The UNIX system was first described in a 1 974 paper in the Communications of the ACM [Thompson 74] by Ken Thompson and Dennis Ritchie. Since that time, it has become increasingly widespread and popular throughout the computer industry where more ' and more vendors are offering support for it on their machines. It is especially popular in universities where it is frequently used for operating systems research and case studies. Many books and papers have described parts of the system, among them, two special issues of the Bell System Technical Journal in 1 978 [ BSTJ 78] and 1 984 [ BL TJ 841. Many books describe the user level interface, particularly how to use electronic mail, how to prepare documents, or how to use the command interpreter called the shell; some books such as The .UNIX Programming Environment [Kernighan 84] and Advanced UNIX Programming [Rochkind 85] describe the programming interface. This book describes the internal algorithms and structures that form the basis of the operating system (called the kernel) and their relationship to the programmer interface. It is thus applicable to several environments. First, it can be used as a textbook for an operating systems course at either the advanced undergraduate or first-year graduate level. It is most beneficial to reference the system source code when using the book, but the book can be read independently, too. Second, system programmers can use the book as a reference to gain better understanding of how the kernel works and to compare algorithms used in the UNIX system to algorithms used in other Op.!rating systems.

Finally, programmers on UNIX systems can gain a deeper understanding of how their programs interact with the system and thereby code more-efficient, sophisticated programs. The material and organization for the book grew out of a course that I prepared and taught at AT&T Bell Laboratories during 1 983 and 1 984. While the course centered on reading the source code for the system, I found that understanding the code was easier once the concepts of the algorithms had been mastered. I have attempted to keep the descriptions of algorithms in this book as simple as possible, reflecting in a small way the simplicity and elegance of the system it describes. Thus, the book is not a line-by-line rendition of the system written in English; it is a description of the general flow of the various algorithms, and most important, a description of how they interact with each other. Algorithms are presented in a Clike pseudo-code to aid the reader in understanding the natural language description, and their names correspond to the procedure names in the kernel. Figures depict the relationship between various data structures as the system manipulates them. In later chapters, small C programs illustrate many system concepts as they manifest themselves to users. In the interests of space and clarity, these examples do not usually check for error conditions, something that should always be done when writing programs. I have run them on System V; except for programs that exercise features specific to System V, they should run on other versions of the system, too. Many exercises originally prepared for the course have been included at the end of each chapter, and they are a key part of the book. Some exercises are straightforward, designed to illustrate concepts brought out in the text. Others are more difficult, designed to help the reader understand the system at a deeper level. Finally, some are exploratory in nature, designed for investigation as a research problem. Difficult exercises are marked with asterisks.

The system description is based on UNIX System V Release 2 supported by AT&T, with some new features from Release 3. This is the system with which I am most familiar, but I have tried to portray interesting contributions of other variations to the operating system, particularly those of Berkeley Software Distribution (BSD) . I have avoided issues that assume particular hardware characteristics, trying to cover the kernel-hardware interface in general terms and ignoring particula,r machine idiosyncrasies. Where machine-specific issues are important to understand implementation of the kernel, however, I delve into the relevant detail. At the very least, examination of these topics will highlight the parts of the operating system that are the most machine dependent. The reader must have programming experience with a high-level language and, Pfekrably, with an assembly language as l!- prerequisite for understanding this book. It is recommended that the reader have experience working with the UNIX system and that the reader knows the C language [Kernighan 781. However, I have attempted to write this book in such a way that the reader should still be able to. absorb the Q1aterial without such background. The appendix contains a simplified deS.crj�tion of the system calls, sufficient to understand the presentation PREFACE xiii in the book, but not a complete reference manual.

