Yukun Liu
Yong Yue
Liwei Guo

UNIX Operating System

The Development Tutorial via UNIX Kernel Services

Yukun Liu
Yong Yue
Liwei Guo

UNIX Operating System

The Development Tutorial via UNIX Kernel Services

Yukun Liu
Yong Yue
Liwei Guo

UNIX Operating System

The Development Tutorial via UNIX Kernel Services

With 132 figures

Authors
Associate Professor Yukun Liu
College of Information Science and Technology
Hebei University of Science and Technology
Hebei 050018, China
E-mail: lyklucky@hebust.edu.cn

Professor Yong Yue
Faculty of Creative Arts, Technologies and Science
University of Bedfordshire
Park Square Luton Bedfordshire
LU1 3JU, United Kingdom
E-mail: yong.yue@beds.ac.uk

Professor Liwei Guo
College of Information Science and Technology
Hebei University of Science and Technology
Hebei 050018, China
E-mail: guoliwei@hebust.edu.cn

ISBN 978-7-04-031907-1
Higher Education Press, Beijing

ISBN 978-3-642-20431-9 e-ISBN 978-3-642-20432-6
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011924668

© Higher Education Press, Beijing and Springer-Verlag Berlin Heidelberg 2011
This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.
The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This book presents UNIX as a practical case of operating systems for the reader to understand and master deeply and tangibly the theory and algorithms in operating systems. It gives discussions and studies on the hierarchical structure, principles, applications, shells, development, and management of the UNIX operation system multi-dimensionally, systematically and from the elementary to the profound. It brings readers to go into the inside of the UNIX operating system and lets them understand clearly what and how UNIX operating system functions.

Subject Matter

This book consists of 11 chapters. The first two chapters discuss the background of UNIX operating system (OS), and give a whole picture of what UNIX OS looks like and the evolution of UNIX.

Chapter 3 focuses on the editors that will be used frequently by UNIX users, no matter who are regular users or seasoned programmers.

Chapters 4, 5, 6, and 7 concentrate on the services of the UNIX kernel. Chapter 4 zooms in the process management, which is usually hidden from the final users. Chapter 5 is to discuss the UNIX memory management, which cooperates with the process management to accomplish the processes' concurrently running. Chapter 6 introduces the UNIX file management, which is involved almost in every service that the UNIX kernel provides to the users. Chapter 6, however, for UNIX users, is fundamentally useful to understand how UNIX works. Chapter 7 explores UNIX I/O, I/O redirection and piping. As UNIX treats its hardware devices as special files, this mechanism brings a whole different concept on UNIX input and output devices. I/O redirection and piping are two useful tools that can be used to deduce different commands to control the terminals through UNIX system calls.

UNIX has almost as many shells as versions of the UNIX operating system. Chapter 8 introduces some types of shells, shell evolution, and some common concepts in UNIX shells. As there are so many kinds of shells, it

is difficult to put all of them in one book. Hence, our strategy is to try to make one of them clear and integral in this book. Our choice is the primary shell for all other shells—Bourne shell. From this point, the readers can learn other shells by themselves from references. Therefore Chapters 9 and 10 focus on the discussion of Bourne shell as a programming language: Chapter 9 introduces basic facilities and Chapter 10 is for the advanced level.

Different from the studies in the previous chapters, which are concentrated on the local services and applications of UNIX in individual computers, Chapter 11 discusses the remote and network functions and services of UNIX in servers and workstations. Since the late 1960s, UNIX has had many original contributions to the development history of the computer networking and Internet.

Even though this book includes 11 chapters, it does not mean they are totally divided and irrelevant. Contrarily, they are coherent and relevant each other. Just like UNIX itself, its knowledge should be a link-up and cooperative "system". And we try very hard to unfold the information gradually and step by step in this book. When you, dear friends and readers, finish this book, you will have a relatively whole and systematical idea about UNIX. From that point, you can develop your applications on UNIX or other operating systems, or even build up a new operating system for a certain computer hardware system. This is just what the authors of this book really expect.

Historic and Active UNIX and Meaningfully UNIX Learning

As an open-source operating system, UNIX made its history during two decades of 1969 –1989. Maybe some say it has gone. However, UNIX's openness, which brought different groups of developers together to communicate their developing ideas and to respond feedback each other, really cultivated an excellent generation of operating system developers. We should remember these names: Dennis M. Ritchie, Ken Thompson, Robert S. Fabry, William N. Joy, Chuck Haley, Samuel J. Lefflerand, and more. The first two made their contribution to the premier UNIX System series and were honored by the ACM Turing Award in 1983 because of their work in UNIX, and the latter four did their great work on the primary UNIX BSD versions. Just as they made the earlier UNIX code open to the academic world and strived to move UNIX from one machine to another, UNIX grew and evolved. And even more, it left a lot of valuable academic papers about operating systems for the successors.

For its development process as an intact software system, UNIX, which presented solutions in detail, is unchallenged for generations of programmers. Compared to UNIX, its commercial counterparts usually provide a perfect environment that hides almost all the development details of lower levels of operating systems, which may leave a limited space for application program-

mers and also confine their imagination and creativity. This tendency can also affect the ability of system development newcomers to develop an intact software system that can handle software as well as hardware by restricting the field of vision to some detached modules or applications so as to result in software maintenance costly and complicated.

Further, just understanding the theory of operating systems, readers cannot image and understand well how the operating system works. With UNIX as a real case, readers can map the abstract algorithms, mechanisms and strategies of operating system theory into the real modules, functions and programs of UNIX implementation one-to-one. The abstract theory can be exemplified. In this way, as the promising programmers, readers can understand well and master these algorithms and mechanisms, practice them in their own development, and stimulate novel algorithms and mechanisms that may be more effective and efficient to their own context.

It seems as if a repetition of the old tale when considering the discussion on UNIX, which, all in all, reached its heyday around 1980s. In the latest two decades, however, due to commercial purposes and activities, there are no other operating systems like UNIX, which is so thoroughly open for the academic community to learn and do research.

