Chapter 1 Introduction

Abstract

Chapter 1 presents an introduction to the book. It describes the book's scope, intended audience and its suitability as textbook in Computer Science/Engineering curriculums. It presents a brief history of Unix, which includes early versions of Unix at Bell Labs, AT&T System V and other developments of Unix, such as BSD, HP UX, IBM AIX and Sun/Solaris Unix. It describes the development of Linux and various Linux distributions, which include Debian, Ubuntu, Mint, Red Hat and Slackware. It lists both the hardware platforms and virtual machines for Linux. It shows how to install Ubuntu Linux to both VirtualBox and Vmware virtual machines inside the Microsoft Windows. It explains the startup sequence of Linux, from booting the Linux kernel to user login and command execution. It describes the Linux file system organization, file types and commonly used Unix/Linux commands. Lastly, it describes some system administration tasks for users to manage and maintain their Linux systems.

1.1. About This Book

This book is about systems programming in the Unix/Linux environment [Thompson and Ritchie, 1974, 1978, Bach, 1986, Linux, 2017]. It covers all the essential components of Unix/Linux, which include process management, concurrent programming, timer and time service, file systems, network programming and MySQL database system. In addition to covering the functionalities of Unix/Linux, it stresses on programming practice. It uses programming exercises and realistic programming projects to allow students to practice systems programming with hands-on experiences.

1.2. Roles of Systems Programming

Systems programming is an indispensable part of Computer Science and Computer Engineering education. System programming courses in Computer Science/Engineering curriculums serve two main purposes. First, it provides students with a wide range of knowledge about computer system software and advanced programming skills, allowing them to interface with operating system kernel, make efficient use of system resources and develop application software. Second, it prepares students with the needed background to pursue advanced studies in Computer Science/Engineering, such as operating systems, embedded systems, database systems, data mining, artificial intelligence, computer networks, network security, distributed and parallel computing.

1.3. Objectives of This Book

This book is intended to achieve the following objectives.

1.3.1. Strengthen Students Programming Background

Computer Science/Engineering students usually take introductory programming courses in their first or second year. Such courses typically cover basic programming in C, C++ or Java. Due to time limitations, these courses can only cover the basic syntax of a programming language, data types, simple program structures and using library functions for I/O. Students coming out of such course are rather weak in programming skills due to a lack of sufficient practices. For instance, they usually do not know much about the software tools, the steps of program development and run-time environment of program executions. The first object of this book is to provide students with the needed background and skills to do advanced programming. This book covers program development steps in detail. These include assembler, compiler, linker, link library, executable file contents, program execution image, function call conventions, parameter passing schemes, local variables, stack frames, link C program with assembly code, program termination and exception handling. In addition, it also shows how to use Makefiles to manage large programming projects and how to debug program executions by GDB.

1.3.2. Applications of Dynamic Data Structures

Most elementary data structure courses cover the concepts of link lists, queues, stacks and trees, etc. but they rarely show where and how are these dynamic data structures used in practice. This book covers these topics in detail. These include C structures, pointers, link lists and trees. It uses programming exercises and programming projects to let the students apply dynamic data structures in real life. Case in point: a programming exercise is for the students to display the stack contents of a program with many levels of function calls, allowing them to see the function calling sequence, passed parameters, local variables and stack frames. Another programming exercise is to let students implement a printf-like function with varying parameters, which requires them to apply parameter passing schemes to access varying parameters on stack. Yet another programming exercise is to print the partitions of a disk, in which the extended partitions form a “link list” in the disk, but not by the conventional pointers as they learned in programming languages. A programming project is to implement a binary tree to simulate the Unix/Linux file system tree, which supports pwd, ls, cd, mkdir, rmdir, creat, rm operations as in a real file system. Besides using dynamic data structures, the programming project allows them to apply a wide range of programming techniques, such as string tokenization, search for nodes in a tree, insert and delete tree nodes, etc. It also allows them to apply binary tree traversal algorithms to save binary trees as files and reconstruct binary trees from saved files.

1.3.3. Process Concept and Process Management

The major focus of systems programming is the abstract notion of processes, which is rather hard to grasp. This book uses a simple C program together with a piece of assembly code to let students see real processes in action. The program implements a multitasking environment to simulate process operations in an OS kernel. It allows the students to create processes, schedule processes by priority, run different processes by context switching, maintain process relations by a binary tree, use sleep and wakeup to

implement wait for child process termination, etc. This concrete example makes it much easier for students to understand processes and process management functions. Then it covers process management in Unix/Linux in detail, which includes fork(), exit(), wait() and change process execution image by exec(). It also covers I/O redirection and pipes. It uses a programming project to let the students implement a sh simulator for command execution. The sh simulator behaves exactly the same as the bash of Linux. It can execute all Unix/Linux commands with I/O redirections, as well as compound commands connected by pipes.

1.3.4. Concurrent Programming

Parallel computing is the future of computing. It is imperative to introduce parallel computing and concurrent programming to CS/CE students at an early stage. This book covers parallel computing and concurrent programming. It explains the concept of threads and their advantages over processes. It covers Pthreads [Pthreads, 2017] programming in detail. It explains the various kinds of threads synchronization tools, such as threads join, mutex lock, condition variables, semaphores and barriers. It demonstrates applications of threads by practical examples, which include matrix computation, quicksort and solving system of linear equations by concurrent threads. It also introduces the concepts of race conditions, critical regions, deadlock and deadlock prevention schemes. In addition to using Pthreads, it uses a programming project to let students experiment with user-level threads, implement threads synchronization tools and practice concurrent programming by user-level threads.

1.3.5. Timer and Time Functions

Timer and time functions are essential to process management and file systems. This book covers the principle of hardware timers, timer interrupts and time service function in Unix/Linux in detail. It shows how to use interval timers and timer generated signals in systems programming. In addition, it uses a programming project to let students implement interval timer support for concurrent tasks.

1.3.6. Signals, Signal Processing and IPC

Signals and signal processing are keys to understanding program exceptions and Inter-Process Communication (IPC). This book covers signals, signal processing and IPC in Unix/Linux. It explains signal origins, signal delivery and processing in the OS kernel and the relation between signals and exceptions. It shows how to install signal catchers to handle program exceptions in user mode. It uses a programming project to let students use Linux signals and pipes to implement an IPC mechanism for concurrent tasks to exchange messages.