The book is organized as follows. Chapter 1 is the introduction, giving a brief, • general description of ·system features as perceived by the user an4 describing the system structure. Chapter 2 describes the general outline of the kernel architecture and presents some basic concepts. The remainder oft he boOk follow� the outline presented by the system architecture, describing the various components in a building block fashion. It can be divided into three parts: the file system, process control, and advanced topics. The' file system is presented first, because its concepts are easier than those for process control. Thus, Chapter 3 pescribes the system buffer cache mechanism that is the foundation of the file system. Chapter 4 describes the data structures and algorithms used internally by the file systetn. These algorithms use the algorithms explained in Ch'apter 3 and take care of the internal bookkeeping needed for managing user files. Chapter 5 describes the system calls that provide the user interface to the file system; they u9e the algorithms in Chapter 4 to access user files.

Chapter 6 turns to the control of processes. It defines the context of a process and investigates the internal kernel primitives that manipulate the process context. In particular, it considers the system call interface, interrupt handling, and the context switch. Chapter 7 presents the system calls that control the process context. Chapter 8 deals with process scheduling, and Chapter 9 tovers memory. management, including swapping and paging systems. Chapter 10 outlines general driver interfaces, with specific discussion of disk drivers and terminal drivers. Although devices are logically. part of the file system, their discussion is deferred until here because of issues in process control that arise in terminal drivers. This chapter also acts as a bridge to the more advanced t0pics presented in the rest of the book. Chapter 11 covers interprocess communication and networking, including System V messages, shared memory and �emaphores, and. BSD sockets. Chapter 12 explains tightly coupled multiprocessor UNIX systems, and Chapter 13 investigates loosely coupled distributed systems.·

The material in the first nine chapters could be covered in a one-semester course on operating systems, and the material in the remaining chapters- could be covered in advanced seminars with various projects being done in parallel. A few caveats must be made at this time. No attempt has been made to describe system performance in absolute terms, nor is there any attempt to suggest configuration parameters for a system installation. Such data is. likely to vary acCording to machine type, hardware configuration, system version and implementation, and application mix. Similarly, I have made a conscious effort to avoid predicting future development of UNIX operating system features. Discussion of advanced topics does not imply a commitment by AT&T to provide particular features, nor should it even imply that particular areas are uncier investigation.

