

Introduction to *Linux*

by

*Charles Abzug, Ph.D.
Department of Computer Science
James Madison University
701 Carrier Drive
Harrisonburg, VA 22807*

*Telephone: 540-801-8746
<http://www.cs.jmu.edu/users/abzugcx>
abzugcx@jmu.edu*

Introduction to Linux

OUTLINE

1. OVERVIEW

- a. What is an Operating System?
 - The View of the Computer System
 - The View of the User and of the Programmer
 - Design Goals for an Operating System
 - Operating Environments
- b. What **Linux** Is All About
- c. Why **Linux** Is Important, and Why It Can Be Useful to You
- d. The Future of **Linux**

2. HISTORY OF *LINUX*

- a. **MULTICS**
- b. **UNIX**
- c. Richard Stallman, the Free Software Foundation, and the GNU Project
- d. Andrew Tanenbaum and Minix
- e. Linus Torvalds and the Initial Development of **Linux**
- f. Why Is **Linux** So Much More Popular than its Progenitors?
- g. Origin, Appropriateness, and Proper Pronunciation of the name **Linux**
- h. GNU and “Copyleft”, and their impact on the evolution of **Linux**
- i. Worldwide Cooperation and Support: a Grassroots Movement

3. HARDWARE ARCHITECTURES SUPPORTED

4. DISTRIBUTIONS OF *LINUX* AND VENDOR SUPPORT

5. EXTENDING YOUR KNOWLEDGE ABOUT *LINUX*

6. ACKNOWLEDGMENTS

GLOSSARY

application software: programs usually of commercial origin that effect the end-goals of the user.

architecture: the fundamental design structure of a computer system.

assembly: the process of generating, from a computer program written in assembly language, a list of machine-language instructions. Cf. compilation.

assembly language code: source code that bears a one-to-one correspondence to the hardware instruction set of the computer. Cf. executable code.

batch job: a task requested to be accomplished by the computer, which can be executed without any further input from or interaction with the user.

Command-Line Interface (CLI): a facility for the user to communicate his wishes to the computer by typing specific commands a line at a time. Cf. Graphical User Interface (GUI).

compilation: the generation, from a source code program written in a High-Level Language, first of an assembly-language program that describes how the High-Level Language instructions are to be implemented as a sequence of machine language instructions specific to the hardware of the physical computer, and then the machine language instructions corresponding to the intermediate assembly language expression of the program. (NOTE: Sometimes the machine-language instructions are produced directly by the compiler, bypassing the assembly-language step. Cf. assembly.

device: a physical apparatus that is part of the hardware of a computer. Usually, a device either stores data on a computer in form sufficiently robust that the data are retained even if the electrical power is turned off, or mediates in the transmission of data between the computer and the external world.

disassembly: a process by which a set of executable machine-language instructions is analyzed and rewritten as a list of assembly language source code instructions. Cf. assembly.

executable code: a list of instructions both written in machine language and also containing all memory locations explicitly specified, and therefore directly executable by the computer. Cf. source code and machine code.

Graphical User Interface (GUI): a facility for the user to communicate his wishes to the computer principally by pointing with an appropriate device and in a designated manner either to pictorial

elements displayed on an appropriate two-dimensional surface or to lists (menus) of various options appearing on the display medium. Cf. command line interface (CLI).

kernel: the portion of the operating system that must reside continuously in the primary memory of the computer from the time of completion of boot-up until system shutdown.

machine code: a program expressed as a list of machine instructions directly executable by the computer, except that some of the references to memory locations might not yet be completely specified. Cf. source code and executable code.

multi-tasking: an environment in which several tasks can be executed concurrently on behalf of the computer user, exploiting time periods when one task must remain idle while waiting until a critical event occurs until that task can be resumed, and performing other tasks in the meantime.

multi-user: an environment in which a computer system can serve the needs of several users concurrently by interleaving the execution of tasks for different users.

open-source: a special program development environment in which, in contrast to the commercial software environment, the source code for the software is freely distributed, thus enabling the user community freely to modify the software either to fix bugs in the code or to enhance or alter the functionality of the software to meet the individual needs of different systems.

portability: the ability of software originally written with the intention of being executed in a particular hardware-combined-with-operating-system environment, known as a platform, to be transformed and transported for execution on a different hardware platform.

real-time: a relatively unusual environment in which a computer system operates under defined constraints for responding to specific external events. For example, a collision warning system in an airplane must generate a warning and deliver it to the pilot sufficiently speedily as to enable him to take appropriate action to avoid the impending collision. In a real-time system, the issuance of an error message instructing the user to close down some applications in order to free up excessively encumbered memory space is usually precluded.

shell: a particular Command-Line Interface having explicitly defined properties. Shells available in the *Linux* environment include the *Bourne Again Shell (bash)*, the *TC Shell (tcsh)*, the *Z shell (zsh)*, and the *Korn shell (ksh)*, and generally contain facilities for program-like operations.

shell script: a file containing a sequence of commands addressed to the operating system that facilitates the repeated execution of the included commands without their having to be laboriously retyped each time they are executed.

source code: a complete list of instructions that describes how a computer program is to be executed, written in a precisely defined programming language that is relatively easy for a human reader to

understand, yet is also readily converted by appropriate special software into a set of machine-language or executable instructions.

utilities: software provided together with the operating system which provides commonly-used functions that can greatly enhance the ability of the computer system to provide service to the user.

ABSTRACT

The *Linux* operating system occupies a special position in the world of Computer Science. Unlike the great majority of operating systems, which are produced by commercial developers and sold at a profit, *Linux* is produced and maintained by a coterie of enthusiastic volunteers, and is distributed with no license fees whatsoever. It is available in several versions that run with nearly identical look and feel on a diverse group of hardware platforms. *Linux* is famed both for its stability and for its efficiency, often running for months, or occasionally years at a time without having to be rebooted, while also achieving excellent performance. It conveys many of the properties of *UNIX* that have made that operating system extremely popular among Computer Science professionals. *Linux* source code is as freely available as the executable code, thus giving users complete freedom to modify and adapt the operating system to the special needs of their systems. *Linux* maintains the tradition of openness and voluntarism that originally characterized the *UNIX* world, while at the same time avoiding the concomitant fragmentation experienced by *UNIX* into a variety of dialects. *Linux* is likely to continue to increase in importance.

Introduction to Linux

1. Overview

Linux is probably the best all-around operating system in use in the world today. **Linux** has a number of competitors, most of which are commercial products, like Microsoft **Windows** in its various manifestations, IBM's **OS/2**, Sun Microsystem's **Solaris**, SGI's **Irix**, and Novell's **Netware**. In contrast to these other operating systems, the development of **Linux** is carried out not by paid employees operating in an industrial environment, but rather by a small army of dedicated and enthusiastic volunteers, many working evenings and weekends during their spare time. Both the mode and the culture within which **Linux** was nurtured are responsible in large measure for the superiority both in usability and in performance attained by this operating system. In order to explain this attainment by **Linux** over its rivals, it is necessary first to describe what is an operating system and what is the role of the operating system in the functioning of a computer. In addition, it is necessary to convey the uniqueness of **Linux** in terms of both how it was originally developed, as well as how it continues to evolve.

a. What is an Operating System?

The operating system is the most important piece of software in a computer. There are two principal points of view for looking at an operating system, that of the computer system and that of the user and programmer.