In addition, there are plenty of references about UNIX that have been published, but most of them were originally published around 1980s. For the recent promising programmers, the published classics may be somewhat obscure because of the sparse context that might not be necessary for readers in those days but can be unfamiliar to nowadays readers. As the well-known rapid development of computer hardware in the latest decades, computer architecture and structure have made a big change. This change has also wielded a deep influence on the theories and concepts of computer, which makes the difficulty for recent readers to understand well descriptions and expressions in the published UNIX classics, and to map them properly into practical cases. It is possible to build an obstacle for readers to learn from them. Otherwise, for the operating system construction, which belongs to software developments but resides the one of the most exciting and integrated of software development, it would be a pity and defect if losing an operational means. Fortunately, this means can be gained by doing research on UNIX.

It is taken that UNIX has its own philosophy and several items in the philosophy are written in different references. If having the right, we can say that the most important one should be the UNIX programmers' dedication and passion to their work. UNIX is also deemed to a programmer's OS. UNIX programmers have done a wonderful work just as for tackling a necessary affair, from which others else really benefit. It is critical for the academic community.

UNIX benefited also from those days. If AT&T, at that time, could market computer products without a 1956 Consent Decree signed with the Federal Government, and if Bell Laboratories did not withdraw Ken Thompson and others from the MULTICS project, and if Professor Robert S. Fabry of the

University of California at Berkeley did not contact Ken Thompson at the Symposium on Operating Systems Principles at Purdue University in November 1973, we would have a totally different story about UNIX. It needs the open and free soil to breed an academic activity. The more relieved the outside environment is, the more natural the academic activity develops within the environment. UNIX was destined for being flourishing in its day.

Even though being just observers on this period of history, the authors of this book are impressed by the passion and concentration that UNIX developers had in the day. During five years of teaching UNIX in their campuses, the authors realized that if this fantastic piece of history was not introduced to more readers, it would be a pity for authors as well as readers. In this high-technology and high-material-civilization age, UNIX development process can give readers some new inspiration—a glowing motivation from inside to accomplish a meaningful work.

A General Survey of UNIX Development

Observing different versions of UNIX emerging, the authors and readers can discover that it is a process of constant development, amendment and enhancement. In this process, UNIX developers' thoughts were adjusted and enriched with the development of computer hardware and peripherals, and the proposal of new application demands. It resulted in UNIX' s being moved to execute on different hardware platforms and fitting in different projects, which also naturally made UNIX' s portability and scalability practice and reinforce repeatedly and concretized the concepts of portability and scalability in operating system theory.

UNIX drew on a lot of ideas of the earlier operating systems, and its openness made the idea-drawing expand into different UNIX versions and different groups of developers. For a new programmer, it is also necessary to derive the precursors' development thoughts and experiences. Only if learning from the precursors, newcomers can enrich their knowledge gradually and effectively, and the novel thinking can just grow from thick knowledge reserves.

For promising developers, the UNIX development process was also a training program. Linux is a successful example of UNIX derivatives. Through this training program with deducing mentally and programming physically, developers can get familiar with the computer system as a whole, including both hardware and software.

With the advent of commercial operating systems, most of the readers do their jobs on encapsulated and transparent operating systems. On the other hand, many students and graduate students of computer disciplines mostly start their studies from the theory of operating systems. A transparent, well-designed and inextricable operating system seems like saving the users a lot of time and effort, but it also cuts the exploring road towards the inside

of operating systems and the underlying hardware parts. For real developers and programmers, it may take a big risk to sit on a seemingly-transparent but unfamiliar system to do their developments—finally they may encounter some bugs that they cannot tackle. They have to experience something that can let them understand what really make the construction of an operating system, what the kernel of an operating system does for users, and how algorithms and mechanisms in the theory of operating systems are implemented. Even though the disassembled UNIX cannot tell all the story of a well-designed modern or future operating system, it can give the mapping or clues to different functions and services, which can be treated as an anatomy lecture of a promising surgeon.

In other words, a well-designed operating system may be daunting for a promising developer, which is complicated and confused. The simplicity and clarity of UNIX can help readers walk out of the swamp and sort out the confusion, and lead them to face and tackle more sophisticated problems.

Targets and Strategy of this Book

Knowledge needs to renew and information needs to update. The updating includes the expression of a convincing, successful and classical process in a proper, timely and new way. Maybe the UNIX story is old, but it can give different inspirations to people in different ages, which is still developing. The authors hope the developing can be reflected in this book.

One of the targets of this book is to let the UNIX philosophy propagate and carry on. Let more readers comprehend this philosophy's results—the fast development, maintainability and scalability of an operating system.

The authors also want to present readers (especially, programmers) two aspects of a whole operating system and any one of its subsystems, in other words, to give not only the inside implementation process, which is viewed by the system or application programmers, but also the outside application performance, which is usually felt by the end users. In this way, readers cannot only keep the view of the system constructors but also be considerate of the end users when developing their systems. During development, a system can benefit from that its developers can consider more for its end users.

For readers, it is easy to enter the learning from user interfaces of UNIX operating systems since they have usually had the experience of using one of operating systems in their daily works or lives. Thus, in this book, we take this strategy: when starting one topic, we present it from its user interface, and then go into the kernel with the system calls and lower-level algorithms if possible. In this way, readers can be brought from a familiar environment into elusive and deep techniques.

To describe algorithms, we try to use common English language rather than some computer language, such as C or assembly language. The primary

reason is: we try to make algorithms more readable and help readers save their time and effort. For a programmer, it is often time-consuming to read some code that is written by others.

Intended Audience

This book is written for the wide groups of readers who want to master the professional knowledge of operating systems through a real and open-source case. Its main readers include graduates, senior undergraduates and teachers of computer and software majors, and potential researchers on applicable computing and engineering modeling. The readers can also be ones who maybe have some or have not much knowledge related to Computer Science and Technology and Software Engineering, but have a strong interest in these fields and want to get into them quickly, acquire some useful and important knowledge and reach an advanced level in the relevant fields after learning. This book can help readers construct, not only as the users of operating systems but also in the view of the operating system designers, the knowledge on the UNIX operating system, and even on other kinds of operating systems. From this point, readers can build up their projects on an operating system. Or on this basis, readers can go deep into how UNIX and other operating systems to be designed and programmed because many versions of UNIX are open-source code, like Linux, and adjust and modify the operating systems on their own computer systems.