1.3.7. File system

Besides process management, file system is the second major component of an operating system. This book provides an in depth coverage of file systems. It describes

file operations at different levels and explains their relationships. These include storage devices, file system support in Unix/Linux kernel, system calls for file operations, library I/O functions, user commands and sh scripts. It uses a series of programming exercises to let the students implement a complete EXT2 file system that is totally Linux compatible. These not only allow the students to have a thorough understanding of file operations but also provide them with the unique experience of writing a complete file system. Another programming project is to implement and compare the performances of different I/O buffer management algorithms. The simulation system supports I/O operations between processes and disk controllers. It allows the students to practice I/O programming, interrupts processing and synchronization between processes and I/O devices.

1.3.8. TCP/IP and Network Programming

Network programming is the third major component of an operating system. This book covers TCP/IP protocols, socket API, UDP and TCP programming using sockets and the server-client model in network computing. A programming project is for the students to implement a server-client based system for remote file operations across the Internet. As of now, WWW and Internet services have become an indispensable part of daily lives. It is essential to introduce CS/CE students to this technology. This book covers HTTP and Web programming in detail. It allows the students to configure the HTTPD server on their Linux machines to support HTTP programming. It shows how to use HTML for static Web pages and PHP to create dynamic Web pages. It covers the MySQL database system, shows how to create and manage databases in both command and batch modes and interface MySQL with both C and PHP. A programming project is for the students to implement a Web server to support file operations by HTTP forms and dynamic Web pages using CGI programming.

1.4. Intended Audience

This book is intended as a textbook for systems programming courses in technically oriented Computer Science/Engineering curriculums that emphasize both theory and programming practice. The book contains many detailed working example programs with complete source code. It is also suitable for self-study by advanced programmers and computer enthusiasts.

1.5. Unique Features of This Book

This book has many unique features which distinguishes it from other books.

1. This book is self-contained. It includes all the foundation and background information for studying systems programming. These include program development steps, program execution and termination, function call conventions, run-time stack usage and link C programs with assembly code. It also covers C structures, pointers and dynamic data structures, such as link lists, queues and binary trees. It shows applications of dynamic data structures in system programming.

2. It uses a simple multitasking system to introduce and illustrate the abstract notion of processes. The multitasking system allows students to create processes, schedule and run different processes by context switching, implement process synchronization mechanisms, control and observe process executions. This unique approach allows the students to have a concrete feeling and better understanding of process operations in real operating systems.

3. It describes the origin and roles of processes in Unix/Linux in detail, from system booting to INIT process, daemon processes, login processes and the user sh process. It describes the execution environment of user processes, such as stdin, stdout, stderr file streams and environment variables. It also explains how does the sh process execute user commands.

4. It covers process management functions in Unix/Linux in detail. These include fork(), exit(), wait(), changing process execution image by exec(), I/O redirection and pipes. Moreover, it allows the students to apply these concepts and programming techniques to implement a sh simulator for command execution. The sh simulator behaves exactly the same as the standard bash of Linux. It supports executions of simple commands with I/O redirections, as well as compound commands connected by pipes.

5. It introduces parallel computing and concurrent programming. It covers Pthreads programming by detailed examples, which include matrix computation, quicksort and solving systems of linear equations by concurrent threads, and it demonstrate the use of threads synchronization tools of join, mutex lock, condition variables and barriers. It also introduces the important concepts and problems in concurrent programming, such as race conditions, critical regions, deadlock and deadlock prevention schemes. In addition, it allows the students to experiment with user-level threads, design and implement threads synchronizing tools, such as threads join, mutex lock, semaphores and demonstrate the capabilities of user-level threads by concurrent programs. These allow the students to have a deeper understanding how threads work in Linux.

6. It covers timers and time service functions in detail. It also allows the reader to implement interval timers for concurrent tasks in a multitasking system.

7. It presents a unified treatment of interrupts and signals, treating signals as interrupts to processes, similar to interrupts to a CPU. It explains signal sources, signal types, signal generation and signal processing in great detail. It shows how to install signal catchers to allow processes to handle signals in user mode. It discusses how to use signals for IPC and other IPC mechanisms in Linux. It uses a programming project to let the students use Linux signals and pipes to implement an IPC mechanism for tasks to exchange messages in a multitasking system.

8. It organizes file operations into a hierarchy of different levels, ranging from storage devices, file system support in Unix/Linux kernel, system calls for file operations, library I/O functions, user commands and sh scripts. This hierarchical view provides the reader

with a complete picture of file operations. It covers file operations at the various levels in detail and explains their relationships. It shows how to use system calls to develop file operation commands, such as ls for listing directory and file information, cat for displaying file contents and cp for copying files.

8. It uses a programming project to lead the students to implement a complete EXT2 file system. The programming project has been used in the CS360, System Programming course, at Washington State University for many years, and it has proved to be very successful. It provides students with the unique experience of writing a complete file system that not only works but is also totally Linux compatible. This is the most unique feature of this book.

9. It covers TCP/IP protocols, socket API, UDP and TCP programming using sockets and the server-client model in network programming. It uses a programming project to let the students implement a server-client based system for remote file operations across the Internet. It also covers HTTP and Web programming in detail. It shows how to configure the Linux HTTPD server to support HTTP programming, and how to use PHP to create dynamic Web pages. It uses a programming project to let the students implement a Web server to support file operations via HTTP forms and dynamic Web pages using CGI programming.

10. It emphasizes on programming practice. Throughout the book, it illustrates system programming techniques by complete and concrete example programs. Above all, it uses a series of programming projects to let students practice system programming. The programming projects include

(1). Implement a binary tree to simulate the Unix/Linux file system tree for file operations (Chapter 2).
(2). Implement a sh simulator for command executions, including I/O redirections and pipes (Chapter 3).
(3). Implement user-level threads, threads synchronization and concurrent programming (Chapter 4).
(4). Implement interval timers for concurrent tasks in a multitasking system (Chapter 5).
(5). Use Linux signals and pipes to implement an IPC mechanism for concurrent task to exchange messages (Chapter 6).
(6). Convert file pathname to file's INODE in an EXT2 file system (Chapter 7).
(7). Recursive file copy by system calls (Chapter 8).
(8). Implement a printf-like function for formatted printing using only the basic operation of putchar() (Chapter 9).
(9). Recursive file copy using sh functions and sh scripts (Chapter 10).
(10). Implement a complete and Linux compatible EXT2 file system (Chapter 11).
(11). Implement and compare I/O buffer management algorithms in an I/O system simulator (Chapter 12).
(12). Implement a file server and clients on the Internet (Chapter 13).
(13). Implement a file server using HTTP forms and dynamic Web pages by CGI programming (Chapter 13).