(1) The View of the Computer System: From the point of view of the computer system, the operating system is the software responsible for mediating the control of all three families of the computer's resources: hardware, software, and data. The hardware resources controlled by the operating system include one or more Central Processing Units or CPUs, the main memory (often referred to as Random Access Memory or RAM), the keyboard and display, and also any devices that may be present. These devices include communications facilities, such as a network card or modem, as well as all storage devices, including magnetic disks ("hard" disks, floppy disks, and various proprietary removable disk cartridges, such as **Zip**, **Jaz**, Bernoulli, etc), CDs or DVDs, and tape drives. Software resources of the computer include, in addition to the operating system itself, a variety of utilities that may be bundled with the operating system by the vendor, as well as any separate commercially purchased or custom-programmed software applications. Applications software, at a minimum, on a computer intended for use as a personal workstation usually includes a word processor, E-mail handler, spreadsheet, web browser, and database manager. Applications software on a server may include a robust database manager specifically designed for simultaneous access by multiple users, or some other heavy-duty application. In a corporate environment, computers are typically purchased *en masse*, and both the hardware configuration and the installed software are usually

identical or nearly identical for a group of desktop systems purchased together as a single purchasing action by a commercial or governmental organization. The data resident on each system, however, are what distinguish that system from all others that may be otherwise identical. The data make each computer different from every other system in the organization. Thus, in summary, the operating system is the highly complex software that controls all three types of resources of the computer: hardware, software, and data.

The operating system is much more, however, than merely the manager of the resources of the computer system. In addition to mediating the control of the disparate resources of the system, the operating system also provides the interface with which the user interacts with the computer, both to pass on the expressed wishes of the user to the computer and to communicate to the user information about what the system is doing, as well as reports regarding the status of requests made by the user, and warnings and messages regarding errors and malfunctions that the user needs to know about.

(2) The View of the User and of the Programmer: So far, we have considered the operating system principally from the point of view of its role in controlling the computer. The other point of view is that of the user or of the programmer. There are several services that are provided both to the user and to the programmer by the operating system. These include:

1. execution of programs at the user's request;
2. performance on behalf of the program of input/output operations, thus saving the programmer from having to program such operations in detail;
3. interaction with devices, thereby relieving the programmer of having to program in detail in accordance with the needs of the individual device;
4. reading and writing of files of various types;
5. handling of communication from the user to the machine, typically via either a Command-Line Interface (CLI) or a Graphical user Interface (GUI);
6. communication between processes;
7. detection and reporting to the user of errors occurring in system operation;
8. communication to the user of status reports, including report of the successful completion of tasks requested by the user;
9. a set of rules for writing function or subroutine calls that can be invoked from an application program and that are carried out on its behalf by the Operating System (Application Programming Interface, or API);
10. provision of an environment conducive to the development by programmers of application programs to carry out useful work on behalf of the user, as well as to the testing, debugging, and maintenance of application programs.

Design Goals for an Operating System: There are four principal goals that the developer has to meet in the design of the operating system. First among these is efficiency of use of the resources of the system. The resources of the computer system are expensive, and the user therefore is eager to see the resources put to use efficiently, as well as to get his work accomplished in a timely manner. Second in importance is ease of use of the system. Users come in a great range of levels of sophistication of computer skills. The entire spectrum of

users, from the almost totally computer-illiterate to the most highly skilled computer professional, should all be able to exercise an adequate level of control over the operation of the system so as to obtain from it a high level of service. A third goal is that the resources of the computer system be adequately apportioned to the various tasks to be accomplished so that each task progresses at a speed and efficiency appropriate to its importance relative to other tasks. In particular, if two or more users share the machine, then each user must be allocated an appropriate share of each of the various resources of the system. Finally, the security needs of the system must be adequately addressed. This last concern is particularly important for a system whose use is shared by multiple users.

Operating Environments: There are several different environments in which computer systems function. In extreme circumstances, a particular environment may require the services of a special-purpose operating system whose design is optimized to serve the special environmental needs of that system particularly well. In most cases, however, a general-purpose operating system can be developed which meets the needs of a broad range of uses and that includes sufficient flexibility to enable the system manager to configure the system on installation to meet the operational needs anticipated for that particular system. A good general-purpose operating system will also provide a range of facilities and services which can be adapted and tuned as necessary so as to enable the system to continue to meet the needs of its operational environment even as these needs change over time.

Broadly speaking, the operational environment of a computer system falls into one of three categories. The simplest environment is that of the single-user-at-a-time. Most desktop computer systems, including those of higher-performance capability also known as workstations, fall into this category. The old *DOS* that came with the original IBM PC (*IBMDOS*) as well as with most PC clones produced by other manufacturers (*MSDOS*) was a single-user operating system. *DOS* was not only a single-user, but also a single-tasking operating system. That is, the user could have the computer do only one thing at a time. The user had to wait until the current task was completed before assigning a new task for the system to accomplish. Single-user operating systems with more sophistication were developed that allowed the user to initiate several tasks that ran concurrently, thus allowing the more efficient use of computer resources. When a disk read or a disk write operation is required, for example, it is usually necessary for the read/write head of the disk drive to travel to some location overlying one particular circular track selected from the many that exist on each of the disk platters. This operation, known as a disk head seek, is incredibly slow by computer standards, occupying an amount of time in which the CPU, were it free and unencumbered, could perform thousands or even millions of operations. Thus, while the disk drive is executing its head seek, the CPU could be either doing something else for the same task that is not dependent upon completion of the disk read or write operation, or else it could work on a completely different and unrelated task. In an old-fashioned single-tasking system, this does not happen, however. Instead, the CPU waits for the disk read or the disk write operation to complete, and only then does it resume operation. In the large-scale computer (minicomputer or mainframe), multi-tasking operating systems were developed as early as the 1960s. At the smaller scale of the personal computer, several operating systems were also developed in the 1980s that enabled the single-user microcomputer also to do multi-

tasking. These included the operating system for the MacIntosh, known as *MacOS*[®], and several variants of Microsoft *Windows*[®], as well as IBM's *OS/2*[®]. Several *UNIX* variants were developed for the PC as well, which were even more capable.

Next in complexity, after the single-user multi-tasking environment, is the environment characterized by multiple simultaneous users. This environment is characteristic of most mini-computers and mainframes, as well as of servers. Operating systems that have facilities that enable their use in a server or multi-user environment include the server versions of *WindowsNT*[®], *Windows2000*[®], and *WindowsXP*[®], also Novell *Netware*[®] the server version of IBM's *OS/2*[®], DEC's *VMS*[®], various mainframe operating systems, and a host of proprietary variants of the *UNIX*[®] operating system, including IBM's *AIX*[®], Hewlett-Packard's *HP-UX*[®], Silicon Graphics' *IRIX*[®], and HP-Compaq/DEC's *Ultrix*[®].

Historically speaking, one of the earliest capabilities to be developed in operating systems was the ability to handle batch jobs. This development took place prior to the development of multi-user, multi-tasking operating systems. In a batch environment, one or more tasks are submitted to be run on the system. Each of these runs to completion, and the results are collated and left for later, off-line examination by the user. Modern systems with multi-user, multi-tasking capability may also be provided with the capability to run batch jobs. Usually, the tasks submitted as batch jobs have no need for ongoing monitoring or supervision by the user. Examples of such jobs are regular, complete system backups, and periodic scanning of the entire system for computer viruses. Such a job can either be scheduled to run at a time when the user is not present, so as not to compete for system resources with tasks of greater urgency, and thus interfere with the performance of those tasks for which the user is anxiously awaiting results. Alternatively, the batch job can be run in background at a reduced priority level, so that it runs only when a task to which the user is attending will not be slowed or delayed. Operating systems that are in the more robust category often have the capability to run batch jobs, in addition to handling multiple tasks and multiple users.