For readers whose mother tongues are not English, it may be more difficult to read and learn an English edition of the academic book than books written with their mother tongue. However, it is necessary for readers to have the ability to read the English editions of academic books, especially for computer and software professionals, because most papers on the advanced and update science and technology are written in English, especially in the field of computer hardware and software. Why not to try to gain this ability just from your learning process? Maybe it is difficult for you now. But nothing is easy when starting it. Maybe when you finish this book, you say, “It is not that hard, is it?” So try it now.

Yukun Liu
Yong Yue
Liwei Guo
January 2011

Acknowledgements

This book is funded by Academic Work Publication Fund of Hebei University of Science and Technology.

The authors of this book would like to give their thanks to Ms. Hongying Chen, Editor of High Education Press in China, who gave generously of her time and expertise to edit this book.

Contents

1 Background of UNIX Operating System · · · · · 1
1.1 Introduction of Operating System · · · · · 1
1.2 Types of UNIX · · · · · 3
1.3 History of UNIX · · · · · 4
1.4 Summary · · · · · 6
Problems · · · · · 7
References · · · · · 7

2 How to Start · · · · · 9
2.1 UNIX Software Architecture · · · · · 9
2.1.1 UNIX Kernel · · · · · 10
2.1.2 System Call Interface · · · · · 12
2.1.3 Standard Libraries and Language Libraries · · · · · 14
2.1.4 UNIX Shell · · · · · 14
2.1.5 Applications · · · · · 14
2.2 UNIX Environment · · · · · 15
2.3 Character User Interface Versus Graphical User Interface · · · · · 16
2.4 UNIX Command Lines · · · · · 17
2.4.1 UNIX Command Syntax · · · · · 18
2.4.2 Directory Operation Commands · · · · · 19
2.4.3 File Operation Commands · · · · · 24
2.4.4 Displaying Online Help · · · · · 30
2.4.5 General Utility Commands · · · · · 32
2.4.6 Summary for Useful Common Commands · · · · · 34
2.5 UNIX Window Systems · · · · · 35
2.5.1 Starting X · · · · · 35
2.5.2 Working with a Mouse and Windows · · · · · 36
2.5.3 Terminal Window · · · · · 37

2.5.4 Using a Mouse in Terminal Windows · · · 37
2.6 Shell Setup Files · · · 38
2.7 Summary · · · 40
Problems · · · 41
References · · · 43

3 Text Editors · · · 45
3.1 Difference Between Text Editors and Word Processors · · · 45
3.2 Introduction of Pico Editor · · · 46
3.2.1 Start pico, Save File, Exit pico · · · 47
3.2.2 Create a New File with Pico · · · 48
3.2.3 Cursor-moving Commands in Pico · · · 49
3.2.4 General Keystroke Commands in Pico · · · 50
3.3 The vi Editor and Modes · · · 52
3.3.1 Three Modes of the vi and Switch Between Them · · · 52
3.3.2 Start vi, Create a File, Exit vi · · · 53
3.3.3 Syntax of the vi Commands · · · 55
3.4 Practicing in Insert Mode of the vi Editor · · · 56
3.5 Practicing in Command Mode and Last Line Mode of the vi Editor · · · 62
3.6 Using Buffers of the vi Editor · · · 65
3.7 The vi Environment Setting · · · 67
3.8 Introduction of the emacs Editor · · · 69
3.8.1 Start emacs, Create File, Exit emacs · · · 70
3.8.2 Buffers, Mark and Region in emacs · · · 71
3.8.3 Cursor Movement Commands · · · 72
3.8.4 Keyboard Macros · · · 73
3.8.5 Search and Replace · · · 73
3.8.6 Operation Example · · · 74
3.8.7 Programming in emacs · · · 76
3.9 Summary · · · 77
Problems · · · 77
References · · · 79

4 UNIX Process Management · · · 81
4.1 Multiple Processes' Running Concurrently · · · 81
4.1.1 Fundamental Concept for Scheduler and Scheduling Algorithm · · · 83
4.1.2 UNIX Scheduling Algorithm and Context Switch · · · 84
4.2 Process States · · · 86

4.2.1 Fundamental Concept for Process States · · · · · · · · · · · 87
4.2.2 UNIX Process States · 88
4.3 Process Image and Attributes · 90
4.3.1 UNIX Process Attributes in Kernel · · · · · · · · · · · · · · · 90
4.3.2 UNIX Process Attributes from User Angle · · · · · · · · · · 91
4.4 Creating a Process in UNIX · 94
4.4.1 Fork System Call · 94
4.4.2 How UNIX Kernel to Execute Shell Commands · · · · · · 96
4.5 Process Control · 99
4.5.1 Running Command in Foreground or in Background · 100
4.5.2 More Concepts about Process Concurrently Execution in UNIX · 104
4.5.3 UNIX Inter-Process Communication · · · · · · · · · · · · · · 106
4.5.4 UNIX Signals · 110
4.5.5 Termination of Processes · 112
4.5.6 Daemons — UNIX Background “Guardian Spirits” · · · · 115
4.6 UNIX System Boot and Init Process · · · · · · · · · · · · · · · · · · 116
4.7 Summary · 118
Problems · 120
References · 121

5 UNIX Memory Management · 123
5.1 Outline of Memory Management · 123
5.1.1 Evolution of Memory Management · · · · · · · · · · · · · · · 124
5.1.2 Memory Allocation Algorithms in Swapping · · · · · · · · 126
5.1.3 Page Replacement Algorithms in Demand Paging · · · · 127
5.2 Process Swapping in UNIX · 130
5.2.1 Swapped Content · 130
5.2.2 Timing of Swapping · 131
5.2.3 Allocation Algorithm · 132
5.2.4 Selection Principle of Swapped Processes · · · · · · · · · · · 133
5.2.5 Swapper · 133
5.2.6 Swapping Effect · 135
5.3 Demand Paging in UNIX · 135
5.3.1 Demand Paging · 136
5.3.2 Page Replacement · 142
5.4 Summary · 145
Problems · 146