Among the programming projects, (1)-(3), (6)-(9), (12)-(13) are standard programming assignments of the CS360 course at Washington State University. (10) has been the class project of the CS360 course for many years. Many of the programming projects, e.g. (12)-(13) and especially the file system project (10), require the students to form two-person teams, allowing them to acquire and practice the skills of communication and cooperation with peers in a team-oriented working environment.

1.6. Use This Book as Textbook in Systems Programming Courses

This book is suitable as a textbook for systems programming courses in technically oriented Computer Science/Engineering curriculums that emphasize both theory and practice. A one-semester course based on this book may include the following topics.

1. Introduction to Unix/Linux (Chapter 1)
2. Programming background (Chapter 2).
3. Process management in Unix/Linux (Chapter 3)
4. Concurrent programming (Parts of Chapter 4)
5. Timer and time Service functions (Parts of Chapter 5)
6. Signals and signal processing (Chapter 6)
7. File operation levels (Chapter 7)
8. File system support in OS kernel and system calls for file operations (Chapter 8)
9. I/O library functions and file operation commands (Chapter 9)
10. Sh programming and sh scripts for file operations (Chapter 10)
11. Implementation of EXT2 file system (Chapter 11)
12. TCP/IP and network programming, HTTP and Web programming (Chapter 13).

The problems section of each chapter contains questions designed to review the concepts and principles presented in the chapter. Many problems are suitable as programming projects to let students gain more experience and skills in systems programming.

Most materials of this book have been used and tested in the Systems Programming course, CS360, in EECS at Washington State University. The course has been the backbone of both Computer Science and Computer Engineering programs in EECS at WSU. The current CS360 course syllabus, class notes and programming assignments are available for review and scrutiny at

http://www.eecs.wsu.edu/~cs360

The course project is to lead students to implement a complete EXT2 file system that is totally Linux compatible. The programming project has proved to be the most successful part of the CS360 course, with excellent feedbacks from both industrial employers and academic institutions.

1.7. Other Reference Books

Due to its importance, systems programming in Unix/Linux has been a popular subject matter in CS/CE education and also for self-study by advanced programmers. As a result, there are a tremendous number of books in this area. Some of these books are listed in the reference section [Curry, 1996, Haviland, et al.1998, Kerrisk, 2010, Love, 2013, Robins, et al. 2003, Rochkind, 2008, Stevens and Rago, 2013]. Many of these books are excellent programmer's guides, which are suitable as additional reference books.

1.8. Introduction to Unix

Unix [Thompson and Ritchie, 1974, 1978] is a general purpose operating system. It was developed by Ken Thompson and Dennis Ritchie on the PDP-11 minicomputers at Bell Labs in the early 70's. In 1975, Bell Labs released Unix to the general public. Initial recipients of this Unix system were mostly universities and non-profit institutions. It was known as the **V6 Unix.** This early version of Unix, together with the classic book on the C programming language [Kernighan and Ritchie, 1988] started the Unix revolution on Operating Systems, with long-lasting effects even to this day.

1.8.1. AT&T Unix

Development of Unix at AT&T continued throughout the 1980's, cumulating in the release of the **AT&T System V** Unix [Unix System V, 2017], which has been the representative Unix of AT&T. System V Unix was a uniprocessor system. It was extended to multiprocessor versions in the late 80's [Bach, 1986].

1.8.2. Berkeley Unix

Berkeley Unix [Leffler et al., 1989, 1996] refers to a set of variants of the Unix operating system developed by the Berkeley Software Distribution **(BSD)** at the University of California, Berkeley, from 1977 to 1985. The most significant contributions of BSD Unix are implementation of the TCP/IP suite and the socket interface, which are incorporated into almost all other operating systems as a standard means of networking, which has helped the tremendous growth of the Internet in the 90's. In addition, BSD Unix advocates open source from the beginning, which stimulated further porting and development of BSD Unix. Later releases of BSD Unix provided a basis for several open source development projects, which include FreeBSD [McKusick et al., 2004], OpenBSD and NetBSD, etc. that are still ongoing to this day.

1.8.3. HP Unix

HP-UX [UP-UX, 2017] is Hewlett Packard's proprietary implementation of the Unix operating system, first released in 1984. Recent versions of HP-UX support the HP 9000 series computer systems, based on the PA-RISC processor architecture, and HP Integrity systems, based on Intel's Itanium. The unique features of HP-UX include a built-in logical volume manager for large file systems and access control lists as an alternative to the standard rwx file permissions of Unix.

1.8.4. IBM Unix

AIX [IBM AIX, 2017] is a series of proprietary Unix operating systems developed by IBM for several of its computer platforms. Originally released for the IBM 6150 RISC workstation, AIX now supports or has supported a wide variety of hardware platforms, including the IBM RS/6000 series and later POWER and PowerPC-based systems, IBM System I, System/370 mainframes, PS/2 personal computers, and the Apple Network Server. AIX is based on UNIX System V with BSD4.3-compatible extensions.

1.8.5. Sun Unix

Solaris [Solaris, 2017] is a Unix operating system originally developed by Sun Microsystems [Sun OS, 2017]. Since January 2010, it is renamed **Oracle Solaris.** Solaris is known for its scalability, especially on SPARC systems. Solaris supports SPARC-based and x86-based workstations and servers from Oracle and other vendors.

As can be seen, most Unix systems are proprietary and tied to specific hardware platforms. An average person may not have access to these systems. This present a challenge to readers who wishes to practice systems programming in the Unix environment. For this reason, we shall use Linux as the platform for programming exercises and practice.