Finally, there is the real-time environment, where the computer is used to control something whose on-time performance is critical. Examples of real-time environments include the flight-control and navigational computers of aircraft and spacecraft, various military environments, such as the command and control computers used in single-vessel as well as integrated fleet defensive systems, like the U.S. Aegis system, aircraft and anti-missile weapons-systems control computers, like the U.S. Patriot anti-aircraft and anti-ballistic-missile system, systems used in medical diagnosis and treatment, systems used in factory or shop floor automation, including robotics, large-scale distributed computer systems used to control transnational telecommunications systems, and the computers used to control nuclear power plants. In several of these cases, a general-purpose operating system can be used that provides, besides the multi-user and multi-tasking features found in many modern operating systems, additional facilities suitable to the more exacting needs of a real-time environment. However, in those instances where the real-time needs are particularly severe, only a special-purpose operating system will do. Under such extremely constrained circumstances, it is usually cost-

effective to have a dedicated, special-purpose computer that does nothing else but see to the needs of the so-called hard real-time environment.

b. What Linux Is All About

Linux is a *UNIX*-like group of operating systems, which was developed as a cooperative effort by a loosely knit team of capable and enthusiastic programmers who volunteered their services. The programmers on the original development team were associated with the Free Software Foundation's *GNU* project. The activities of programmers working on the *GNU* project were augmented by Linus Torvalds and a small number of associates working with him. This latter group contributed the operating system's kernel. The name *Linux*, which properly applies to the kernel alone, is derived from Linus Torvalds' first name. The *Linux* kernel was originally targeted at the Intel x86 family of processors. However, it now comes in several varieties, each of which runs on a particular hardware platform. The various incarnations of the kernel intended for the different hardware platforms are all integrated with the pre-existing *GNU* software to form a complete operating system. Quite a number of hardware platforms are currently supported. The various members of the *Linux* family are all multi-user, multiple-concurrent-process operating systems featuring time-sharing, but also supporting batch processing in background. They differ from each other only in whatever way is necessary to support the particular hardware architectures on which they run. The look-and-feel of all these systems is very similar.

Linux is open-source software. The source code is bundled together with the executable code and the documentation, and all can be had without charge. Should the user require the code and manuals on CDs, then these are available for purchase at a very modest charge to cover the media, duplication, packaging and distribution costs only. The software is universally usable by anyone for any purpose with absolutely no licensing fees.

The *Linux* user is free to modify any or all of the source code, and then to recompile the modified code, to meet the needs of his system. If the modification is potentially of use to other members of the *Linux* community, then he is encouraged to share with the community the changes made. Furthermore, anyone at all who enhances the software is welcome to distribute the enhanced version, provided that the provisions of the *GNU* General Public License are upheld, preserved, and transmitted onwards together with the new code and documentation, thus preserving the users' freedom to read, study, modify, and enhance the software.

c. Why Linux Is Important, and Why It Can Be Useful to You

The importance of *Linux* is due to a number of factors. Some of these are technical and some are non-technical, particularly related to the *Linux* culture.

Introduction to *Linux*, by Charles Abzug

- (1) Performance and efficiency are superb. In particular, users who find themselves often frustrated by the notorious hourglass encountered in the various *Windows* environments find themselves warming up to *Linux* as they see a snappier and more responsive system.
- (2) Reliability is certainly among the best in the industry. *Linux* systems are extraordinarily stable, with several server systems known to have been running continuously for several years without ever having to reboot. There is an organization called “The *Linux* Counter” (URL: counter.li.org), which encourages *Linux* users to register as well as to provide information voluntarily regarding their installation of *Linux*, and especially its performance. The data displayed recently show that there is one machine that has been running *Linux* continuously for 1000.3 days (i.e., nearly three years). This machine happens to be not an Intel architecture machine, but a Hewlett-Packard/Compaq/DEC Alpha. However, there is a list of the top ten machines reported to be up continuously on *Linux*, and the other nine, all of which consist of Intel hardware (one 80486 and the rest in the Pentium family) have all been running for more than 500 days. In fact, the average running time reported for *Linux* is an astounding 37.7 days. The contrast between the ultra-high reliability of *Linux* and the frequently appearing “blue screen of death” of the *Windows* world is particularly striking.
- (3) *Linux* complies with the Portable Operating System Interface Standard (POSIX). More than 95% of the code is written in the programming language C, and thus it can be ported relatively readily to any new hardware architectures that might be developed.
- (4) *Linux* already runs on a huge variety of hardware, ranging from desktop systems to mainframes (details presented below). Over the entire range of hardware supported, there is a consistency both of user interface and of programmer interface. Thus, the user does not need to learn a new operating system when switching to a new hardware platform.
- (5) There is a standardized structure to the *Linux* file system, which results in a uniform location of critical files over both different hardware platforms and different *Linux* distributions. The *Linux* community recognized relatively early on that the originally different file structure used by the various vendors in their *Linux* distributions was chaotic, and so a FileSystem Standard (FSSTND) was developed to bring the situation under control. Subsequently, a superior Filesystem Hierarchy Standard (FHS) replaced FSSTND.
- (6) Development and evolution is vastly different for *Linux* from what it is for proprietary operating systems. Since they are volunteers, *Linux* developers do not have to answer to either a marketing department or a profit-oriented corporate bureaucracy. A proposed change or a reported software error is posted by the interested party on the Internet, where it is likely to be seen very quickly by any number of *Linux* enthusiasts. If one of them deems it to be worthwhile, then either the change is implemented or the bug gets fixed, whichever is applicable, and detailed information on the change is then also posted.

Introduction to *Linux*, by Charles Abzug

There is a spirit and a culture of cooperation, and therefore anyone who programs an improvement in the code, even though primarily intended for his/her own private use, will usually also post it on the internet so that other users may also benefit from it if they wish. All of the improvements are then collated for integration into the next release of the kernel software. This is in marked contrast to the philosophy encountered in the commercial software-development world, where the attitude taken towards the users is typically, as described by Richard Stallman (2001, page 1), “If you want any changes, beg us to make them.” In addition, in the commercial environment users must often pay for upgrades with no assurance that the bugs that they have reported have already been fixed rather than still sitting in the queue waiting to be worked on.

- (7) New modules can be linked in to the kernel, and old modules de-linked, while the system is running. In most cases, it is not necessary to bring down the system for reboot.
- (8) The operating system is highly flexible in use. The user can exercise a high degree of control from a single window over several jobs, including both foreground and background processes.
- (9) *Linux* incorporates features of both major streams of the *UNIX* world: AT&T *UNIX* and Berkeley Standard Distribution. For example, there are several shells (command-line interfaces) available under *Linux*. The first popular *UNIX* shell was the Bourne Shell. The “Bourne Again Shell” (*bash*) is an improved version of the original *UNIX* Bourne Shell from AT&T *UNIX*. The developers of Berkeley *UNIX* produced the “C Shell”, which differs in several key aspects from the earlier Bourne shell. *Linux* offers the TC Shell (*tcsh*), which is an improvement on the C Shell. The “Z Shell” (*zsh*) is a hybrid, and contains several features of both *bash* and *tcsh*, and of the Korn Shell as well. Several additional shells are also available. Some of the more valuable facilities that were once unique to one or another of the primordial shells have now been incorporated into one or more additional shells, thus to some extent reducing the differences among them in facilities offered. For example, the C Shell of Berkeley *UNIX* introduced a *history* facility, which had not been present in the original Bourne Shell. *History* retains some number of most recently issued commands. The retained commands can then be re-executed, either in the identical form to which it had originally been issued, or in altered form. *Bash* incorporates a history facility, thus in some small measure reducing the number of features that distinguish *bash* from *tcsh*. While the facilities that originally distinguished certain shells have been incorporated under *Linux* into other shells, as well, thus diminishing the uniqueness of the feature set offered by each separate shell, nevertheless the original differences in command syntax between the various shells have been retained.