References · · · 147

6 UNIX File System · · · 149
6.1 UNIX File System Structure · · · 149
6.1.1 File System Organization · · · 150
6.1.2 Home and Working Directories · · · 153
6.1.3 Absolute and Relative Pathnames · · · 153
6.1.4 UNIX Inodes and Data Structures for File System · · · 154
6.2 UNIX File Concept and Types of Files · · · 155
6.2.1 Types of Files · · · 155
6.2.2 Ordinary Files · · · 155
6.2.3 Directories · · · 157
6.2.4 Special Files · · · 157
6.2.5 Pipes · · · 158
6.2.6 Sockets · · · 158
6.2.7 Link Files · · · 159
6.3 Managing Files and Directories · · · 159
6.3.1 Displaying Pathname for Home Directory and Changing Directories · · · 160
6.3.2 Viewing Directories and File Attributes · · · 161
6.3.3 Creating Directories and Files · · · 164
6.3.4 Displaying Type of a File · · · 165
6.3.5 Making Lines in File Ordered · · · 165
6.3.6 Searching Strings in Files · · · 168
6.3.7 The eof and CRTL-D · · · 169
6.4 File and Directory Wildcards · · · 170
6.5 UNIX File Storage and File System Implementation · · · 171
6.5.1 File System Physical Structure and Allocation Strategies · · · 171
6.5.2 Inode, Inode List and Inode Table · · · 174
6.5.3 Disk Physical Structure and Mapping Pathname to Inode · · · 175
6.5.4 File Descriptors · · · 177
6.5.5 System Calls for File System Management · · · 178
6.5.6 Standard Files · · · 179
6.6 Summary · · · 180
Problems · · · 182
References · · · 185

7 UNIX I/O System, I/O Redirection and Piping · · · · · · · · · 187
7.1 Standard Input and Output, Standard Files · · · · · · · · · · · · · 187
7.1.1 Standard Input and Output · 188
7.1.2 Standard Input, Output and Error Files · · · · · · · · · · · 188
7.2 Input Redirection · 189
7.2.1 Input Redirection with < Operator · · · · · · · · · · · · · · · 190
7.2.2 Input Redirection with File Descriptor · · · · · · · · · · · · 191
7.3 Output Redirection · 191
7.3.1 Output Redirection with > Operator · · · · · · · · · · · · · 192
7.3.2 Creating a File with Output Redirection · · · · · · · · · · 193
7.3.3 Output Redirection with File Descriptor · · · · · · · · · · · 193
7.4 Appending Output Redirection · 194
7.4.1 Appending Output Redirection with >> Operator · · · · 194
7.4.2 Appending Output Redirection with the File Descriptor · 195
7.5 Standard Error Redirection · 195
7.5.1 Error Redirection by Using File Descriptor · · · · · · · · · 196
7.5.2 Appending Error Redirection by Using File Descriptor · 197
7.5.3 Error Redirection in C Shell · · · · · · · · · · · · · · · · · · · 197
7.6 Combining Several Redirection Operators in One Command Line · 198
7.6.1 Combining Input and Output Redirections in One Command Line · 199
7.6.2 Combining Output and Error Redirections in One Command Line · 200
7.6.3 Combining Input, Output and Error Redirections in One Command Line · 202
7.6.4 Combining Appending Redirection with Other Redirections in One Command Line · · · · · · · · · · · · · 203
7.7 UNIX Pipes and Filters · 203
7.7.1 Concepts of Pipe and Filter · · · · · · · · · · · · · · · · · · · 204
7.7.2 Examples of Pipes and Filters · · · · · · · · · · · · · · · · · · 205
7.7.3 Combining Pipes and I/O Redirections in One Command Line · 205
7.7.4 Practical Examples of Pipes · · · · · · · · · · · · · · · · · · · 207
7.7.5 Pipes in C Shell · 208
7.7.6 Named Pipes · 209
7.8 UNIX Redirection and Pipe Summary · · · · · · · · · · · · · · · · · 212

7.9 I/O System Implementation in UNIX · · · 213
7.9.1 I/O Mechanisms in UNIX · · · 213
7.9.2 Block Special Files and Buffer Cache · · · 216
7.9.3 Character Special Files and Streams · · · 219
7.9.4 Sockets for Networks in UNIX · · · 223
7.10 Summary · · · 224
Problems · · · 226
References · · · 227

8 UNIX Shell Introduction · · · 229
8.1 Variety of UNIX Shells · · · 229
8.1.1 Shell Evolution · · · 230
8.1.2 Login Shell · · · 231
8.2 UNIX Shell as a Command Interpreter · · · 231
8.2.1 Shell Internal and External Commands · · · 232
8.2.2 Shell's Interpreting Function · · · 232
8.2.3 Searching Files Corresponding to External Commands · · · 233
8.3 Environment Variables · · · 234
8.3.1 Some Important Environment Variables · · · 234
8.3.2 How to Change Environment Variables · · · 235
8.3.3 Displaying the Current Values of Environment Variables · · · 236
8.4 Switching Between UNIX Shells · · · 236
8.4.1 Why to Change Shell · · · 236
8.4.2 How to Change Shell · · · 237
8.4.3 Searching for a Shell Program · · · 238
8.5 Shell Metacharacters · · · 239
8.6 Summary · · · 241
Problems · · · 242
References · · · 242

9 How to Program in Bourne Shell (1) · · · 245
9.1 Bourne Shell Scripts · · · 245
9.1.1 Simplified Structure of Bourne Shell Scripts · · · 246
9.1.2 Program Headers and Comments · · · 247
9.1.3 Exit Command · · · 248
9.2 Shell Variables · · · 248
9.3 Bourne Shell Variable Commands · · · 250
9.3.1 Reading Shell Variables · · · 250

9.3.2 Assignment Statement · · · 251
9.3.3 Resetting Variables · · · 253
9.3.4 Exporting Variables · · · 253
9.3.5 Making Variables Read-only · · · 256
9.3.6 Reading Standard Input · · · 256
9.4 Shell Scripts' Argument Transport · · · 258
9.4.1 Shell Positional Parameters · · · 258
9.4.2 Setting Values of Positional Parameters · · · 259
9.4.3 Shift Command · · · 261
9.5 How to Execute a Bourne Shell Script · · · 262
9.5.1 Setting File Access Permissions · · · 262
9.5.2 One Way to Make Bourne Shell Script Executable · · · 266
9.5.3 Another Way to Make Bourne Shell Script Executable · · · 267
9.6 Program Control Flow Statement (a): if Statement · · · 267
9.6.1 The Simplest if Statement · · · 268
9.6.2 The test Command · · · 269
9.6.3 The if Statement with the else Keyword · · · 272
9.6.4 Integral Structure of if Statement · · · 274
9.7 Program Control Flow Statement (b): for Statement · · · 276
9.7.1 The for Statement with a Word List · · · 276
9.7.2 The for Statement without a Word List · · · 278
9.8 Summary · · · 280
Problems · · · 281
References · · · 283