1. is my pleasure to acknowledge the assistance of many friends and colleagues who �ncouraged me while I wrote this book. and provided constructive criticism of the· man'fscript. My 4eepest appreciation goes\_ to .Ian· Johnstone, who suggested· . � xit PREFACE that I write this book, gave me early encouragement, and reviewed the earliest draft of the first chapters. Ian taught me many tricks of the trade, and I will always be indebted to him. Doris Ryan also had a hand in encouraging me from the very begjnning, and I will always appreciate her kindness and thoughtfulness. Dennis Ritchie freely answered numerous questions on the historical and technical background of the system. Many people gave freely of their time and energy to review drafts of the manuscrip�. and this book owes' a lot to their detailed comments. They are Deb�y Bach, Doug Bayer, Lenny Brandwein, Steve Buroff, Tom Butle\_r, Ron Gomes, Mesut Gunduc, Laura Israel, D�an Jagels, Keith· Kelleman, Brian Kernighan, Bob Martin, Bob Mitze, Dave. Nowiti, Michael Poppers, Marilyn Safran, Curt Schimmel, Zvi Spitz, Tom Vaden, Bill Weber, Larry Wehr, and Bob Zarro� Mary Fruhstuck ·provided help in preparing the manuscript· for typesetting. I would like to thank my manag�ment for their continued' support throughout this project and my colleagues, for providing 'iuch a stimulating atmosphere and wonderful work environment at AT&T Bell Laboratories. John Wait and the staff at Prentice-Hall provided much valuable assitance and advice to get the book into its final form. Last, but not least, my wife, Debby, gave me lots of emotional support, without which I could never have succeeded.
2. GENERAL OVERVIEW OF THE SYSTEM The UNIX system has become quite popular since its inception in 1 969, running on machine$ of varying processing power from microprocessors to mainframes and providing a common execution environment across them. The system is divided into two parts. The first part consists of programs and services that have made the UNIX system environment so popular; it is the part readily apparent to users, including such programs as the shell, mail, text processing packages, and source code eontrol systems. The second part consists of the operating system that supports these programs and services. This book gives a detailed description of the operating system. It concentrates on a description of UNIX System V produced by AT&T but considers interesting features provided by other versions too. It examines the major data structures and algorithms used in the operating system that ultimately provide users with the standard user interface. This chapter provides an introduction to the UNIX system. It reviews its history clnd outlines the overall system structure. "rhe next chapter gives a rifore detailed introduction to the operating system. 1.1 HISTORY In 1965, Bell Telephone Laboratories joined an effort with the General Electric Compa'7 and Project MAC of the Massachusetts Institute of Technology to 1 2 GENERAL OVERVIEW OF THE SYSTEM develop a new operating system called Multics [Organick 72] The goals of the Multics system were to provide simultaneous computer access to a large community of users, to supply ample computation power and data storage, and to allow users to share their data easily, if desired. Many people who later took part in the early development of the UNIX system participated in the Multics work at Bell Laboratories. Although a primitive version of the Multics system was running on a GE 645 computer by 1 969, it did not provide the general service computing for which it was intended, nor was it clear when its development goals would be met. Consequently, Bell Laboratories ended its participation in the project. With the end of their work on the Multics project, members· of the Computing Science Research Center at Bell Laboratories were left without a "convenient interactive computing service" [Ritchie 84a1. In an attempt to improve their programming environment, Ken Thompson, Dennis Ritchie, and others sketched a paper design of a file. system that later evolved into an early version of the UNIX file system. Thompson wrote programs that simulated the behavior of the proposed file system and of programs in a demand-paging environment, and he even encoded a simple kernel (or the GE 645 computer. At the same time, he wrote a game program, "Space Travel," in Fortran for a GECOS system (the Honeywell 635) , but the program was unsatisfactory because it was difficult to control the "space ship" and the program was expensive to run. Thompson later found a little-used PDP-7 computer that provided good graphic display and cheap executing power. Programming "Space Travel" for the PDP-7 enabled Thompson to learn about the machine, but its environment for program development required cross-assembly of the program on tlle GECOS machine and carrying paper tape for input to the PDP-7. To create a better development environment, Thompson and Ritchie implemented their system design on the PDP-7, including an early version of the UNIX file system, the process subsystem, and a small set of utility programs. Eventually, the new system no longer needed the GECOS system as a development environment but could support itself. The new system was given the name UNIX, a pun on the name Multics coined by another member of the· Computing Science Research Center, Brian Kernighan.
3. Although this early version of the UNIX system held much promise, it could not realize its potential until it was used in a real project. Thus, while providing a text processing system for the patent department at Bell Laboratories, the UNIX '>ystem was moved to a PDP- 1 1 in 1 971. The system was characterized by its small size: 1 6K bytes for the system, 8K bytes for user programs, a disk of 512K bytes, and a limit of 64K bytes per file. After its early success, Thompson set out to implement a Fortran compiler for the new system, but instead came up with the language B, influenced by BCPL [Richards 691. B was an interpretive language with the performance drawbacks implied by such languages, so Ritchie developed it into one he called C, allowing generation of machine code, declaration of data types, and definition of data structures. In 1 973, the operating system was rewritten in C, an unheard of step at the time, but one that was to have tremendous impact on its acceptance among outside users. The number of installations at Bell 1.1 HISTORY 3 Laboratories grew to about 25, and a UNIX Systems Group was formed to provide internal support. At this time, AT&T coqJq not market computer products because of a 1 956 Consent Decree it had signed with the Federal government, but it provided the UNIX system to universities who requested it .for educational purposes. AT&T neither advertised, marketed, nor supported the system, in adherence to the terms of the Consent Decree. Nevertheless, the system's popularity steadily increased. In 1 974. Thompson and Ritchie published a paper describing the UNIX system in tli� Communications of the ACM [Thompson 74], giving further impetus to its acceptance. By 1 977, the number of UNIX system sites had grown to about 500, of which 125 were in universities. UNIX systems became popular in the operating telephone companies, providing a good environment for program development, network transaction operations services, and real-time services (via MERT [Lycklama 78a]) . Licenses of UNIX systems were provided to commercial institutions as well as universities. In 1 977, Interactive Systems Corporation became the first Value Added Reseller (V AR) 1 of a UNIX system, enhancing it for use in office automation environments. 