Several additional *UNIX* features are also supported by *Linux*. Redirection of both input and output is a simple means of executing a program and providing the input from a file instead of from the console or standard input device (keyboard), and of having the program’s output written to a file instead of to the standard display device (usually either

a video display tube or a flat-panel liquid-crystal display). Many other operating systems also allow input and output to be redirected to files, but redirection in other operating systems can be a cumbersome affair (e.g., in *VMS*). Filters are also prevalent in *Linux*, as they are in *UNIX*. These are programs that are designed specifically to process a stream of input data and to produce a stream of output data. Filters often intervene to process output data from one program before the data are input into another program. *Linux* also supports both hard links and soft links to facilitate the sharing of files. A hard link points to the file's physical structure on the disk, whereas a soft link connects references the file through the directory structure. A large number of *UNIX* utilities are also supported. These include about 15 extremely helpful utilities that support either simple tasks, including `date` (writes the current date and time to the standard output device), `echo` (specifies a text string to be written to the standard output device), and `lpr` (sends a file to te printer), or routine file manipulation and disk management tasks, including `cat` (concatenates files, and displays the concatenation on the standard output device), `cd` (changes the present working directory to a specified new location), `chgrp` (changes the group ownership associated with a specified file), `chmod` (changes the permissible access mode for a file), `chown` (changes the ownership of a file), `cp` (makes a duplicate copy of one or more files, either in the same directory or in a different, specified directory), `df` (prints on the standard output device either the number of free disk blocks or the size in kilobytes of the free disk space, and the number of free files), `dd` (copies files between devices, converting between different block sizes when so specified), `ln` (creates either a hard link or a soft link to a file), `ls` (lists the contents of a directory), `mkdir` (creates a new directory), `rmdir` (deletes specified directories), `mv` (moves and renames files and directories), and `umask` (sets the default access permission bits for all subsequently newly-created files and directories). There are also several well-known complex or special-purpose utilities. These include `awk` and `grep`, which are used for searching for patterns in a file, `sed` (a batch editor), `finger`, which provides specific information regarding authorized users of the system, `make`, which is used principally to update a set of executable programs after modifications are made to the source code, and `pine`, which is used both to receive and send news and e-mail.

- (10) *Shell script* is the term used in the *UNIX* culture to denote what is usually termed a *command procedure* in other operating systems. A command procedure or shell script is a list of operating-system commands that may contain program-like features. If there is a distinct ordered list of operating system commands that the user needs to execute repeatedly, for example, immediately after every login or immediately before every logout, then most operating systems have a facility for recording the list of commands in a file, which can then either be executed automatically upon login or logout, or can be invoked by the user through the issuance of a single command that results in the execution of the entire contents of the batch file, which can contain as few as one operating system command or as many as thousands. In most cases, both interactively typed operating system commands and pre-recorded batch command procedures (like the *.bat files of *MS-DOS/PC-DOS* and the *.cmd files of *Windows*) are executed through

interpretation. That is, each command in turn is examined, parsed, and translated in turn into a series of executable machine instructions. Several of the *Linux* shells, however, provide a facility for the user to write command procedures called *shell functions*, which are retained in memory in partially parsed (preprocessed) form. This allows the command procedure to be executed more speedily when called, as it is not necessary to do all of the work of parsing every time the command procedure is executed. Note that the shell function can be stored in memory either in *bash* or in *zsh*. In addition, *zsh* has a facility called **autoload** that allows the user to specify a shell function which is not immediately loaded into memory but is, instead, stored on disk, thereby economizing in the use of memory. Only when the shell function is actually invoked is it copied from disk to memory, thus economizing both in the setup time and in the encumbrance of memory, which can be significant if there are many shell functions that are seldom used.

- (11) Additional features from the *UNIX* world that are supported in *Linux* include two very powerful editors, *vi* and *emacs*, as well as a wide variety of utilities.
- (12) Support is provided for a huge variety of peripheral devices and adapter cards.
- (13) Ample tools and facilities are provided for software development, including, in addition to **make**, the *Concurrent Versions System (CVS)*, and the *Revision Control System (RVS)*. These utilities are very useful not only for developing new software, but also for taking existing software and both configuring and installing it to run, either on a different hardware architecture, or even on a different operating system.
- (14) Several emulators are available to enable the running of programs designed for other operating systems. The operating systems supported include MacOS for the Macintosh (*executor*), the old DOS operating system that ran on the early Intel hardware (*dosemu*), the *Windows* environment (*wine* and *wabi*), several flavors of *UNIX*, including AT&T's System V Release 4, the University of California at Berkeley's BSD, and Santa Cruz Operation (SCO) *UNIX*. In addition, *GNOME* and *KDE* provide *Windows*-like environments for running modern *Windows* applications under *Linux*, and *Plex86*, distributed by Debian, allows other operating systems designed for the Intel 80x86 hardware environment to run under *Linux* on Intel 80x86 hardware. There is also a commercial product called *VMWARE* that comes in versions that allow other operating systems to run under *Linux*. Note that several of these operating-system emulators require licenses as well as executable code for the operating system environment that is being emulated.

Note that *Linux* is currently reported as being installed and used in a total of 183 distinct countries, pseudo-countries, and other geographical areas, such as the Faeroes Islands (counter.li.org). The total number of users is currently estimated as being somewhere between 3 million and 24 million, with a "best guess" of 18 million (counter.li.org/estimates.php).

d. *The Future of Linux*

At present, there are two principal factors limiting the expansion of *Linux*. First is the limited number of mainstream software applications that run under *Linux*. In the words of Linus Torvalds (foreword to Sobell, 1997), “*Linux* has the same problem all other operating systems have had: a lack of applications. . . . In order to be a real driving force, *Linux* needs to have more applications, and those applications need to be readily available with wide distribution and low price.” However, he is optimistic regarding the future, as he also states, “So far *Linux* ports of various applications have followed DOS/Windows pricing (less expensive) rather than the *UNIX* pricing (more expensive), so I’m reasonably hopeful we can have the types of applications that will help *Linux* grow.” In fact, several vendors have released versions of their mainstream applications ported to *Linux*. The number of such applications is still modest, but is growing.

The other major factor limiting the expansion of *Linux* is the natural hesitancy of corporate IT managers to rely upon major software that is not a product of a monolithic “reputable” vendor. The corporate world feels more comfortable when dealing with entities that can be sued if something goes wrong with the software. The *Linux* culture doesn’t fit into the corporate mold. Some IT managers, nevertheless, have listened to their technical experts and have taken the leap to *Linux*. There is a modest, but growing foothold occupied by *Linux* in the commercial world. Augmenting this process are two packages that allow *Windows* applications to run under *Linux*: KDE and GNOME. Unfortunately, not all *Windows* applications run successfully in one of these environments. If either of them should develop to the point where nearly everything runs, or if a new windowing environment for *Linux* were to appear incorporating an improvement in functionality providing universal compatibility with all *Windows* applications, then that could have a major impact in expanding the acceptance of *Linux*.