1.9. Introduction to Linux

Linux [Linux, 2017] is a Unix-like system. It started as an experimental kernel for the Intel x86 based PCs by Linus Torvalds in 1991. Subsequently, it was developed by a group of people world-wide. A big milestone of Linux occurred in the late 90's when it combined with **GNU** [Stallman, 2017] by incorporating many GNU software, such as the GCC compiler, GNU Emacs editor and bash, etc. which greatly facilitated the further development of Linux. Not long after that, Linux implemented the TCP/IP suite to access the Internet and ported X11 (X-windows) to support GUI, making it a complete OS.

Linux includes many features of other Unix systems. In some sense, it represents a union of the most popular Unix systems. To a large extent, Linux is POSIX compliant. Linux has been ported to many hardware architectures, such as Motorola, SPARC and ARM, etc. The predominate Linux platform is still the Intel x86 based PCs, including both desktops and laptops, which are widely available. Also, Linux is free and easy to install, which makes it popular with computer science students.

1.10. Linux Versions

The development of Linux kernel is under the strict supervision of the Linux kernel development group. All Linux kernels are the same except for different release versions. However, depending on distributions, Linux has many different versions, which may differ in distribution packages, user interface and service functions. The following lists some of the most popular versions of Linux distributions.

1.10.1. Debian Linux

Debian is a Linux distribution that emphasizes on free software. It supports many hardware platforms. Debian distributions use the .deb package format and the dpkg package manager and its front ends.

1.10.2. Ubuntu Linux

Ubuntu is a Linux distribution based on Debian. Ubuntu is designed to have regular releases, a consistent user experience and commercial support on both desktops and servers. Ubuntu Linux has several official distributions. These Ubuntu variants simply install a set of packages different from the original Ubuntu. Since they draw additional packages and updates from the same repositories as Ubuntu, the same set of software is available in all of them.

1.10.3. Linux Mint

Linux Mint is a community-driven Linux distribution based on Debian and Ubuntu. According to Linux Mint, it strives to be a "modern, elegant and comfortable operating system which is both powerful and easy to use". Linux Mint provides full out-of-the-box multimedia support by including some proprietary software, and it comes bundled with a variety of free and open-source applications. For this reason, it was welcomed by many beginner Linux users.

1.10.4. RPM-based Linux

Red Hat Linux and SUSE Linux were the original major distributions that used the RPM file format, which is still used in several package management systems. Both Red Hat and SUSE Linux divided into commercial and community-supported distributions. For example, Red Hat Linux provides a community-supported distribution sponsored by Red Hat called Fedora, and a commercially supported distribution called Red Hat Enterprise Linux, whereas SUSE divided into openSUSE and SUSE Linux Enterprise

1.10.5. Slackware Linux

Slackware Linux is known as a highly customizable distribution that stresses ease of maintenance and reliability over cutting-edge software and automated tools. Slackware Linux is considered as a distribution for advanced Linux users. It allows users to choose Linux system components to install and configure the installed system, allowing them to learn the inner workings of the Linux operating system.

1.11. Linux Hardware Platforms

Linux was originally designed for the Intel x86 based PCs. Earlier versions of Linux run on Intel x86 based PCs in 32-bit protected mode. It is now available in both 32-bit and 64-bit modes.

In addition to the Intel x86, Linux has bee ported to many other computer architectures, which include MC6800 of Motorola, MIP, SPARC, PowerPC and recently ARM. But the predominant hardware platform of Linux is still the Intel x86 based PCs, although ARM based Linux for embedded systems are gaining popularity rapidly.

1.12. Linux on Virtual Machines

Presently, most Intel x86 based PCs are equipped with Microsoft Windows, e.g. Windows 7 or 10, as the default operating system. It is fairly easy to install Linux alongside Windows on the same PC and use dual-boot to boot up either Windows or Linux when the PC starts. However, most users are reluctant to do so due to either technical difficulty or preference to stay in the Windows environment. A common practice is to install and run Linux on a virtual machine inside the Windows host. In the following, we shall show how to install and run Linux on virtual machines inside the Microsoft Windows 10.

1.12.1. VirtualBox

VirtualBox [VirtualBox, 2017] is a powerful x86 and AMD64/Intel64 virtualization product of Oracle. VirtualBox runs on Windows, Linux, Macintosh, and Solaris hosts. It supports a large number of guest operating systems, including Windows (NT 4.0, 2000, XP, Vista, Windows 7, Windows 8, Windows 10) and Linux (2.4, 2.6, 3.x and 4.x), Solaris and OpenSolaris, OS/2, and OpenBSD. To install virtualBox on Windows 10, follow these steps.

(1). Download VirtualBox.

At the VirtualBox website, **http://download.virtualbox.org,** you will find links to VirtualBox binaries and its source code. For Windows hosts, the VirtualBox binary is

VirtualBox-5.1.12-112440-win.exe

In addition, you should also download the

VirtualBox 5.1.12 Oracle VM VirtualBox Extension Pack

which provides **s**upport for USB 2.0 and USB 3.0 devices, VirtualBox RDP, disk encryption, NVMe and PXE boot for Intel cards.

(2). Install VirtualBox

After downloading the **VirtualBox-5.1.12-112440-win.exe** file, double click on the file name to run it, which will install the VirtualBox VM under Windows 10. It also creates an Oracle VM VirtualBox icon on the desktop.

(3). Create a VirtualBox Virtual Machine

Start up the VirtualBox. An initial VM window will appear, as shown in Figure 1.1.

Figure 1.1. VirtualBox VM Window

Choose the New button to create a new VM with 1024 MB memory and a virtual disk of 12GB.

(4). Install Ubuntu 14.04 to VirtualBox VM

Insert the Ubuntu 14.04 install DVD to a DVD drive. Start up the VM, which will boot up from the DVD to install Ubuntu 14.04 to the VM.

(5). Adjust Display Size

For some unknown reason, when Ubuntu first starts up, the screen size will be stuck at the 640x480 resolution. To change the display resolution, open a terminal and enter the command line **xdiagnose**. On the X Diagnostic settings window, enable all the options under **Debug,** which consist of

Extra graphic debug message
Display boot messages
Enable automatic crash bug reporting

Although none of these options seems to be related to the screen resolution, it does change the resolution to 1024x768 for a normal Ubuntu display screen. Figure 1.2 shows the screen of Ubuntu on the VirtualBox VM.