1 977 also marked the year that the UNIX system was first "ported" to a non-PDP machine (that is, made to run on another machine with few or no changes) , the Interdata 8/32. With the growing popularity of mi�roprocessors, other companies ported the UNIX system to new machines, but its simplicity and clarity tempted many developers to enhance it in their own way, resulting in several variants of the basic system. In the period from 1 977 to 1 982, Bell Laboratories combined several AT&T variants into a single system, known commercially as UNIX System III. Bell Laboratories later added several features to UNIX System III, calling tlie new product UNIX System V/ and AT&T announced official support for System V in January 1 983. However,. people at the University of California at Berkeley had developed a variant to the UNIX system, the most rec�nt version of which is called 4.3 BSD for VAX machines; providing some new, interesting features. This book· will concentrate Oljl the description of UNIX System V and will occasionally talk about features prurided in the BSD system. By the beginm ng of H84, there were about 1 00,000 UNIX system installations in the world, ruJflmg on machines with a wide range of computing power from microprocessors to mainframes and on machines across different manufacturers' product lines. No other operating system can make that claim. Several reasons have been suggested for the popularity and success of the UNIX system.
4. • The system is written in a high-level language, making it easy to read, understand, change, and move to other machines. Ritchie estimates that the first system in C was 20 to 40 percent larger and slower because it was not written in assembly language, but the advantages of using a higher-level language far outweigh the disadvantages (see page 1965 of [Ritchie 78b]). • It has a simple user interface that has the power to provide the services that users want. • It provides primitives that permit complex programs to be built from simpler programs. • It uses a hierarchical file system that allows easy maintenance and efficient implementation. • It uses a consistent format for files, the byte stream, making application programs easier to write. • It provides a simple, consistent interface to peripheral devices. • It is a multi-user, multiprocess system; each user can execute several processes simultaneously. • It hides the machine architecture from the user, making it easier to write programs that run on diffet-ent hardware implementations. The philosophy of simplicity and -consistency underscores the UNIX system and ac·::ounts for many of the reasons cited above. Although the operating system and many of the command programs are written in C, UNIX systems support other languages, including Fortran, Basic, Pascal, Ada. Cobol, Lisp, and Prolog. The UNIX systrm can support any language that has a compiler or interpreter and a system interface that maps user requests for operating system services to the standard set of requests used on UNIX systems.
5. l.l SYSTEM STRUCI'URE Figure 1.1 depict!o. the high-level architecture gf the UNIX system. The hardware at the center of the diagram provides the operating system with basic services that will be described in SectiQn 1 .5. The operating system interacts directly3 with the hardwart·, providirg ·common services to programs and insulating them from hardware idiosyncrasies. Viewing the system as a set of layers, the operating system is commonly called the system kernel, or just the kernel, emphasizing its isolation from user programs. Because programs are independent of the underlying hardware, it is easy to move them between UNIX systems running on different hardware if the programs do not make assumptions about the underlying hardware. For instance, programs that assume the size of a machine word are more difficult to move to other machines than programs that do not make this assumption. Programs such as the shell and editors (ed and vi) shown in the outer layers interact with the kernel by invoking a well defined set of system calls. The system calls instruct the kernel to uo various operations for the calling program anc:l exchange data between the kernel and the program. Several programs shown in the figure are in standard system configurations and are known as COlflmands, but private dser programs may also exist in this layer as indicated by the program whose name is a. out, the standard name for executable files produced by the C compiler. Other application programs can build on top o( lower-level programs, hence the existence of the outermost layer in the figure. For example, the standar:d C compiler, cc, is in the outermost layer of the figure: it invokes a C preproctss6'r,, two-pass compiler, assembler, and loader Oink-editor), all separate lower-level programs. Although the figure depicts a two-level hierarchy of application programs, users can extend the hierarchy to whatever levels are appropriate. Indeed, the style of programming favored by the UNIX system encourages the combination of existing programs to accomplish a task. Many application subsystems and programs that provide a high-level view of the system such as the shell, editors, SCCS (Source Code Control System), and document preparation packages, have gradually become synonymous with the name "UNIX system." However, they all use lower-level services ultimately provided by the kernel, and they avail themselves of these services via the set of system calls. There are about 64 system calls in System V, of which fewer than 32 are used frequently. They have simple options that make them easy to use but provide the .user with a lot of power. The set of system calls and the internal algorithms that implement them form the body of the kernel, and the study of the UNIX operating system presented in .this book reduces to a detailed study and analysis of the system calls and their interaction with one another. In short, the kernel provides the services upon which all application programs in the UNIX system rely, and it defines those services. This book will frequently use the terms "UNIX system," "kernel," or "system," but the intent is to refer to the kernel of the UNIX operating system and should be clear in context.