Where will *Linux* be in another year; or in two, five, ten, or 20 years? Prediction is a notoriously unreliable endeavor in the computer industry, probably even more so than in general society. The further out we look, the harder it is to predict with any level of accuracy. For example, Bill Gates is supposed to have declared in 1981 that 640 kiloBytes is as much memory as anyone would ever need. In addition, it is said that Ken Olson opined in the sixties or seventies that there would never be computers in individual households. These are both very intelligent people who each possessed an excellent grasp of the computer industry at the time they made the pronouncements for which they are famous. The reason that their prognostications were so far off the mark is that developments took place in the industry that were simply unpredictable at the time they spoke. As a rule, the shorter the outlook the more straightforward it becomes to make a credible prediction. Looking at *Linux*, certainly in the short term, that is, for the next one or two years, we can expect to see continued growth at a rate comparable to what has occurred in the past few years. *Linux* enthusiasts expect and hope that this growth will continue beyond the immediate future into the medium-term and far-term future as well. While there are some who expect *Linux* eventually to unseat *Windows* from its current position of industry dominance, a more balanced view would not consider that to be a high-likelihood event. The basis for the expectation of accelerated growth is the expected

continuation of at-best mediocre performance, insufficient reliability, and erratic operation of *Windows*, coupled with the traditionally sluggish responsiveness by the *Windows* software vendor in the fixing of bugs. This latter, in particular, is generally thought to be due perhaps in part to the attitude of the company (let the user find the bugs for us, and let him come to us on his hands and knees begging us to fix them), but is probably due in greater measure to the limited ability of the software producer to manage a project of enormous complexity consisting of millions of lines of code, while also trying to keep up with changes in the hardware environment and increasing demands by users for added capabilities in the software. However, it must be borne in mind that the deeper the inroads that *Linux* makes in *Windows*' share of the market, the greater the incentive this gives to Microsoft to become competitive! No one can predict where that will lead. Will Microsoft change its long-standing policy and release the source code to allow critical examination, as well as custom modification, by the programming community? Will they solicit suggestions for code changes and for bug fixes, perhaps offering to pay handsomely for sound code contributed by users? Will Microsoft surprise the world by itself adopting the *Linux* kernel, thoroughly rebuilding the rest of its operating software to maintain the look and feel of *Windows*, while achieving the benefits of the Open-Source movement for continued evolution of the kernel? Alternatively, will some other commercial firm perhaps come up with a new operating system that will be technically competitive with *Linux*, while also having the advantages of roots in the commercial environment and complete compatibility with *Windows* applications that would make it competitive with *Windows*? I certainly do not mean to suggest that a commercial challenger is likely to arise to *Windows*' dominance of the marketplace, but surprises and revolutions in the technological arena certainly do occur, and such a possibility cannot be glibly ruled out. The high-tech marketplace is littered with the corpses of formerly giant firms that had dominated major segments of the industry, and that subsequently either abandoned the computer marketplace and limited themselves to other areas of business, or went out of existence entirely, or have been swallowed up by later or once-smaller upstarts. Names like *General Electric*, *Honeywell*, *Control Data Corporation*, and *Digital Equipment Corporation* come readily to mind. No one can foretell the future of *Microsoft*.

2. History of *Linux*

Linux is not an isolated phenomenon. It arose as the culmination of what can be considered in Computer Science terms to be a rather lengthy history of development. Understanding what were the antecedents of *Linux* and how it came into being provides substantial insight into what *Linux* is all about.

a. MULTICS

The *Linux* story begins with the development in 1965 of an operating system called *MULTICS* (*MULT*iplexed *I*nformation and *C*omputing *S*ervice). *MULTICS* was a joint effort undertaken originally by MIT and General Electric (GE). Shortly after the work was begun, General Electric decided to get out of the computer business, and sold off its computer operation to Honeywell, which thus became the successor to GE. Subsequently to initiation of the project, AT&T's Bell Laboratories joined the development team, thus increasing the team's membership from two to three. However, in 1969 Bell Labs withdrew from the *MULTICS* project.

b. UNIX

Two members of the Bell Labs team that had been working on the *MULTICS* project were Dennis Ritchie and Ken Thompson. There was a particular computer game that they had enjoyed playing that had run on the *MULTICS* system, and they were frustrated that due to their employer's withdrawal from that project they were no longer able to play their favorite game. There was an unused Digital Equipment Corporation PDP-7 computer available in their laboratory, so Thompson developed a simple operating system to enable them to port the game to the PDP-7. He originally wrote the operating system in PDP-7 assembler in 1969, and gave it the name *UNICS* as a play on the name of the *MULTICS* system they had previously been working on. Subsequently, the spelling was changed to *UNIX*.

As it was originally developed, *UNIX* was not portable. It was closely linked to the PDP-7 hardware for which, and in whose assembly language, it was originally developed. A need for portability was soon recognized, however, and in order to facilitate the achievement of portability, Thompson developed a new programming language. The new language was derived from a pre-existing language called BCPL, and it was given the simple name "B". The new language provided, unlike most previous languages available at that time, such as COBOL and FORTRAN, an unusual level of access to the hardware of the machine. Subsequently, Ritchie developed another language derived from "B", which he called "C". Then Ritchie and Thompson together rewrote *UNIX* in "C" in 1973. They did this to make it possible easily to port most of the operating system to other hardware, as most of the software could be compiled and executed on any computer for which a "C" compiler was available. Only a relatively small part of the code of the kernel, that had to be written in assembly language, would have to be rewritten in order to allow execution on the new hardware.

AT&T recognized that *UNIX* had some fine qualities, although initially they appeared to be oblivious to its commercial potential. Therefore, it was initially distributed free of charge. Consequently, there developed early on a *UNIX* culture in which contributions of code for use in *UNIX* were made from all over the world. AT&T belatedly realized that *UNIX* had commercial potential. They therefore claimed it as their intellectual property, and thereafter they undertook a very far-sighted strategy to continue popularizing the operating system. They licensed it to academic institutions at very low rates. As a result, several generations of Computer Science students were exposed to *UNIX* at a very early stage in their careers.

According to the Bible, at one point in its development the human race enjoyed a common language: “Now the whole world had one language and a common speech.” — Genesis 11:1. The *UNIX* world at one time enjoyed the same homogeneity. Just as in general human civilization, so, too, for *UNIX* this state of affairs did not last very long. The University of California at Berkeley started its own separate dialect of *UNIX*, which featured the C shell, a Command-Line Interface that was decidedly different from the Bourne shell, which had become popular early on in the *UNIX* world. There were other differences that also distinguished the Berkeley Standard Distribution of *UNIX* (BSD). AT&T tried to control and standardize *UNIX* around their evolutionary dialect, which eventually became “System V Release 4 (SVR4)”. Furthermore, individual hardware vendors started getting into the *UNIX* business, and one of the first things a hardware vendor tries to do is to differentiate its product from that of its competitors to facilitate its goal of selling more of its hardware. Thus, not only did the two principal streams of *UNIX* diverge from each other, but also each one developed into multiple dialects. One of the major advantages of having a single standardized operating system in use on different hardware platforms is that the user does not have to undertake a major personal training program every time he migrates to a new hardware platform. If the *UNIX* dialect in use on each platform is different, however, then an appreciable portion of the advantage of having a common operating system is lost.