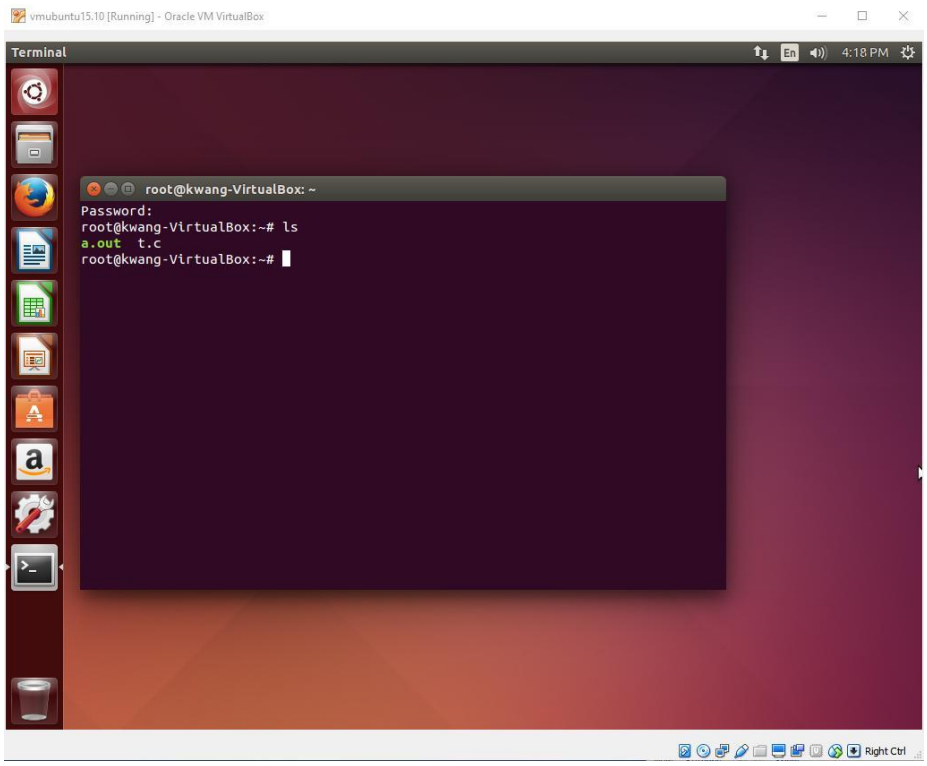

Figure 1.2. Ubuntu Linux on VirtualBox VM

(6). Test C programming under Ubuntu

Ubuntu 14.04 has the gcc package installed. After installing Ubuntu, the user may create C source files, compile them and run the C programs. If the user needs emacs, install it by

```
sudo apt-get install emacs
```

Emacs provides an Integrated Development Environment (IDE) for text-edit, compile C programs and run the resulting binary executables under GDB. We shall cover and demonstrate the emacs IDE in Chapter 2.

1.12.2. VMware

VMware is another popular VM for x86 based PCs. The full versions of VMware, which include VM servers, are not free, but VMware Workstation Players, which are sufficient for simple VM tasks, are free.

(1). Install VMware Player on Windows 10

The reader may get VMware Workstation Player from VMware's download site. After downloading, double click on the file name to run it, which will install VMware and create a VMware Workstation icon on the Desktop. Click on the icon to start up the VMware VM, which displays a VMware VM window, as shown in Figure 1.3.

Figure 1.3. VMware VM Window

(2). Install Ubuntu 15.10 on VMware VM

To install Ubuntu 15.10 on the VMware VM, follow the following steps.

1. Download Ubuntu 15.10 install DVD image; burn it to a DVD disc.
2. Download Vmware Workstation Player 12 exe file for Windows 10.
3. Install Vmware Player.
4. Start up Vmware Player
   Choose: Create a new virtual machine;
   Choose: Installer disc: DVD RW Drive (D:)
   => insert the installer disc until it is ready to install
   Then, enter Next
   Choose: Linux
   Version: ubuntu
   Virtual machine name: change to a suitable name, e.g. ubuntu
   Vmware will create a VM with 20GB disk size, 1GB memory, etc.
   Choose Finish to finish creating the new VM
   Next Screen: Choose: play virtual machine to start the VM.
   The VM will boot from the Ubuntu install DVD to install Ubuntu.
5. Run C Program under Ubuntu Linux

Figure 1.4 shows the startup screen of Ubuntu and running a C program under Ubuntu.

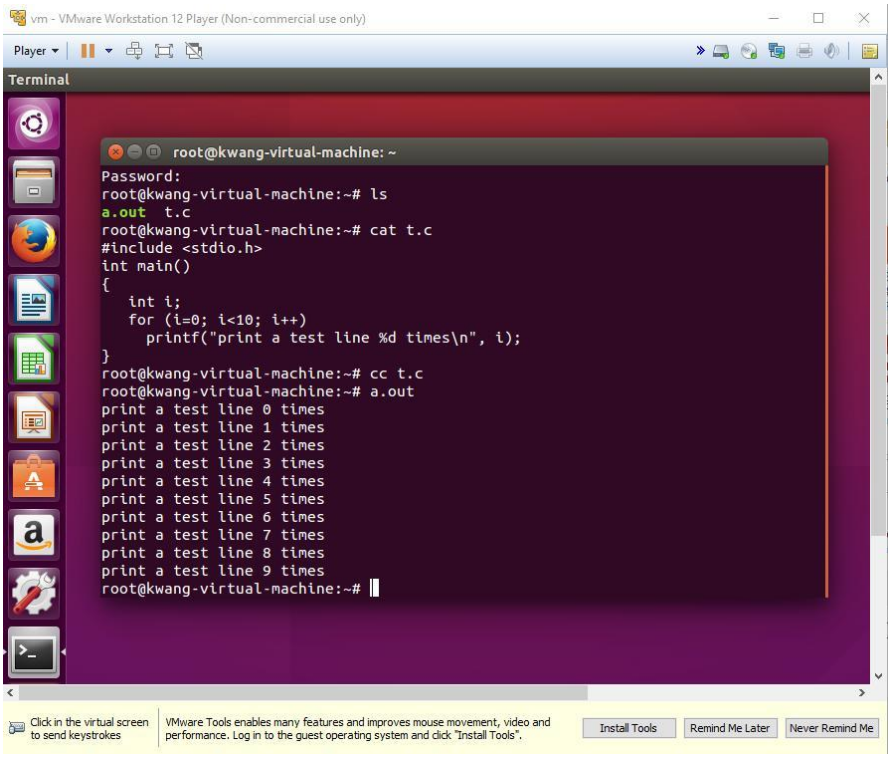

Figure 1.4. Ubuntu on VMware VM

1.12.3. Dual Boot Slackware and Ubuntu Linux

It seems that free VMware Player only supports 32-bit VMs. The install steps are identical to above except the installer disc is the Slackware14.2 install DVD. After installing both Slackware 14.2 and Ubuntu 15.10, set up LILO to boot up either system when the VM starts up. If the reader installs Slackware first followed by installing Ubuntu, Ubuntu will recognize Slackware and configure GRUB for dual boot. The following figure shows the dual-boot menu of LILO.

