Preface

This book describes application development using the Microsoft Windows Application Programming Interface (API), concentrating on the core system services, including the file system, process and thread management, interprocess communication, network programming, and synchronization. The examples concentrate on realistic scenarios, and in many cases they're based on real applications I've encountered in practice.

The Win32/Win64 API, or the Windows API, is supported by Microsoft's family of 32-bit and 64-bit operating systems; versions currently supported and widely used include Windows 7, XP, Vista, Server 2003, Server 2008, and CE. Older Windows family members include Windows 2000, NT, Me, 98, and 95; these systems are obsolete, but many topics in this book still apply to these older systems.

The Windows API is an important factor for application development, frequently replacing the POSIX API (supported by UNIX and Linux) as the preferred API for applications targeted at desktop, server, and embedded systems now and for the indefinite future. Many programmers, regardless of experience level, need to learn the Windows API quickly, and this book is designed for them to do so.

Objectives and Approach

The objectives I've set for the book are to explain what Windows is, show how to use it in realistic situations, and do so as quickly as possible without burdening you with unnecessary detail. This book is not a reference guide, but it explains the central features of the most important functions and shows how to use them together in practical programming situations. Equipped with this knowledge, you will be able to use the comprehensive Microsoft reference documentation to explore details, advanced options, and the more obscure functions as requirements or interests dictate. I have found the Windows API easy to learn using this approach and have greatly enjoyed developing Windows programs, despite occasional frustration. This enthusiasm will show through at times, as it should. This does not mean that I feel that Windows is necessarily better than other operating system (OS) APIs, but it certainly has many attractive features and improves significantly with each major new release.

Many Windows books spend a great deal of time explaining how processes, virtual memory, interprocess communication, and preemptive scheduling work without showing how to use them in realistic situations. A programmer experienced in UNIX, Linux, IBM MVS, or another OS will be familiar with these concepts and will be

impatient to find out how they are implemented in Windows. Most Windows books also spend a great deal of space on the important topic of user interface programming. This book intentionally avoids the user interface, beyond discussing simple character-based console I/O, in the interest of concentrating on the important core features.

I've taken the point of view that Windows is just an OS API, providing a well-understood set of features. Many programmers, regardless of experience level, need to learn Windows quickly. Furthermore, understanding the Windows API is invaluable background for programmers developing for the Microsoft .NET Framework.

The Windows systems, when compared with other systems, have good, bad, and average features and quality. Recent releases (Windows 7, Vista, Server 2008) provide new features, such as condition variables, that both improve performance and simplify programming. The purpose of this book is to show how to use those features efficiently and in realistic situations to develop practical, high-quality, and high-performance applications.

Audience

I've enjoyed receiving valuable input, ideas, and feedback from readers in all areas of the target audience, which includes:

- Anyone who wants to learn about Windows application development quickly, regardless of previous experience.
- Programmers and software engineers who want to port existing Linux or UNIX (the POSIX API) applications to Windows. Frequently, the source code must continue to support POSIX; that is, source code portability is a requirement. The book frequently compares Windows, POSIX, and standard C library functions and programming models.
- Developers starting new projects who are not constrained by the need to port existing code. Many aspects of program design and implementation are covered, and Windows functions are used to create useful applications and to solve common programming problems.
- Application architects and designers who need to understand Windows capabilities and principles.
- Programmers using COM and the .NET Framework, who will find much of the information here helpful in understanding topics such as dynamic link libraries (DLLs), thread usage and models, interfaces, and synchronization.
- Computer science students at the upper-class undergraduate or beginning graduate level in courses covering systems programming or application devel-

opment. This book will also be useful to those who are learning multithreaded programming or need to build networked applications. This book would be a useful complementary text to a classic book such as *Advanced Programming in the UNIX Environment* (by W. Richard Stevens and Stephen A. Rago) so that students could compare Windows and UNIX. Students in OS courses will find this book to be a useful supplement because it illustrates how a commercially important OS provides essential functionality.

The only other assumption, implicit in all the others, is a knowledge of C or C++ programming.