10 How to Program in Bourne Shell (2) · · · 285
10.1 Program Control Flow Statement (c): case Statement · · · 285
10.2 Program Control Flow Statement (d): while Statement · · · 288
10.3 Program Control Flow Statement (e): until Statement · · · 291
10.4 Program Control Flow Statement (f): break and continue Commands · · · 293
10.4.1 The break Command · · · 293
10.4.2 The continue Command · · · 295
10.5 Processing Numeric Data · · · 297
10.6 The exec Command · · · 299
10.6.1 Execution Function of the exec Command · · · 300
10.6.2 Redirection function of the exec Command · · · 301
10.7 Bourne Shell Functions · · · 307

10.7.1 Defining Function · · · 308
10.7.2 Calling Function · · · 309
10.8 How to Debug Shell Scripts · · · 310
10.9 Summary · · · 312
Problems · · · 313
References · · · 314

11 UNIX in Internet and Computer Networking · · · 315
11.1 UNIX's Contribution to Development of Computer Networking · · · 315
11.2 General Concepts of Computer Networks and Internet · · · 316
11.2.1 Request for Comments · · · 316
11.2.2 Computer Networks and Internet · · · 317
11.2.3 Client-server vs Peer-to-peer Models · · · 319
11.2.4 TCP/IP and ISO models · · · 322
11.2.5 TCP/IP Protocol Suite · · · 324
11.3 Encapsulation and Demultiplexing · · · 330
11.4 Networking Operating Systems · · · 332
11.5 IP Protocol · · · 334
11.5.1 IP Header · · · 334
11.5.2 IPv4 Addressing · · · 337
11.5.3 IPv4 Routing · · · 338
11.5.4 Two commands, ping and traceroute · · · 341
11.6 TCP Protocol and Applications · · · 344
11.6.1 TCP Segment, Header and Services · · · 344
11.6.2 TCP Applications · · · 347
11.7 UDP Protocol and Applications · · · 358
11.7.1 UDP Protocol · · · 358
11.7.2 UDP Applications · · · 359
11.8 Summary · · · 360
Problems · · · 361
References · · · 363

Index · · · 365

1 Background of UNIX Operating System

Before presenting the UNIX operating system, we describe the fundamental purpose of operating systems, different types of operating systems, and which type of operating systems UNIX belongs to. Then we discuss the development history of the UNIX, along with introducing different types of the UNIX.

1.1 Introduction of Operating System

An operating system (or OS) is usually sitting between the user's application programs and the computer hardware, which exploits the computer hardware resources to provide a set of services to the computer system users. The hierarchical view of a typical computer system is shown in Figure 1.1.

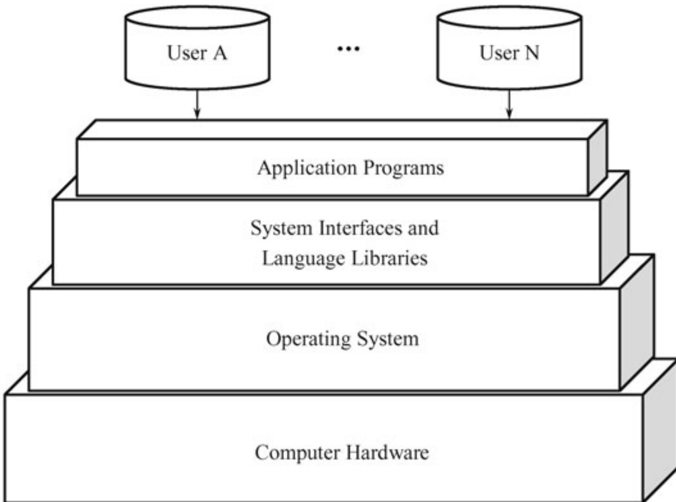

Fig. 1.1 The hierarchical view of a computer system.

The operating system functions as an interface between a user and a com-

puter. Usually, a common user is concerned about the applications rather than the architecture of computer. The application programs are developed by application programmers who develop the programs generally without responsibility of how to control the computer hardware. The reason beneath this is that there is an overwhelmingly complex task related to the computer hardware control, especially when it comes to the portability of the application programs from one computer system to another different one. To ease this task, a set of system programs should be built. Some of these system programs are frequently used as facilities, which implement functions, such as the file creation and management, I/O devices management, etc. An application programmer can use those functions to develop an application program. And once it is developed and executed in the environment where there are system programs, the application program can call some of the system programs to perform its own functions. And it is just the operating system that consists of a set of these system programs to take the responsibility of the details of the solution to the control and management of different computer resources, and builds a convenient software environment and interface for the application programmers to develop their applications for end users of computer systems.

Usually, the operating system provides some typical services for its users:

- Execute a program: When a program is executed, the operating system must load the instructions and data of the program into the main memory, initialize files and I/O devices, and prepare some other resources for its need.
- Create a program: The operating system should provide platforms and tools, such as editors, to assist the programmer in creating, editing and debugging programs.
- Operate files: For file management, the operating system must know not only the types of different files but also the characters of different storage devices. It also should provide the share and protection mechanisms of files.
- Control I/O devices: As there are so many types of I/O devices in the computer system and each of them has its own control instructions, the operating system must control I/O devices accurately when the application programs need them.
- Manage system and users: For multi-user computer systems, the operating system can not only let its users share system resources but also protect system resources, including CPU, primary memory, I/O devices and data, in order to make the execution of the operating system and applications smooth and proper. It also needs to make an account for each user in the system in order to collect some usage statistics and monitor the system performance.
- Detect and respond errors: As some errors can occur when the computers running, the operating system must detect errors and give an appropriate response in time. Different errors should be tackled in different ways, so

the operating system should give a variety of responses, such as reporting the error, retrying the operation, canceling the operation, or terminating the program that causes the error.

The UNIX operating system has solutions to the above services, which will be discussed detailed in the following chapters.

Operating systems are usually classified as three kinds according to numbers of users and the processes that operating systems can handle simultaneously.