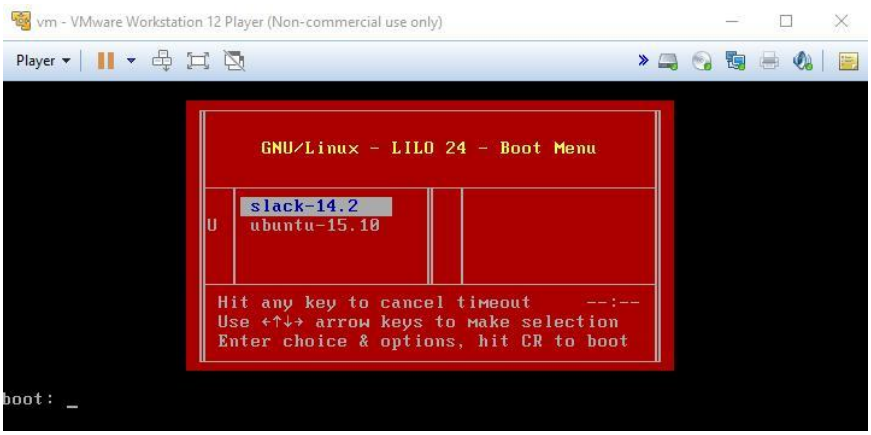

Figure 1.5. Dual-Boot Window of Slackware and Ubuntu Linux

1.13. Use Linux

1.13.1. Linux kernel image

In a typical Linux system, Linux kernel images are in the /boot directory. Bootable Linux kernel images are named as

vmlinuz-generic-VERSION_NUMBER
initrd as the initial ramdisk image for the Linux kernel.

A bootable Linux kernel image is composed of three pieces:

| BOOT | SETUP | linux kernel |

BOOT is a 512-byte booter for booting early versions of Linux from floppy disk images. It is no longer used for Linux booting but it contains some parameters for SETUP to use. SETUP is a piece of 16-bit and 32-bit assembly code, which provides transition from the 16-bit mode to 32-bit protected mode during booting. Linux kernel is the actual kernel image of Linux. It is in compressed form but it has a piece of decompressing code at beginning, which decompresses the Linux kernel image and relocates it to high memory.

1.13.2. Linux Booters

The Linux kernel can be booted up by several different boot-loaders. The most popular Linux boot-loaders are GRUB and LILO. Alternatively, the HD booter of [Wang, 2015] can also be used to boot up Linux.

1.13.3. Linux Booting

During booting, the Linux boot-loader first locates the Linux kernel image (file). Then it loads

BOOT+SETUP to 0x90000 in real mode memory
Linux kernel to 1MB in high memory.

For generic Linux kernel images, it also loads an initial ramdisk image, initrd, to high memory. Then it transfers control to run SETUP code at 0x902000, which starts up the Linux kernel. When the Linux kernel first starts up, it runs on initrd as a temporary root file system. The Linux kernel executes a sh script, which directs the kernel to load the needed modules for the real root device. When the real root device is activated and ready, the kernel abandons the initial ramdisk and mounts the real root device as the root file system, thus completing a two-phase booting of the Linux kernel.

1.13.4. Linux Run-levels

The Linux kernel starts up in the single user mode. It mimics the run-levels of System V Unix to run in multi-user mode. Then it creates and run the INIT process P1, which creates the various

daemon processes and also terminal processes for users to login. Then the INIT process waits for any child process to terminate.

1.13.5. Login Process

Each login process opens three file streams, stdin for input, stdout for output and stderr for error output, on its terminal. Then it waits for users to login. On Linux systems using X-windows as user interface, the X-window server usually acts as an interface for users to login. After a user login, the (pseudo) terminals belong to the user by default.

1.13.6. Command Executions

After login, the user process typically executes the command interpreter sh, which prompts the user for commands to execute. Some special commands, such as cd (change directory), exit, logout, &, are performed by sh directly. Non-special commands are usually executable files. For a non-special command, sh forks a child process and waits for the child to terminate. The child process changes its execution image to the file and executes the new image. When the child process terminates, it wakes up the parent sh, which prompts for another command, etc. In addition to simple commands, sh also supports I/O redirections and compound commands connected by pipes. In addition to built-in commands, the user may develop programs, compile-link them into binary executable files and run the programs as commands.

1.14. Use Ubuntu Linux

1.14.1. Ubuntu Versions

Among the different versions of Linux, we recommend **Ubuntu Linux** 15.10 or later for the following reasons.

(1). It is very easy to install. It can be installed online if the user has connection to the Internet.
(2). It is very easy to install additional software packages by
sudo apt-get install PACKAGE
(3). It is updated and improved regularly with new releases.
(4). It has a large user base, with many problems and solutions posted in discussion forums online.
(5). It provides easy connections to wireless networks and access to the Internet.

1.14.2. Special Features of Ubuntu Linux

Here are some helps on how to use the Ubuntu Linux.

(1). When installing Ubuntu on a desktop or laptop computer, it will ask for a user name and a password to create a user account with a default home directory /home/username. When Ubuntu boots up, it immediately runs in the environment of the user because it already has the default user logged in automatically. Enter **Control-Alter-T** to open a pseudo-terminal. Right click the **Term** icon and choose "**lock to launcher**" to lock the Term icon in the menu bar. Subsequently,

launch new terminals by choosing the **terminal->new terminal** on the menu bar. Each new terminal runs a sh for the user to execute commands.