Windows Progress Since the Previous Editions

The first edition of this book, titled *Win32 System Programming,* was published in 1997 and was updated with the second edition (2000) and the third edition (2004). Much has changed, and much has stayed the same since these previous editions, and Windows has been part of ongoing, rapid progress in computing technology. The outstanding factors to me that explain the fourth edition changes are the following:

- The Windows API is extremely stable. Programs written in 1997 continue to run on the latest Windows releases, and Windows skills learned now or even years ago will be valuable for decades to come.
- Nonetheless, the API has expanded, and there are new features and functions that are useful and sometimes mandatory. Three examples of many that come to mind and have been important in my work are (1) the ability to work easily with large files and large, 64-bit address spaces, (2) thread pools, and (3) the new condition variables that efficiently solve an important synchronization problem.
- Windows scales from phones to handheld and embedded devices to laptops and desktop systems and up to the largest servers.
- Windows has grown and scaled from the modest resources required in 1997 (16MB of RAM and 250MB of free disk space!) to operate efficiently on systems orders of magnitude larger and faster but often cheaper.
- 64-bit systems, multicore processors, and large file systems are common, and our application programs must be able to exploit these systems. Frequently, the programs must also continue to run on 32-bit systems.

Changes in the Fourth Edition

This fourth edition presents extensive new material along with updates and reorganization to keep up with recent progress and:

- Covers important new features in Windows 7, Vista, and Server 2008.
- Demonstrates example program operation and performance with screenshots.
- Describes and illustrates techniques to assure that relevant applications scale to run on 64-bit systems and can use large files. Enhancements throughout the book address this issue.
- Eliminates discussion of Windows 95, 98, and Me (the "Windows 9x" family), as well as NT and other obsolete systems. Program examples freely exploit features supported only in current Windows versions.
- Provides enhanced coverage of threads, synchronization, and parallelism, including performance, scalability, and reliability considerations.
- Emphasizes the important role and new features of Windows servers running high-performance, scalable, multithreaded applications.
- Studies performance implications of different program designs, especially in file access and multithreaded applications with synchronization and parallel programs running on multicore systems.
- Addresses source code portability to assure operation on Windows, Linux, and UNIX systems. Appendix B is enhanced from the previous versions to help those who need to build code, usually for server applications, that will run on multiple target platforms.
- Incorporates large quantities of excellent reader and reviewer feedback to fix defects, improve explanations, improve the organization, and address numerous details, large and small.

Organization

Chapters are organized topically so that the features required in even a single-threaded application are covered first, followed by process and thread management features, and finally network programming in a multithreaded environment. This organization allows you to advance logically from file systems to memory management and file mapping, and then to processes, threads, and synchronization, followed by interprocess and network communication and security. This organization also allows the examples to evolve in a natural way, much as a developer might cre-

ate a simple prototype and then add additional capability. The advanced features, such as asynchronous I/O and security, appear last.

Within each chapter, after introducing the functionality area, such as process management or memory-mapped files, we discuss important Windows functions and their relationships in detail. Illustrative examples follow. Within the text, only essential program segments are listed; complete projects, programs, include files, utility functions, and documentation are on the book's Web site (www.jmhartsoftware.com). Throughout, we identify those features supported only by current Windows versions. Each chapter suggests related additional reading and gives some exercises. Many exercises address interesting and important issues that did not fit within the normal text, and others suggest ways for you to explore advanced or specialized topics.

Chapter 1 is a high-level introduction to the Windows OS family and Windows. A simple example program shows the basic elements of Windows programming style and lays the foundation for more advanced Windows features. Win64 compatibility issues are introduced in Chapter 1 and are included throughout the book.

Chapters 2 and 3 deal with file systems, console I/O, file locking, and directory management. Unicode, the extended character set used by Windows, is also introduced in Chapter 2. Examples include sequential and direct file processing, directory traversal, and management. Chapter 3 ends with a discussion of registry management programming, which is analogous in many ways to file and directory management.

Chapter 4 introduces Windows exception handling, including Structured Exception Handling (SEH), which is used extensively throughout the book. By introducing it early, we can use SEH throughout and simplify some programming tasks and improve quality. Vectored exception handling is also described.

Chapter 5 treats Windows memory management and shows how to use memory-mapped files both to simplify programming and to improve performance. This chapter also covers DLLs. An example compares memory-mapped file access performance and scalability to normal file I/O on both 32-bit and 64-bit systems.