Another problem afflicting the *UNIX* world was the set of licensing restrictions. Commercial *UNIX* was supplied without source code, as is the usual practice with fully proprietary operating systems, such as Hewlett-Packard/Compaq/Digital Equipment Corporation’s *VMS*, the various members of Microsoft’s *Windows* family, IBM’s *OS/2*, Apple’s *MacOS*, and various other proprietary operating systems used on PCs, workstations, minicomputers and mainframes. Not only is the source code not supplied, but also usually the license provisions explicitly forbid the user from disassembling the software, that is, from generating an assembly-language source code program by constructing it from the executable machine code. If the user were to have access to an assembly-language source-code program, then he/she would be free to study the program, attain an understanding of how it works, and possibly modify the program, either to fix program errors or to introduce new functionality not present in the original version. Having access to the original source code is more valuable in this regard than is a disassembled program, because the original source code usually contains identifiers (names of variables and of methods or functions or procedures) that are meaningful and that hint at the function carried out by the item identified. In addition, the original source code usually includes embedded comments that are put there specifically to help the person reading the code to understand the design concept as well as the function or purpose of constants, variables, methods, functions, procedures, or of individual lines or groups of lines of assembly code. A disassembled program, on the other hand, will usually assign identifiers arbitrarily, and therefore the identifier names provide no hint whatsoever at the functionality of the identified items. Also, of course, a disassembled program will include absolutely no explanatory comments.

In following a policy of both not revealing the source code and also explicitly forbidding the user from disassembling the executable code, the operating-system vendor typically keeps the user dependent upon him both for fixing any errors that may be present and for making any needed functional improvements. The vendors' policy is eloquently summarized by Richard M. Stallman (2001): "If you want any changes, beg us to make them."

The history presented here of the roots of *Linux* in the *UNIX* world has necessarily been very brief. Much more extensive coverage of the history of *UNIX* can be found in the separate chapter in this volume covering *UNIX*.

c. Richard Stallman, the Free Software Foundation, and the GNU Project

Stallman had watched the *UNIX* environment deteriorate from one of cooperative development, with a lot of sharing of code and ideas and a spirit of community, to an insular environment in which individual vendors competed with each other and foreclosed any possibility of user involvement in the development of the operating system or of mutual cooperation to fix and improve. In reaction to this development, Stallman founded the Free Software Foundation (FSF) in 1984, and also initiated the FSF's major activity, the *GNU* Project. He actually quit his job in the MIT Artificial Intelligence Laboratory, both so that he could devote himself fully to the project, and also in order that his employer would not be able to claim the developed system as its intellectual property.

The principal project undertaken by the Free Software Foundation was the development of a *UNIX*-like operating system that would be totally free of proprietary code and thus would be open, giving programmers the ability to study and modify the code at will and, in particular, to share in the benefit made by improvements contributed by others. The overall name for the project was "*GNU*", a name that stands recursively for, "*GNU* is Not Unix". Development started on proprietary *UNIX* systems, but as various code modules were developed, these were swapped in one by one in exchange for their proprietary cousins, and eventually the entire system was implemented in non-proprietary Open-Source ("free") code.

The major goal of the *GNU* project was to produce a *UNIX*-like operating system totally as an Open-Source product. Nevertheless, several decisions were made that exerted strong influence on the technical aspects of the project. Three of these are especially worthy of note. First, although 32-bit microprocessors were relatively new in 1984 when the project was initiated, nevertheless it was decided that 16-bit processors would not be supported. Second, the traditional *UNIX* emphasis on minimization of memory usage was dropped, at least for utilization of up to one megabyte. Finally, wherever possible dynamically allocated data structures were used, in order to avoid the problem of arbitrarily determined size limits. These decisions have had far-reaching effect on the efficiency and performance of *GNU/Linux*.

The various parts of *GNU* were developed over a period of several years, leaving the kernel for last. The concept adopted for the *GNU* kernel was that it would be based upon the Mach microkernel. Mach was originally developed at Carnegie-Mellon University, and was subsequently continued at the University of Utah. The major advance of Mach over previous *UNIX* and *UNIX*-like kernels was its capability to support loosely-coupled multiple processors. The *GNU* kernel was therefore based upon a collection of servers, in the terminology of the project a “herd of *GNUs*”. It was therefore given the name “*HURD*”¹. This was a functionally very useful concept, but it did have a distinct downside. Development of the *GNU* kernel was very much drawn out in time because of the difficulty involved in debugging the asynchronous multi-threaded servers inherent in the concept that must pass messages to each other (Stallings, 2000). This extended development time for the *GNU* kernel was the sole remaining obstacle to fielding the entire *GNU* operating system. Thus, the time was ripe for Linus Torvalds to step in with his *Linux* kernel. This had a less ambitious design concept than the *HURD* kernel being worked on by the *GNU* stalwarts, but although differently conceived, nevertheless it was very well matched to all of the extra-kernel elements of the *GNU* project. It came along at just the right time to be plugged into the main body of *GNU* software and thus to complete the free and open-source *UNIX*-like operating system that was the goal of *GNU*.

d. Andrew Tanenbaum and Minix

Due to the evolution of *UNIX* into a number of disparate proprietary dialects, as well as its ever-increasing complexity and the absence of source code, *UNIX* had evolved into a system that was no longer well suited to teaching about operating systems. Two well-known faculty members responded to this situation by designing *UNIX*-like operating-system kernels that were intended to meet pedagogic needs. Robert Comer developed *XINU* (note that *XINU* is *UNIX* spelled backwards), and Andrew Tanenbaum produced *MINIX*. These were both available at low cost, and with complete disclosure of source code, thus lending themselves well to use by students. The student could study and understand all of the code, and could also produce his own modifications. From a pedagogical perspective, this results in a very good grasp by the student of how an operating system works.

e. Linus Torvalds and the Initial Development of Linux

A graduate student in Computer Science named Linus Torvalds (pronunciation: Lee'-nuhs) studied operating systems in 1991 at the University of Helsinki in Finland, using Andrew Tanenbaum's textbook and the *UNIX*-like kernel called *MINIX* that Tanenbaum had developed. Torvalds was not satisfied, however, both with the *MINIX* kernel, because of its limited capability, and also with the *MINIX* file system. He therefore decided to try to develop a more

¹ One of the delightful features of the *Linux* culture is the sense of humor that pervades it.

capable as well as a pedagogically useful *UNIX*-like kernel on his own. Torvalds recognized, however, that this was a major undertaking, particularly in light of the modest amount of experience that he had had with operating systems at that time (he was then a relatively junior graduate student). Therefore, when he started work on the new kernel in April 1991, he posted a notice in a widely read news group, to inform others of his new project and to solicit their collaboration. The solicitation was successful, the code was posted, corrections were made and also posted, and the first kernel was released publicly in October 1991. Originally, this was just a fun pastime for computer geeks. However, the code evolved into a mature form, and eventually a stable version of the kernel was produced. The first release of the stable kernel took place in March 1994.

f. Why Is Linux So Much More Popular than its Progenitors?

Linux greatly exceeds in popularity its immediate ancestor, *Minix*, as well as most flavors of *UNIX*, from which it is derived. It is much more robust and capable than *Minix*, probably due in large measure to its multi-authored origin together with the “Copyleft” provisions that were so critical in encouraging voluntary contributions to the development of the code. And its magnificent stability, together with the uniformity of look and feel across platforms, are probably the major reasons, in addition to the openness of the source code, for the headway made by *Linux* against the various proprietary flavors of *UNIX*.

g. Origin, Appropriateness, and Proper Pronunciation of the name Linux

Torvalds originally used the name *Linux* only privately, and attempted publicly to name his new kernel *Freax*. An associate, Ari Lemmke, who ran the FTP site used originally to distribute the code, didn’t like that name, and therefore decided to use *Linux* instead. Thus, *Linux* was born.