(2). For security reasons, the user is an **ordinary user**, not the root or **superuser**. In order to run any privileged commands, the user must enter

sudo command

which will verify the user's password first.

(3). The user's **PATH** environment variable setting usually does not include the user's current directory. In order to run programs in the current directory, the user must enter **./a.out** every time. For convenience, the users should change the PATH setting to include the current directory. In the user's home directory, create a **.bashrc** file containing

PATH=$PATH:./

Every time the user opens a pseudo-terminal, sh will execute the .bashrc file first to set PATH to include the current working directory ./

(4). Many users may have installed 64-bit Ubuntu Linux. Some of the programming exercises and assignments in this book are intended for 32-bit machines. In 64-bit Linux, use

gcc -m32 t.c # compile t.c into 32-bit code

to generate 32-bit code. If the 64-bit Linux does not take the **-m32** option, the user must install additional support for gcc to generate 32-bit code.

(5). Ubuntu provides an excellent GUI user interface. Many users are so accustomed to the GUI that they tend to rely on it too much, often wasting time by repeatedly dragging and clicking the pointing device. In systems programming, the user should also learn how to use command lines and sh scripts, which are much more general and powerful than GUI.

(6). Nowadays, most users can connect to computer networks. Ubuntu supports both wired and wireless connections to networks. When Ubuntu runs on a PC/laptop with wireless hardware, it displays a wireless icon on top and it allows wireless connections by a simple user interface. Open the wireless icon. It will show a list of available wireless networks near by. Select a network and open the **Edit Connections** submenu to edit the connection file by entering the required login name and password. Close the Edit submenu. Ubuntu will try to login to the selected wireless network automatically.

1.15. Unix/Linux File System Organization

The Unix/Linux file system is organized as a tree, which is shown (sideway) in Figure 1.6.

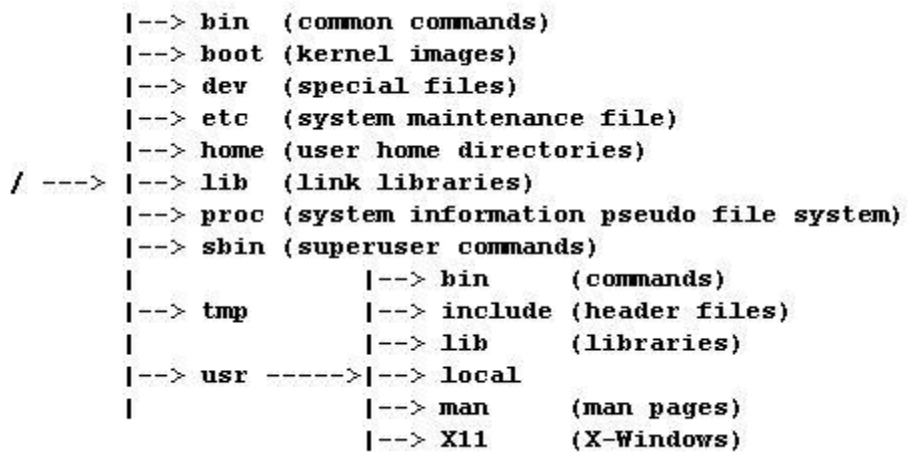

Figure 1.6. Unix/Linux File System Tree

Unix/Linux considers everything that can store or provide information as a file. In a general sense, each node of the file system tree is a FILE. In Unix/Linux, files have the following types.

1.15.1. File Types

(1). Directory files: A directory may contain other directories and (non-directory) files.

(2). Non-directory files: Non-directory files are either REGULAR or SPECIAL files, which can only be leaf-nodes in the file system tree. Non-directory files can be classified further as

(2).1 REGULAR files: Regular files are also called ORDINARY files. They contain either ordinary text or executable binary code.

(2).2 SPECIAL files: Special files are entries in the /dev directory. They represent I/O devices, which are further classified as

> **CHAR special files: I/O by chars**, e.g. /dev/tty0, /dev/pts/1, etc.
> **BLOCK special files: I/O by blocks**, e.g. /dev/had, /dev/sda, etc.
> Other types such as network (socket) special files, named pipes, etc.

(3). Symbolic LINK files: These are Regular files whose contents are pathnames of other files. As such, they act as pointers to other files. As an example, the Linux command

> **ln -s aVeryLongFileName myLink**

creates a symbolic link file, mylink, which points to aVeryLongFileName. Access to myLink will be redirected to the actual file aVeryLongFileName.

1.15.2. File Pathnames

The root node of a Unix/Linux file system tree, symbolized by /, is called the **root directory** or simply the root. Each node of the file system tree is specified by a **pathname** of the form

/a/b/c/d OR a/b/c/d

A pathname is **ABSOLUTE** if it begins with a /. Otherwise, it is **RELATIVE** to the **Current Working Directory (CWD)** of the process**.** When a user login to Unix/Linux, the CWD is set to the user's HOME directory. The CWD can be changed by the cd (change directory) command. The pwd command prints the absolute pathname of the CWD.

1.15.3. Unix/Linux Commands

When using an operating system, the user must learn how to use the system commands. The following lists the most often used commands in Unix/Linux.

ls: ls dirname: list the contents of CWD or a directory
cd dirname: change directory
pwd: print absolute pathname of CWD
touch filename: change filename timestamp (create file if it does not exist)
cat filename: display file contents
cp src dest: copy files
mv src dest: move or rename files
mkdir dirname: create directory
rmdir dirname: remove (empty) directory
rm filename: remove or delete file
ln oldfile newfile: create links between files
find: search for files
grep: search file for lines containing a pattern
ssh: login to remote hosts
gzip filename: compress filename to .gz file
gunzip file.gz: uncompress .gz file
tar –zcvf file.tgz . : create compressed tar file from current directory
tar -zxvf file.tgz . : extract files from .tgz file
man: display online manual pages
zip file.zip filenames : compress files to .zip file
unzip file.zip : uncompress .zip file

1.15.4. Linux Man Pages

Linux maintains online man (manual) pages in the standard /usr/man/ directory. In Ubuntu Linux, it is in /usr/share/man directory. The man pages are organized into several different categories, denoted by man1, man2, etc.