- Single-user and single-process operating systems: Only one user at a time can use this kind of operating systems. And the user can run only one process at a time. Many older operating systems belong to this kind, such as MS-DOS, MacOS (Katzan 1970; Quarterman 1985; Stallings 1998; Tanenbaum 2005).
- Single-user and multiprocessing operating systems: Only one user at a time can use this kind of operating systems. But the user can run more-than-one processes at a time. Microsoft Windows XP Workstation is an example of this kind (Tanenbaum 2005).
- Multi-user and multiprocessing operating systems: More-than-one users at a time can use this kind of operating systems. And each of the users can run more-than-one processes at a time. They have examples, like UNIX, Linux, Microsoft Windows NT Server and Windows 2003 Server (Tanenbaum 2005).

On the other hand, as the evolution of operating systems, a contemporary operating system should have some major features, such as a multi-processing batch system and a time-sharing system. A multiprocessing batch system allows several programs to run concurrently, automatically without human intervention. UNIX and Linux execute programs in multi-processing batch mode, by running programs in the background (see Section 4.5.1). In a time-sharing system, several users access the system through different terminals simultaneously. So the operating system needs to switch the executing of each user program in a short period of computation in order to give a sole-user-in-system impression to each of the users. UNIX, Linux, Microsoft Windows NT Server, and Microsoft Windows 2003 Server are examples of the time-sharing systems.

1.2 Types of UNIX

In fact, the types of the UNIX operating system are almost countless (Kim 1999; Mann 1992; Martin 1995; Mckusick et al 2005; Perrone 1991; Perrone 1993; Riggs 1995). Since 1969, when Ken Thompson, Dennis Ritchie and others started working on an idle PDP-7 at AT&T Bell Labs and developed what was to become UNIX (Cooke 1999; McKusick 1999; Ritchie et al 1974; Sarwar 2006), many groups of developers and programmers, no matter who

came from the companies for commercial purposes or who were from universities for academic reasons, have been involved in different stages of UNIX development.

In the early 1970s, there was UNIX Time-Sharing System firstly (Cooke 1999; McKusick 1999; Ritchie 1974; Sarwar 2006). Since then, it has split into several branches. Even though there are assorted branches and some of their contributors moved from one branch to another or combined with others, two of the most influential are the System variant and BSD (Berkeley Software Distribution) variant. The former is the UNIX Time-Sharing System (Nieh et al 1997; Ritchie et al 1974), including six editions from System I to System VI. Even though at the very beginning it was developed just for programmers' own, this branch is more likely for the commercial purpose later, whose owner was AT&T. The latter was originated from the former because of one of the former developers, Ken Thompson, who installed one of UNIX operating system to a PDP-11/70 at the University of California, Berkeley (McKusick 1999). This branch tends more towards academic activities.

Linux is one of the popular, free members of the UNIX operating system. Another free derivative was 386BSD, which also led to FreeBSD (McKusick et al 2005), OpenBSD, and NetBSD. Now the UNIX operating system is widely applied in both servers and workstations. Linux and BSD are also expanding their application fields into such as consumer desktops, mobile phones, and embedded devices.

From 2007, the trademark UNIX® has been owned by the Open Group, an industry standards consortium, and only systems that are fully compliant with and certified to the Single UNIX Specification are qualified as "UNIX®" while others are called "Unix system-like" or "Unix-like". However, this result should be just for the commercial purpose. Thus, no matter whether it is "UNIX®" or "Unix-like", in this book, for academic research, it will be called one of the UNIX operating system, which may be more likely following the UNIX philosophy.

1.3 History of UNIX

As there are so many different types of the UNIX operating system and so many contributors involved into the UNIX development as well, it is necessary to learn the development history of the UNIX operating system. It can give a different angle to see how the UNIX development thoughts, which are usually brought into the basic theory of the textbooks of operating systems, influence the development of the whole operating systems.

No matter how many branches the UNIX operating system has, they have the same performances that are also called as the UNIX philosophy (Bach 2006; McKusick 1999; McKusick et al 2005; Quarterman et al 1985). These performances include: the use of plain text for data storage (see Section 3.1);

a hierarchical file system structure (see Section 6.1); treating devices as files (see Section 6.2); treating sockets that can be used inter-process communication as files (see Section 6.2); and the use of a large number of software tools that can be batched together through a command line interpreter by using pipes, creating a chain of producer-consumer processes (see Chapter 7); and UNIX shell which can be used to add new commands without changing the shell itself. Except these, the UNIX operating system is portable, multi-tasking and multi-user in a time-sharing configuration.

In the mid-1960s, AT&T Bell Labs worked on a collaborative project — an experimental operating system called Multics (Multiplexed Information and Computing Service). After AT&T Bell Labs pulled out of the project, one of the Multics developers, Ken Thompson, continued his development and led to start a new operating system for the PDP-7 at AT&T Bell Labs (Sarwar et al 2006; Thompson 1978). With the help of Dennis Ritchie, Ken Thompson and his team developed a multi-tasking operating system with a file system, which was the very beginning of the UNIX operating system.

During the 1970s, UNIX Time-Sharing System I to VI was developed sequentially. At that time, one breakthrough for UNIX and also for operating systems was to rewrite the UNIX operating system in the C programming language (Quarterman et al 1985; Ritchie et al 1974; Thompson 1978), contrary to the assembly language that, at that time, was considered as the general tool for such complex projects as an operating system, which must deal with time-critical events. And this also meant the UNIX operating system more portable — requiring only rewriting a relatively small amount of machine-dependent code in order to port the UNIX operating system from one computer system to another.

Another shinning idea, during that time, was related to modularizing the program code, which was resulted in scalability and expandability of the UNIX operating system.

In November 1973, the first UNIX paper of Ken Thompson and Dennis Ritchie was presented at the Symposium on Operating Systems Principles at Purdue University. During the conference, Professor Bob Fabry of the University of California at Berkeley obtained a copy of the UNIX system to test with at Berkeley. About one year later, Ken Thompson decided to take a one-year sabbatical as a visiting professor at the University of California at Berkeley, his alma mater, and brought up the latest UNIX, Version VI, to install in the PDP-11/70 at Berkeley. Later on, William N. Joy, Chuck Haley, Samuel J. Lefflerand, and others at Berkeley involved in the development of UNIX BSD versions. Meanwhile, Bob Fabry got a contract from Defense Advanced Research Projects Agency (DARPA) to add some features to BSD in order to meet the DARPA needs.