Chapter 6 introduces Windows processes, process management, and simple process synchronization. Chapter 7 then describes thread management in similar terms and introduces parallelism to exploit multiprocessor systems. Examples in each chapter show the many benefits of using threads and processes, including program simplicity and performance.

Chapters 8, 9, and 10 give an extended, in-depth treatment of Windows thread synchronization, thread pools, and performance considerations. These topics are complex, and the chapters use extended examples and well-understood models to help you obtain the programming and performance benefits of threads while avoiding the numerous pitfalls. New material covers new functionality along with

performance and scalability issues, which are important when building server-based applications, including those that will run on multiprocessor systems.

Chapters 11 and 12 are concerned with interprocess and interthread communication and networking. Chapter 11 concentrates on the features that are properly part of Windows—namely, anonymous pipes, named pipes, and mailslots. Chapter 12 discusses Windows Sockets, which allow interoperability with non-Windows systems using industry-standard protocols, primarily TCP/IP. Windows Sockets, while not strictly part of the Windows API, provide for network and Internet communication and interoperability, and the subject matter is consistent with the rest of the book. A multithreaded client/server system illustrates how to use interprocess communication along with threads.

Chapter 13 describes how Windows allows server applications, such as the ones created in Chapters 11 and 12, to be converted to Windows Services that can be managed as background servers. Some small programming changes will turn the servers into services.

Chapter 14 shows how to perform asynchronous I/O using overlapped I/O with events and completion routines. You can achieve much the same thing with threads, so examples compare the different solutions for simplicity and performance. In particular, as of Windows Vista, completion routines provide very good performance. The closely related I/O completion ports are useful for some scalable multithreaded servers, so this feature is illustrated with the server programs from earlier chapters. The final topic is waitable timers, which require concepts introduced earlier in the chapter.

Chapter 15 briefly explains Windows object security, showing, in an example, how to emulate UNIX-style file permissions. Additional examples shows how to secure processes, threads, and named pipes. Security upgrades can then be applied to the earlier examples as appropriate.

There are three appendixes. Appendix A describes the example code that you can download from the book's Web site (www.jmhartsoftware.com). Appendix B shows how to create source code that can also be built to run on POSIX (Linux and UNIX) systems; this requirement is common with server applications and organizations that need to support systems other than just Windows. Appendix C compares the performance of alternative implementations of some of the text examples so that you can gauge the trade-offs between Windows features, both basic and advanced.

UNIX and C Library Notes and Tables

Within the text at appropriate points, we contrast Windows style and functionality with the comparable POSIX (UNIX and Linux) and ANSI Standard C library features. Appendix B reviews source code portability and also contains a table list-

ing these comparable functions. This information is included for two principal reasons:

- Many people are familiar with UNIX or Linux and are interested in the comparisons between the two systems. If you don't have a UNIX/Linux background, feel free to skip those paragraphs in the text, which are indented and set in a smaller font.
- Source code portability is important to many developers and organizations.

Examples

The examples are designed to:

- Illustrate common, representative, and useful applications of the Windows functions.
- Correspond to real programming situations encountered in program development, consulting, and training. Some of my clients and course participants have used the code examples as the bases for their own systems. During consulting activities, I frequently encounter code that is similar to that used in the examples, and on several occasions I have seen code taken directly or modified from previous editions. (Feel free to do so yourself; an acknowledgment in your documentation would be greatly appreciated.) Frequently, this code occurs as part of COM, .NET, or C++ objects. The examples, subject to time and space constraints, are "real-world" examples and solve "real-world" problems.
- Emphasize how the functions actually behave and interact, which is not always as you might first expect after reading the documentation. Throughout this book, the text and the examples concentrate on interactions between functions rather than on the functions themselves.
- Grow and expand, both adding new capability to a previous solution in a natural manner and exploring alternative implementation techniques.
- Implement UNIX/Linux commands, such as `ls`, `touch`, `chmod`, and `sort`, showing the Windows functions in a familiar context while creating a useful set of utilities.[1] Different implementations of the same command also give us

[1] Several commercial and open source products provide complete sets of UNIX/Linux utilities; there is no intent to supplement them. These examples, although useful, are primarily intended to illustrate Windows usage. Anyone unfamiliar with UNIX or Linux should not, however, have any difficulty understanding the programs or their functionality.

an easy way to compare performance benefits available with advanced Windows features. Appendix C contains the performance test results.

