSock-ready versions of the other examples, as well as detailed descriptions of the code modifications required, are available from the book's Web site at *www.mkp.com/socket*. Note also that almost all of our example code works with minimal modifications under the **Cygwin** UNIX library package for Windows, which is available online.

For this second edition, we have adopted the C99 language standard. This version of the language is supported by most compilers and offers so many readability-improving advantages—including line-delimited comments, fixed-size integer types, and declarations anywhere in a block—that we could not justify not using it.

Our code makes use of the "Basic Socket Interface Extensions for IPv6" **?**. Among these extensions is a new and different interface to the name system. Because we rely completely on this new interface (`getaddrinfo()`), our generic code may not run on some older platforms. However, we expect that most modern systems will run our code just fine.

The example programs included here have all been tested (and should compile and run without modification) on both *NIX and MacOS. Header (`.h`) file locations and dependencies are, alas, not quite standard and may require some fiddling on your system. Socket option support also varies widely across systems; we have tried to focus on those that are most universally supported. Consult your API documentation for system specifics. (By API documentation we mean the "man pages" for your system. To learn about this, type "man man" or use your favorite web search tool.)

Please be aware that although we strive for a basic level of robustness, the primary goal of our code examples is pedagogy, and the code is **not production quality.** We have sacrificed some robustness for brevity and clarity, especially in the generic server code. (It turns out to be nontrivial to write a server that works under all combinations of IPv4 and IPv6 protocol configurations and also maximizes the likelihood of successful client connection under all circumstances.)

This Book Will Not Make You an Expert!

We hope this second edition will be useful as a resource, even to those who already know quite a bit about sockets. As with the first edition, we learned some things in writing it. But becoming an expert takes years of experience, as well as other, more comprehensive sources **?**, **?**.

The first chapter is intended to give "just enough" of the big picture to get you ready to write code. Chapter **??** shows you how to write TCP clients and servers using either IPv4 or IPv6. Chapter **??** shows how to make your clients and servers use the network's name service, and also describes how to make them IP-version-independent. Chapter **??** covers User Datagram Protocol (UDP). Chapters **??** and **??** provide background needed to write more programs, while Chapter **??** relates some of what is going on in the Sockets implementation to the API calls; these three are essentially independent and may be presented in any order. Finally, Chapter **??** presents a C++ class library that provides simplified access to socket functionality.

Throughout the book, certain statements are highlighted like this: **This book will not make you an expert!** Our goal is to bring to your attention those subtle but important facts and ideas that one might miss on first reading. The marks in the margin tell you to "**note well**" whatever is in bold.

Acknowledgments

Many people contributed to making this book a reality. In addition to all those who helped us with the first edition (Michel Barbeau, Steve Bernier, Arian Durresi, Gary Harkin, Ted Herman, Lee Hollaar, David Hutchison, Shunge Li, Paul Linton, Ivan Marsic, Willis Marti, Kihong Park, Dan Schmitt, Michael Scott, Robert Strader, Ben Wah, and Ellen Zegura), we especially thank David B. Sturgill, who contributed code and text for Chapter **??**, and Bobby Krupczak for his help in reviewing the draft of this second edition. Finally, to the folks at Morgan Kaufmann/Elsevier—Rick Adams, our editor, assistant editor Maria Alonso, and project manager Melinda Ritchie—thank you for your patience, help, and caring about the quality of our book.

Feedback

We are very interested in weeding out errors and otherwise improving future editions/printings, so if you find any errors, please send an e-mail to either of us. We will maintain an errata list on the book's Web page.

M.J.D. jeff_donahoo@baylor.edu

K.L.C. calvert@netlab.uky.edu