6. 1.3 USER PERSPECTIVE This section briefly reviews high-level features of the UNIX system such as the file system, the processing environment, and building block primitives (for. example, pipes). Later chapters will explore kernel support of these features in detail. 1 .3.1 The File System The UNIX file system is characterized by • a hierarchical structure, • consistent treatment of file data, e 'the ability to create and delete files, • dynamic growth of files, • the protection of file data. • the treatment of peripheral devices (such as terminals and tape units) as files. The file system is organized as a tree with a single root node called root (written "/") ;'every non-leaf node of the file system structure is a directory of files, and files at the leaf nodes of the tree are either directories, regular files, or special device files. The name of a file is given by a path name that describes how to locate the file in the file system hierarchy. A path name is a sequence of component names separated by sl\_a.sh characters; a component is a sequence of characters that designates a file name that is uniquely contained in the previous (directory) component. A full path name starts with a slash character and specifies a ·file that can be found by starting at the file system root and traversing the file/ tree, following the branches that lead to successive component names of the path name. Thus, the path names "/etc/passwd", "/bin/who", and "/usr/src/cmd/who.c" designate files in the tree shown in Figure 1.2, but "/bin/passwd" and "/usr/src/date.c" do not. A path name does not have to start from root but can be designated relative to the current directory of an executing process, by omitting the initial slash in the path name. Thus, starting from directory "/dev", the path name "tty01" designates the file whose full path name is "/dev/tty01 ". Programs in the UNIX system have no knowledge of the internal format in which the kernel stores file data, treating the data as an unformatted stream of bytes. Programs may interpret the byte stream as they wish, but the interpretation has no bearing on how the operating system stores the data. Thus, the syntax of accessing the data in a file is defined by the system and is identical for all programs, but the semantics of the data are imposed by the program. For example, the text formatting program troff expects to find "new-line" characters at the end of each line of text, and the system accounting program acctcom expects to find fixed length records. Both programs use the same system services to access the data in the file as a byte stream, and internally, they parse the stream into a suitable format. If either program discovers that the format is incorrect, it is responsible for taking the appropriate action.
7. isolation from user programs. Because programs are independent of the underlying hardware, it is easy to move them between UNIX systems running on different hardware if the programs do not make assumptions about the underlying hardware. For instance, programs that assume the size of a machine word are more difficult to move to other machines than programs that do not make this assumption. Programs such as the shell and editors (ed and vi) shown in the outer layers interact with the kernel by invoking a well defined set of system calls. The system calls instruct the kernel to uo various operations for the calling program anc:l exchange data between the kernel and the program. Several programs shown in the figure are in standard system configurations and are known as COlflmands, but private dser programs may also exist in this layer as indicated by the program whose name is a. out, the standard name for executable files produced by the C compiler. Other application programs can build on top o( lower-level programs, hence the existence of the outermost layer in the figure. For example, the standar:d C compiler, cc, is in the outermost layer of the figure: it invokes a C preproctss6'r,, two-pass compiler, assembler, and loader Oink-editor), all separate lower-level programs. Although the figure depicts a two-level hierarchy of application programs, users can extend the hierarchy to whatever levels are appropriate. Indeed, the style of programming favored by the UNIX system encourages the combination of existing programs to accomplish a task. Many application subsystems and programs that provide a high-level view of the system such as the shell, editors, SCCS (Source Code Control System), and document preparation packages, have gradually become synonymous with the name "UNIX system." However, they all use lower-level services ultimately provided by the kernel, and they avail themselves of these services via the set of system calls. There are about 64 system calls in System V, of which fewer than 32 are used frequently. They have simple options that make them easy to use but provide the .user with a lot of power. The set of system calls and the internal algorithms that implement them form the body of the kernel, and the study of the UNIX operating system presented in .this book reduces to a detailed study and analysis of the system calls and their interaction with one another. In short, the kernel provides the services upon which all application programs in the UNIX system rely, and it defines those services. This book will frequently use the terms "UNIX system," "kernel," or "system," but the intent is to refer to the kernel of the UNIX operating system and should be clear in context. 1.3 USER PERSPECTIVE This section briefly reviews high-level features of the UNIX system such as the file system, the processing environment, and building block primitives (for. example, pipes). Later chapters will explore kernel support of these features in detail. 1 .3.1 The File System The UNIX file system is characterized by • a hierarchical structure, • consistent treatment of file data, e 'the ability to create and delete files, • dynamic growth of files, • the protection of file data. • the treatment of peripheral devices (such as terminals and tape units) as files. The file system is organized as a tree with a single root node called root (written "/") ;'every non-leaf node of the file system structure is a directory of files, and files at the leaf nodes of the tree are either directories, regular files, or special device files. The name of a file is given by a path name that describes how to locate the file in the file system hierarchy. A path name is a sequence of component names separated by sl\_a.sh characters; a component is a sequence of characters that