Examples in the early chapters are usually short, but the later chapters present longer examples when appropriate.

Exercises at the end of each chapter suggest alternative designs, subjects for investigation, and additional functionality that is important but beyond the book's scope. Some exercises are easy, and a few are very challenging. Frequently, clearly labeled defective solutions are provided, because fixing the bugs is an excellent way to sharpen skills.

All examples have been debugged and tested under Windows 7, Vista, Server 2008, XP, and earlier systems. Testing included 32-bit and 64-bit versions. All programs were also tested on both single-processor and multiprocessor systems using as many as 16 processors. The client/server applications have been tested using multiple clients simultaneously interacting with a server. Nonetheless, there is no guarantee or assurance of program correctness, completeness, or fitness for any purpose. Undoubtedly, even the simplest examples contain defects or will fail under some conditions; such is the fate of nearly all software. I will, however, gratefully appreciate any messages regarding program defects—and, better still, fixes, and I'll post this information on the book's Web site so that everyone will benefit.

The Web Site

The book's Web site (www.jmhartsoftware.com) contains a downloadable *Examples* file with complete code and projects for all the book's examples, a number of exercise solutions, alternative implementations, instructions, and performance evaluation tests. This material will be updated periodically to include new material and corrections.

The Web site also contains book errata, along with additional examples, reader contributions, additional explanations, and much more. The site also contains PowerPoint slides that can be used for noncommercial instructional purposes. I've used these slides numerous times in professional training courses, and they are also suitable for college courses.

The material will be updated as required when defects are fixed and as new input is received. If you encounter any difficulties with the programs or any material in the book, check these locations first because there may already be a fix or explanation. If that does not answer your question, feel free to send e-mail to `jmh_assoc@hotmail.com` or `jmhart62@gmail.com`.

Acknowledgments

Numerous people have provided assistance, advice, and encouragement during the fourth edition's preparation, and readers have provided many important ideas and corrections. The Web site acknowledges the significant contributions that have found their way into the fourth edition, and the first three editions acknowledge earlier valuable contributions. See the Web site for a complete list.

Three reviewers deserve the highest possible praise and thanks for their incisive comments, patience, excellent suggestions, and deep expertise. Chris Sells, Jason Beres, and especially Raymond Chen made contributions that improved the book immeasurably. To the best of my ability, I've revised the text to address their points and invaluable input.

Numerous friends and colleagues also deserve a note of special thanks; I've learned a lot from them over the years, and many of their ideas have found their way into the book in one way or another. They've also been generous in providing access to test systems. In particular, I'd like to thank my friends at Sierra Atlantic, Cilk Arts (now part of Intel), Vault USA, and Rimes Technologies.

Anne H. Smith, the compositor, used her skill, persistence, and patience to prepare this new edition for publication; the book simply would not have been possible without her assistance. Anne and her husband, Kerry, also have generously tested the sample programs on their equipment.

The staff at Addison-Wesley exhibited the professionalism and expertise that make an author's work a pleasure. Joan Murray, the editor, and Karen Gettman, the editor-in-chief, worked with the project from the beginning making sure that no barriers got in the way and assuring that hardly any schedules slipped. Olivia Basegio, the editorial assistant, managed the process throughout, and John Fuller and Elizabeth Ryan from production made the production process seem almost simple. Anna Popick, the project editor, guided the final editing steps and schedule. Carol Lallier and Lori Newhouse, the copy editor and proofreader, made valuable contributions to the book's readability and consistency.

Johnson (John) M. Hart
jmhart62@gmail.com
December, 2009

This page intentionally left blank

About the Author

Johnson (John) M. Hart is a consultant in the fields of Microsoft Windows and .NET application development, open systems computing, technical training and writing, and software engineering. He has more than twenty-five years of experience as a software engineer, manager, engineering director, and senior technology consultant at Cilk Arts, Inc., Sierra Atlantic, Hewlett-Packard, and Apollo Computer. John also develops and delivers professional training courses in Windows, UNIX, and Linux and was a computer science professor at the University of Kentucky for nine years. He is the author of technical, trade, and academic articles and books including the first, second, and third editions of *Windows System Programming*.

This page intentionally left blank