Strictly speaking, the name *Linux* should be applied only to the kernel, which is the particular contribution made by Linus Torvalds and collaborators. The great majority of the code included in the various distributions of *Linux* belongs not to the kernel but to other, non-continuously-memory-resident parts of the operating system, such as those comprising the *GNU* project, which was nearly complete except for the kernel before Linus Torvalds came on the scene. Richard Stallman (2000) has contended that the most appropriate name for the total software package **should** really be *GNU/Linux*, reflecting the fact that the contribution of *GNU* code to the entire enterprise is greater than that of *Linux*, and that the name *Linux* belongs principally just to the kernel. Despite the attractiveness of Stallman’s point, however, most of the world community has come to accept the name *Linux* for the entire package. This may be not completely fair to the many programmers who made immense contributions of time and effort to the total body of the software both in pre-Linus Torvalds days and subsequently, and in

particular to Richard Stallman himself, who almost single-handedly developed and promoted the concept of free software, together with the “Copyleft” that has been crucial to its success, and who also initiated the *GNU* project, nurtured it, and led it to success by dint of substantial personal sacrifice as well as incisive leadership. Nevertheless, the world has accepted the name *Linux* for the entire package, including the part contributed by *GNU*, and therefore the name *Linux* is retained here.

What is the proper pronunciation of *Linux*? There are two principal streams of thought. Some computer folk use a pronunciation based upon the Anglicized pronunciation of Torvalds’ first name. Native speakers of English commonly butcher the pronunciation of other peoples’ languages, including the pronunciation of foreign names. Thus, the typical pronunciation of the name Linus by most speakers of English is **Lye’-nuhs**. This pronunciation is normally used, for example, both for the name of the character in Charles Schultz’s “Charlie Brown” cartoon series, as well as for the name of the two-time winner of the Nobel Prize (chemistry, and the Nobel Peace Prize), Prof. Linus Pauling of Cal Tech. The sound of Linus as pronounced by most speakers of English is similar to the English pronunciation of the Persian name Cyrus (**Sigh’-ruhs**). The equivalent pronunciation of the name of the operating system, *Linux*, would thus be **Lye’-nuks**. However, in Swedish² the correct pronunciation of Linus is **Lee’-noos**, and Linus Torvalds himself uses the pronunciation **Lee’-nooks** for the name of the operating system. This author is strongly of the opinion that English-speaking people should pay foreigners the courtesy of at least attempting to pronounce their names authentically, and is therefore a strong advocate of the pronunciation **Lee’-nooks**. The modern miracle of the Internet allows us all to hear the actual voice of Linus Torvalds pronouncing the name *Linux*. The URL to hear him do that is: <ftp://ftp.kernel.org/pub/linux/kernel/SillySounds/english.au>

h. GNU and “Copyleft”, and their impact on the evolution of Linux

In setting up the Free Software Foundation, Richard Stallman very creatively adapted the provisions of copyright law to his purpose of assuring both: (a) that the source code would perpetually remain open and (b) that a vendor would be strictly prevented from making derivative works from the free software and making these derivative works proprietary, and then legally exerting control over the users and enticing them away from the free product towards the vendor’s proprietary derivatives. To accomplish these goals, Stallman copyrighted the software, and required, as a condition for obtaining a license to use it, that all derivative works made from Free-Software-Foundation-produced software and subsequently copied and distributed, be covered by the identical copyright. There could be no license fee required for the use of the

² According to Eric S. Raymond’s “Rampantly Unofficial Linus Torvalds FAQ” (<http://www.tuxedo.org/~esr/faqs/linux/>), although Linus Torvalds comes from Finland, nevertheless his native language is not Finnish but Swedish, there being a considerable minority of native Swedish speakers in Finland. Finland has two official languages, Finnish and Swedish, and residents of localities that have substantial native-Swedish population are entitled by law to carry out their communications with government entities in Swedish (<http://virtual.finland.fi/finfo/english/finnswedes.html>).

software, although a modest charge could be made to defray the cost of distribution media. Also, it was required that the source code either be distributed in its entirety together with the executable code or else be made readily available for no more than a modest charge for reproduction. Deliberate obfuscation of the source code is prohibited, as is restriction on further modification and derivation of new software (www.opensource.org).

A friend of Stallman's, Don Hopkins, sent him a letter around 1984/5, on the envelope of which he had scribbled, "Copyleft—all rights reversed." This summed up very nicely Stallman's use of copyright law to accomplish the opposite of what copyright normally does. Instead of restricting the user's right to the software, it guaranteed completely unobstructed right. Therefore, Stallman adopted Hopkins' term, "copyleft", and used it to refer to the **GNU** Public License Agreement, or GPL. This remains the standard license under which all **GNU** software and the **Linux** kernel are distributed today.

i. Worldwide Cooperation and Support: a Grassroots Movement

Linux is not just an operating system; it is a culture. The concepts promulgated by Richard Stallman under the aegis of his brainchild, the Free Software Foundation, have become the bywords of the **Linux** movement. There is a small army of enthusiasts who enjoy the camaraderie and the easygoing humor of the **Linux** community. Anyone may participate who has the technical smarts, as well as the time and will. The **Linux** community is a true meritocracy. No one needs to know, nor does anyone care, what is your race, religion, nationality, or sexual orientation, what you look like, or how you dress. Whether you have a pleasant or an obnoxious personality makes very little difference. Unlike the situation in the commercial world, there are no supervisors or executives to suck up to; all that counts is the quality of your contribution. And the key to its astonishing success has been the Internet. As Linus Torvalds himself has said, "One of the most important and unique facets of the **Linux** development project has been the effect that feedback (mostly via the Internet) has on development: feedback accelerates development dramatically" (foreword to Sobell, 1997). Also, "**Linux** wouldn't be what it is today without the Internet and the contributions of an incredible number of people" (*op. cit.*).

3. Hardware Architectures Supported

In the commercial environment, in most instances an operating system is designed for a single hardware architecture. In fact, the kernel of the operating system, which is the part that must reside in memory starting immediately after boot-up and continuing until system shut-down, is highly specific to the hardware architecture, so much so that the name "**UNIX**", for example, although widely thought of as a single operating system, is, in reality, a family of

operating systems, since the kernel that implements *UNIX* on each and every different hardware platform is different from every other *UNIX* kernel. This despite the fact that the different versions of *UNIX* implemented on two particular hardware architectures may have a totally common look and feel.

In the *UNIX* world, in fact, in addition to the various kernels each specific to a particular hardware architecture, there are also several varieties of *UNIX*, of which the major variants are AT&T's System V Release 4 (SVR4) and the Berkeley Software Distribution (BSD). The look and feel of these major variants is similar, but definitely not identical.

In the case of *Linux*, again we are really dealing not just with one single operating system, but rather with a family of operating systems. However, *Linux* comes in only one flavor, the look and feel being nearly identical across all hardware architectures. The kernel is standardized for each hardware family. *Linux* runs on quite a number of hardware architectures, including the Amiga, the Apple-IBM-Motorola PowerPC, the Atari, the DEC/Compaq/Hewlett-Packard Alpha, the Intel 32-bit processor family (consisting of the 80386, 80486, Pentium, PentiumII, PentiumIII, Pentium4, Celeron, and Xeon), the MIPS 10000 series, the Motorola M68000 series, the IBM System/390 zSeries iSeries architecture, and also the 64-bit architectures produced by Intel (Itanium) and AMD (Sledgehammer).