<https://www.computer.org/csdl/mags/co/1987/09/01663710.pdf>

Unix系统

Maurice Bach（t AT&T Bell

Laboratories in 1983 and 1984）

<https://cse.yeditepe.edu.tr/~kserdaroglu/spring2014/cse331/termproject/BOOKS/ProfessionalLinuxKernelArchitecture-WolfgangMauerer.pdf>

Linux

Wolfgang Mauerer

a retrieval system

<http://www.wiley.com/go/permissions>

John Wiley & Sons, Inc. and/or its affiliates, in the United States and

other countries

Wolfgang Mauerer

Quantum cryptography, quantum electrodynamics

typesetting with LaTeX

compiler

Plankalkul, the world’s earliest high-level language devised in 1942–1946 by Konrad Zuse

Linux kernel——commercially based

$ALL\_FILES\_COVERED\_IN\_THIS\_BOOK; do

git log --pretty="format:%an" $file; done |

sort -u -k 2,2

Carl Hanser Verlag in 2003

EAL4+ security

evaluation of Red Hat Enterprise Linux 5

Dr. Xiaodong Zhang

Consider Dennis Ritchie’s quote: Is the coinventor of Unix at Bell Labs completely right

in saying that only a genius can appreciate the simplicity of Unix? Luckily not, because he puts himself

into perspective immediately by adding that programmers also qualify to value the essence of Unix.

Benny Goodheart and James Cox

Unix System V

Brian W. Kernighan and

Denis M. Ritchie [KR88]

Design and Implementation, by Andrew S. Tanenbaum and

Albert S. Woodhull [TW06] on Unix (Minix) in particular

a handbook (The Unix-Haters Handbook,

edited by Simson Garfinkel et al. [GWS94])

[wm@linux-kernel.net](mailto:wm@linux-kernel.net)

[www.wrox.com](http://www.wrox.com)

ProfessionalLinuxKernelArchitecture-WolfgangMauerer.pdf

Red Hat Enterprise Linux 5

the first edition was published in German by Carl Hanser Verlag in 2003

Hewlett-Packard

Leslie MackayPoulton

HP and Red Hat people

Claudio Kopper and Hans Lohr

The Magic Garden Explained（the internals of Unix System V）

Additionally, Appendix C contains some information about extensions of the GNU C compiler that are

used by the kernel, but do not necessarily find widespread use in general programming.