At the same time, the UNIX operating system went into two directions, one for the commercial use and the other for the academic purpose. The former was represented by UNIX System III and System V of AT&T; the latter were mainly Berkeley Software Distribution (BSD) releases of UNIX

developed at the University of California, Berkeley. During this time, BSD researchers brought to the UNIX operating system a few features such as the vi editor, C shell with job control, TCP/IP network code, and Berkeley Sockets application programming interface (API).

In the 1980s, some other companies started to develop their UNIX operating system for their own computers, such as Sun Microsystems' SunOS (Solaris later on), Microsoft's Xenix (SCO's SCO UNIX later on).

At the end of the 1980s, AT&T's SVR4 (System V Release 4) added new features, such as file locking, system administration, new forms of interprocess communication (IPC), the Remote File System, etc. And the UNIX-related standard — the IEEE's Portable Operating System Interface for UNIX POSIX (Portable Operating System Interface for UNIX) specification was established (Jespersen 1995; Walli 1995).

In the 1990s, splitting from BSD developers, William Jolitz led his team to develop the 386BSD, which was the free software pioneer for FreeBSD, OpenBSD, and NetBSD. At the same time, AT&T sold all its rights to UNIX to Novell, which developed the UnixWare. Linux, which was a typical UNIX-like system, was released as free software in 1992 and soon after became a popular operating system.

In the 2000s, many free UNIX operating systems are developed flourishingly, such as NetBSD and FreeBSD. And Sun Microsystems released its open-source OpenSolaris (German et al 2010). And most of the programs and scripts in this book have been debugged and passed on FreeBSD.

1.4 Summary

An operating system (OS) is usually in between the user's application programs and the computer hardware, which operates the computer hardware resources to provide a set of services for the computer system users. Usually, the operating system provides some typical services for its users: execute a program, create a program, operate files, control I/O devices, manage system and users, and detect and respond errors.

Operating systems usually have three types according to the numbers of the users and the processes that operating systems can handle simultaneously: single-user and single-process operating systems, single-user and multiprocessing operating systems, and multi-user and multiprocessing operating systems. UNIX, Linux, Microsoft Windows NT Server, and Windows 2003 Server belong to the multi-user and multiprocessing operating systems.

Among the UNIX operating systems, two of the most influential are System III and V variant and BSD variant. The former is more likely for the commercial use, whose owner was AT&T; the latter tends more towards academic use, whose place of origin was the University of California, Berkeley. Linux is one of the popular, free members of the UNIX operating system.

No matter how many branches the UNIX operating system has, the UNIX characters are the same, including: the use of plain text for data storage, a hierarchical file system structure, treating devices as files, treating sockets as files, the use of a large number of small programs that can be batched together through a command line interpreter using pipes, creating a chain of producer-consumer processes, and UNIX shell which can be used to add new commands without changing the shell itself. In addition, the UNIX operating system is portable, multi-tasking and multi-user in a time-sharing configuration.

Problems

Problem 1.1 What is an operating system? What typical services can operating systems provide?

Problem 1.2 How many kinds can operating systems usually be classified, according to the numbers of the users and the processes that operating systems can handle simultaneously? What are they? Which kind does the UNIX operating system belong to?

Problem 1.3 What are the two of the most influential UNIX operating systems?

References

Bach MJ (2006) The design of the UNIX operating system. China Machine Press, Beijing

Cooke D, Urvan J, Hailton S (1999) UNIX and beyond: An interview with Ken Thompson. Computer 32(5): 58 – 64

German DM, Penta MD, Davies J (2010) Understanding and auditing the licensing of open source software distributions. IEEE 18th International Conference on Program Comprehension, Braga, Minho Portugal, June 2010: pp 84 – 93

Jespersen H (1995) POSIX retrospective. ACM, StandardView 3(1): 2 – 10

Katzan H Jr (1970) Operating system architecture. AFIPS: Spring Joint Computer Conference, Atlantic City, New Jersey, 5 – 7 May 1970: pp109 – 118

Kim T, Shin G, Hong E (1999) Experience with porting a UNIX version case tool to the corresponding Java version. APSEC'99: IEEE sixth Asia-Pacific software engineering conference, Takamatsu, December 1999: pp 622 – 629

Lawrence DC (1998) Internetnews server: Inside an open-source project. IEEE Internet Comput 2(5): 49 – 52

Mann D (1992) UNIX and the Am29000 microprocessor. IEEE Micro 12(1): 23 – 31

Martin VC (1995) There can be only one! A summary of the UNIX standardization movement. ACM, Crossroads 1(3): 9 – 11

McKusick MK (1999) Twenty years of Berkeley Unix: from AT&T-owned to freely redistributable. LINUXjunkies.org. http://www.linuxjunkies.org/articles/kirkmck.pdf. Accessed 20 Aug 2010

McKusick MK, Neville-Neil GV (2005) The design and implementation of FreeBSD operating system. Addison-Wesley, Boston

Nieh J, Lam MS (1997) Smart UNIX SVR4 support for multimedia applications. ICMCS'97: The IEEE International conference on multimedia computing and systems, Ottawa, Ontario, Canada, June 1997: pp 404 – 414

Perrone G (1991) Alternate versions of UNIX. Computer 24(11): 84 – 90

Perrone G (1993) The Macintosh A/Ux operating system release 3.0. Computer 26(2): 103 – 106

Quarterman JS, Silberschatz A, Peterson JL (1985) Operating systems concepts, 2nd edn. Addison-Wesley, Reading, Massachusetts

Riggs B, English E (1995) UNIX transformations. Computer 28(4): 8 – 9

Ritchie DM, Thompson K (1974) The Unix time-sharing system. Commun ACM 17(7): 365 – 375

Sarwar SM, Koretesky R, Sarwar SA (2006) UNIX: The textbook, 2nd edn. China Machine Press, Beijing

Stallings W (1998) Operating systems: Internals and design principles 3rd edn. Prentice Hall, Upper Saddle River, New Jersey

Tanenbaum AS (2005) Modern operating systems, 2nd edn. China Machine Press, Beijing

Thompson K (1978) UNIX Implementation. Bell Syst Tech J 57(6), part 2: 1931 – 1946

Walli SR (1995) The POSIX family of standards. ACM, StandardView 3 (1): 11 – 17

2 How to Start

Before using the UNIX operating system, a brief introduction of the components and software structure of UNIX will be given. And it is also necessary for the users to know the working environment in UNIX in order to start to work in there and use its facilities. After learning the difference between a character user interface (CUI) and a graphical user interface (GUI) and also benefits of a CUI, it is helpful to learn the UNIX command lines and UNIX window systems. A UNIX shell is an important interface between the user and the operating system, and also a programming tool for users to write their own commands. This chapter will introduce only the shell setup files, and the detailed discussion on UNIX shells will be given in Chapter 8.