4. Distributions of *Linux* and Vendor Support

A *distribution* of *Linux* consists of a package containing all of the software necessary both to install *Linux* and to run it. There is an Internet site that lists many *Linux* distributions (www.linuxhq.com/). A recent search on this site revealed 164 distributions. Some of these are specifically geared to particular languages. There are, for example, distributions listed for the Arabic language (www.haydarlinux.com/p), as well as for Hebrew (ivrix.org.il/). There are seven principal *Linux* distributions for English-speaking users. One of these, Debian, is produced by a non-profit corporation called "Software in the Public Interest, Inc.". The Debian *Linux* distribution is produced through the contribution of volunteers. Other commonly-used *Linux* distributions are commercial, and are listed in alphabetical order, together with the percentage of users reported to have installed each distribution, according to the *Linux* Counter (counter.li.org): Caldera (percentage not reported), Connectiva (1.2%), Corel (percentage not reported), Mandrake (20%), Red Hat (30%), Slackware (12.1%), and SuSE (11.7%). The non-commercial Debian distribution is reported to be installed on 13.2% of users' machines. Thus, the five most commonly used distributions account collectively for more than 85% of the installed base of machines. On 30 May 2002, four corporations that produce *Linux* distributions announced that they were combining forces to attain a uniformity of their distributions. The four firms are Caldera, Connectiva, SuSE, and Turbolinux, and they are naming their uniform version of *Linux* UnitedLinux.

5. Extending Your Knowledge about *Linux*

Following are the various sources cited in the body of the text. These are mostly traditional-style publications (i.e., hardcopy books and articles published in hardcopy journals) that provide in most instances a solid start in the use and also in the care and feeding of *Linux*. In some instances, the content is either also or only accessible via the Internet:

BOVET, DANIEL P.; & CESATI, MARCO (2001). *Understanding the Linux Kernel. First Edition.* Sebastopol, CA: O'Reilly & Associates. ISBN 0-596-00002-2; O'Reilly Order Number: 0022.

KOFLER, MICHAEL (1999). *Linux. Second Edition.* Reading, MA: Addison-Wesley. ISBN 0-201-59628-8. [NOTE: This book includes two CDs containing part of the Red Hat *Linux* distribution. Red Hat has produced several distributions since the one on the accompanying CDs, but in any case the earlier distribution is certainly adequate to get started learning *Linux*, and it is always possible to update later.]

MCCARTY, BILL (1990). *Learning Red Hat LINUX.* Sebastopol, CA: O'Reilly & Associates, Inc. ISBN 1-56592-627-7. [NOTE: This book includes a CD containing part of the Red Hat *Linux* distribution. Red Hat has produced several distributions since the one on the accompanying CD, but in any case the earlier distribution is certainly adequate to get started learning *Linux*, and it is always possible to update later.]

MOODY, GLYN (1997). "The Greatest OS that (N)ever Was." *Wired*, 5.08: August 1997. URL: www.wired.com/wired/archive/5.08/linux_pr.html

PETERSEN, RICHARD (2001a). *Linux: The Complete Reference. Fourth Edition.* Berkeley, CA: Osborne/McGraw-Hill. ISBN 0-07-212940-9. [Comes with significant parts of several *Linux* distributions, including Caldera ("OpenLinux eDesktop and eServer"), Red Hat, and SuSE on several compact disks.]

PETERSEN, RICHARD (2001b). *Red Hat Linux 7.2: The Complete Reference. Second Edition.* Berkeley, CA: Osborne/McGraw-Hill. ISBN 0-07-212178-3. [Comes with a significant part of the Red Hat *Linux* distribution on compact disk.]

RED HAT (2002a). *The Official Red Hat Linux x86 Installation Guide.* (for Red Hat *Linux* 7.2): available on-line at: www.redhat.com/docs/manuals/linux/RHL-7.2-Manual/install-guide/, and also available for downloading in printable Adobe Acrobat (*.pdf) format at: www.redhat.com/docs/manuals/linux/ [NOTE that Red Hat also has installation guides for the Hewlett-Packard/Compaq/DEC Alpha, for the Intel Itanium, for IBM's

Introduction to *Linux*, by Charles Abzug

- eServer zSeries of System/390 mainframes, in addition to the Intel x86 processor series.]
- RED HAT (2002b). *Getting Started Guide* (for Red Hat *Linux* 7.2): Available for downloading in printable Adobe Acrobat (*.pdf) format at: www.redhat.com/docs/manuals/linux/
- SARWAR, SYED MANSOOR; KORETSKY, ROBERT; & SARWAR, SYED AQEEL (2002(SIC!)). *LINUX: The Textbook. First edition*. Boston, MA: Addison Wesley Longman, Inc. QA76.76.063 S35545 2001; 005.4'32—dc21; 00-069963; ISBN 0-201-72595-9(pbk.). [NOTE: This book includes a CD containing part of the Mandrake *Linux* distribution.]
- SOBELL, MARK G. (1997). *A Practical Guide to LINUX*. Reading, MA: Addison-Wesley Publishing Company. QA76.76.063S5948 1997; 005.4'469—dc21; 97-8248; ISBN 0-201-89549-8. [NOTE: This is a superbly written book. It delves very thoroughly and very systematically into the technical aspects of working with *Linux*. Linus Torvalds learned about *UNIX* and was inspired to work with it through a predecessor to this book. Probably the best all-around work on *Linux* available today.]
- SOBELL, MARK G. (2003). *A Practical Guide to Red Hat LINUX*. Reading, MA: Addison-Wesley Publishing Company. ISBN 0-201-70313-0.
- STALLINGS, WILLIAM (2001). *Operating Systems: Internals and Design Principles. Fourth Edition*. Upper Saddle River, NJ: Prentice-Hall. QA76.9C643673 2000; 004.2'2—dc21; 00-7405; ISBN 0-13-031999-6.
- STALLMAN, RICHARD (1998). “Linux and the GNU Project.” URL: www.gnu.org/gnu/linuxandgnu.html
- STALLMAN, RICHARD (2000). “What’s in a Name?” URL: www.gnu.org/gnu/why-gnu-linux.html
- STALLMAN, RICHARD (2001). “The GNU Project.” Originally published in: DIBONA, CHRIS; STONE, MARK; OCKMAN, SAM, editors (1999). *Open Sources: Voices from the Open Source Revolution*. Sebastopol, CA: O’Reilly & Associates. ISBN 1565925823 (but subsequently revised). URL: www.gnu.org/gnu/thegnuproject.html

Following are several modern-type Internet sources that provide a wealth of information regarding *Linux*. In many cases, these sources, in true Internet tradition, are frequently updated to reflect changes occurring in the *Linux* world:

Introduction to *Linux*, by Charles Abzug

History and Development of ***Linux***: www.wired.com/wired/archive/5.08/linux.html
counter.li.org/ (an organization that compiles statistics on ***Linux*** usage)

Open Source Initiative: Open Source Definition. www.opensource.org
www.gnu.org

Linux Headquarters: www.LINUXhq.com

Linux Documentation Project: tldp.org/

Linux Distributions: www.fokus.gmd.de/LINUX/LINUX-distrib.html (describes several distributions)
www.caldera.com (Caldera distribution)
en.conectiva.com/ (Connectiva distribution)
linux.corel.com/ (Corel distribution)
www.debian.org (Debian distribution)
www.mandrakelinux.com/en/ (Mandrake distribution)
www.redhat.com/ (Red Hat distribution)
www.slackware.com/ (Slackware distribution)
www.suse.com (SuSE distribution)

Linux Kernel Archives: www.kernel.org

Linux Journal: www2.linuxjournal.com/

NOTE that there are also various ***Linux*** Users' Groups (LUGs) that are excellent forums for dissemination of information.

6. ACKNOWLEDGMENTS

I thank my son, Mordechai Tsvi Abzug, for several conversations about ***Linux*** in which I bounced some ideas off him and obtained valuable feedback which contributed to the content of this article. I also thank Dr. Mal Lane, head of the Computer Science Department at James Madison University, for giving me the opportunity to teach Operating Systems.