/usr/man/
|-- **man1: commonly used commands: ls, cat, mkdir ...., etc.**
|-- **man2**: system calls
|-- **man3**: library functions: strtok, strcat, basename, dirname
etc.

All the man pages are compressed .gz files. They contain text describing how to use the command with input parameters and options. Man is a program, which reads man page files and displays their contents in a user friendly format. Here are some examples of using man pages.

man ls : show man page of ls in man1
man 2 open : show man page of open in man2
man strtok : show man page of strtok in man 3, etc.
man 3 dirname: show dirname in man3, NOT that of man1

Whenever needed, the reader should consult the man pages for how to use a specific Linux command. Many of the so called Unix/Linux systems programming books are essentially condensed versions of the Unix/Linux man pages.

1.16. Ubuntu Linux System Administration

1.16.1. User Accounts

As in all Linux, user accounts are maintained in the **/etc/passwd** file, which is owned by the superuser but readable by anyone. Each user has a line in the /etc/passwd file of the form

loginName:x:gid:uid:usefInfo:homeDir:initialProgram

where the second field x indicates checking user password. Encrypted user passwords are maintained in the **/etc/shadow** file. Each line of the **shadow** file contains the encrypted user password, followed by optional aging limit information, such as expiration date and time, etc. When a user tries to login with a login name and password, Linux will check both the /etc/passwd and /etc/shadow files to authenticate the user. After a user login successfully, the login process becomes the user process by acquiring the user's gid and uid, changes directory to the user's homeDir and executes the listed initialProgram, which is usually the command interpreter sh.

1.16.2. Add New User

This may be a pathological case for most users who run Ubuntu Linux on their personal PCs or laptops. But let's assume that the reader may want to add a family member to use the same computer but as a different user. As in all Linux, Ubuntu supports an **adduser** command, which can be run as

sudo adduer username

It adds a new user by creating an account and also a default home directory /home/username for the new user. Henceforth, Ubuntu will display a list of user names in its "About The Computer" menu. The new user may login to the system by selecting the new username.

1.16.3. The sudo Command

For security reasons, the **root** or **superuser** account is disabled in Ubuntu, which prevents anyone from login as the root user (well, not quite; there is a way but I won't disclose it). **sudo** ("superuser do") allows a user to execute a command as another user, usually the superuser. It temporarily elevates the user process to the superuser privilege while executing a command. When the command execution finishes, the user process reverts back to its original privilege level. In order to be able to use sudo, the user's name must be in the **/etc/sudoers** file. To allow a user to issue sudo, simply add a line to sudoers files, as in

username ALL(ALL) ALL

However, the /etc/sudoers file has a very rigid format. Any syntax error in the file could breech the system security. Linux recommends editing the file only by the special command **visudo**, which invokes the vi editor but with checking and validation.

Summary

Chapter 1 presents an introduction to the book. It describes the book's scope, intended audience and its suitability as textbook in Computer Science/Engineering curriculums. It presents a brief history of Unix, which includes early versions of Unix at Bell Labs, AT&T System V and other developments of Unix, such as BSD, HP UX, IBM AIX and Sun/Solaris Unix. It describes the development of Linux and various Linux distributions, which include Debian, Ubuntu, Mint, Red Hat and Slackware. It lists both the hardware platforms and virtual machines for Linux. It shows how to install Ubuntu Linux to both VirtualBox and Vmware virtual machines inside the Microsoft Windows. It explains the startup sequence of Linux, from booting the Linux kernel to user login and command execution. It describes the Linux file system organization, file types and commonly used Unix/Linux commands. Lastly, it describes some system administration tasks for users to manage and maintain their Linux systems.

References

1. Bach M., "The Design of the UNIX Operating System", Prentice-Hall, 1986

2. Curry, David A., Unix Systems Programming for SRV4, O'Reilly, 1996.

3. Haviland, Keith, Gray, Dian, Salama, Ben, Unix System Programming, Second Edition, Addison-Wesley, 1998.

4. HP-UX, http://www.operating-system.org/betriebssystem/_english/bs-hpux.htm, 2017

5. IBM AIX, https://www.ibm.com/power/operating-systems/aix, 2017

6. Kernighan, B.W., Ritchie D.M, "The C Programming Language" 2nd Edition, 1988

7. Kerrisk, Michael, The Linux Programming Interface: A Linux and UNIX System Programming Handbook, 1st Edition, No Starch Press Inc, 2010.

8. Leffler, S.J., McKusick, M. K, Karels, M. J. Quarterman, J., "The Design and Implementation of the 4.3BSD UNIX Operating System", Addison Wesley, 1989

9. Leffler, S.J., McKusick, M. K, Karels, M. J. Quarterman, J., "The Design and Implementation of the 4.4BSD UNIX Operating System", Addison Wesley, 1996

10. Linux, https://en.wikipedia.org/wiki/Linux , 2017

11. Love, Robert, Linux System Programming: Talking Directly to the Kernel and C Library, 2nd Edition, O'Reilly, 2013.

12. McKusick, M. K, Karels, Neville-Neil, G.V., "The Design and Implementation of the FreeBSD Operating System", Addison Wesley, 2004

13. Robbins Kay, Robbins, Steve, UNIX Systems Programming: Communication, Concurrency and Threads: Communication, Concurrency and Threads, 2nd Edition, Prentice Hall, 2003.

14. Pthreads: https://computing.llnl.gov/tutorials/pthreads, 2017

15. Rochkind, Marc J., Advanced UNIX Programming, 2nd Edition, Addison-Wesley, 2008.

16. Stallman, R., Linux and the GNU System, 2017

17. Stevens, W. Richard, Rago, Stephan A., Advanced Programming in the UNIX Environment, 3rd Edition, Addison-Wesley, 2013.

18. Solaris, http://www.operating-system.org/betriebssystem/_english/bs-solaris.htm, 2017

19. Sun OS, https://en.wikipedia.org/wiki/SunOS, 2017

20. Thompson, K., Ritchie, D. M., "The UNIX Time-Sharing System", CACM., Vol. 17, No.7, pp. 365-375, 1974.

21. Thompson, K., Ritchie, D.M.; "The UNIX Time-Sharing System", Bell System Tech.

J. Vol. 57, No.6, pp. 1905–1929, 1978

22. Unix System V, https://en.wikipedia.org/wiki/UNIX_System V, 2017