2.1 UNIX Software Architecture

Figure 2.1 shows a layered view for a UNIX-based computer system, identifying the system's software components and their logical relationship with the user and hardware. Following will be given each software layer from the bottom up.

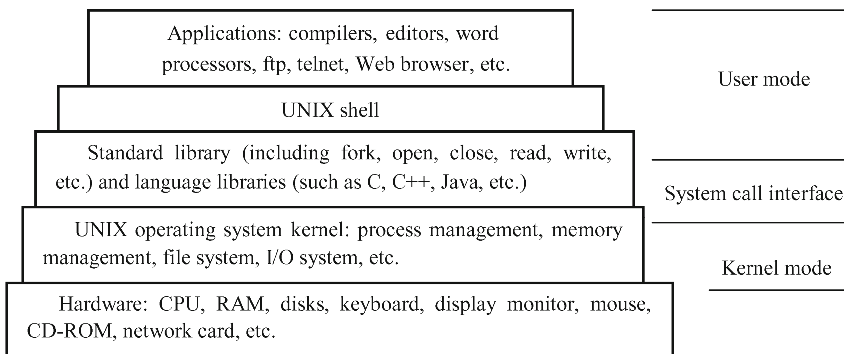

Fig. 2.1 The UNIX software architecture.

2.1.1 UNIX Kernel

The UNIX operating system kernel is the core of the system (Bach 2006; McKusick et al 2005; Nguyen et al 1997; Quarterman et al 1985; Spinellis 2008; Stallings 1998). Its main functions are shown in Figure 2.1, which are the process management, memory management, file system, and I/O system. Many application subsystems and programs that provide a high-level performance of the system, such as the shell and editors, have gradually been seen as features of the UNIX operating system. However, they all use lower-level services ultimately provided by the kernel, and they avail themselves of these services via a set of system calls.

2.1.1.1 Process Management

A program is an executable file while a process is an instance of the program in execution. Processes have “life” (Braams 1995; McKusick et al 2005; Stallings 1998). When it is created with the fork system call, a process is brought to be active in the system; when it is terminated for some reasons, it is dying out from the system. Many processes can execute concurrently on the UNIX operating system (this feature is also called multiprogramming or multitasking) with no limit to their number logically. Some system calls allow processes to create new processes, terminate processes, synchronize stages of process execution, and control reaction to various events.

In a UNIX system, the only active entities are the processes. Each process runs a single program and initially has a single thread of control. Process management of the UNIX operating system is typically responsible for tasks: to create, suspend and terminate processes, to switch their states, to schedule the CPU to execute multiple processes concurrently in a time-sharing system, and to manage the communication between processes (inter-process communication, IPC).

Pipes are one of IPC mechanisms of the UNIX operating system. A pipe is a temporary unidirectional channel in the main memory, which is created by the kernel for two processes, and into which one process can write a stream of bytes for the other to read. In other words, by using a pipe, two producer-consumer processes can be connected together so that the output from the producer becomes the input of the consumer in order to transfer data between them in the first-in-first-out (FIFO) way. Synchronization is attained because when a process tries to read from an empty pipe it is blocked until data are sent by the other.

Compared to the temporary pipe, a permanent channel that is located on the disk is called a named pipe which can be used for IPC by two or more processes that are running on the same computer.

Signals can be used for handling exceptional conditions, such as a keyboard interrupt, an error in a process, or a number of asynchronous events. Almost all the signals can be generated by the kill system call. With signals,

it is implemented to control processes asynchronously with events.

2.1.1.2 Memory Management

This part of kernel job is to keep track of which parts of memory are in use and which parts are not in use, to allocate memory to processes when they need it and deallocate it when they finish their tasks and put this memory space back in the free space pool so that it can be reallocated, and to manage swapping some processes between main memory and disk when main memory is not enough to hold all the processes (Ritchie et al 1974). It protects the memory space of a process from entrance by others. It also manages the share memory space for processes.

With a demand-paged virtual memory system (Bach 2006; McKusick et al 2005; Quarterman et al 1985; Rashid et al 1988), the kernel can implement paging for processes without size limitations. In other words, the kernel can load its related parts into main memory dynamically and execute it even though other portions are not in memory. In this way, more tasks can be kept in memory and swapping can be reduced to a minimum.

2.1.1.3 File System

Files are also managed by the kernel. How they are structured, named, accessed, used, protected, and implemented is handled by the file system of the kernel. File systems are stored on disks. Usually, in directories, there are files and sub-directories. In UNIX, a directory is also a file. The kernel manages files and directories (also called folders) by performing to create and remove files and directories, to change file and directory attribute, to protect files and directories according to their attribute and access modes, and to share files among several users, etc.

In UNIX, except directories and ordinary files, there are also special device files, sockets, pipes, and links kept in the file system (Bach 2006; Badger et al 1995; Quarterman et al 1985; Sarwar et al 2006). Thus users can access devices in the same way as to reference ordinary files via the file system.

In the storage, each file has its unique inode. When a file is accessed by a user, the kernel needs to load its inode from the disk into memory and give the user a file descriptor directing to its inode.

Sockets are another type of facilities in the UNIX kernel, which are used for a general interface, such as networking. Links, otherwise, are different references towards a certain file.

2.1.1.4 I/O System

In UNIX, all I/O devices are made to look like files and are accessed as such with the same read and write system calls that are used to access all ordinary files. I/O devices are those connected to the computers, such as, disks, printers, CD-ROM, keyboard, display, and network cards. UNIX operating systems integrate the devices into the file system as what are called special files (Carson et al 1992; Heindel et al 1995; Nelson et al 1996). Each I/O