1.1 Introduction

Welcome to the world of operating systems. During the past several decades, com-
puting has evolved at an unprecedented pace. Computer power continues to
increase at phenomenal rates as costs decline dramatically. Today, computer users
have desktop workstations that execute billions of instructions per second (BIPS),
and supercomputers that execute over a trillion instructions per second have been
built, 1,2 numbers that just a few years ago were inconceivable.

Processors are becoming so inexpensive and powerful that computers can be
employed in almost every aspect of our lives. On personal computers, users can
edit
documents, play games, listen to music, watch videos and manage their personal
finances. Portable devices, including laptop computers, personal digital assistants
(PDAs), cell phones, and MP3 players all have computers as key components.
Wired
and wireless network architectures are increasing our interconnectivity, allowing
users to communicate instantly over vast distances. The Internet and World Wide
Web
have revolutionized business, creating demand for networks of large, powerful
com-
puters that service vast numbers of transactions per second. Networks of computers
have become so powerful that they are used to conduct complex research and
simula-
tion projects, such as modeling the Earth's climate, emulating human intelligence
and
constructing lifelike 3D animations. Such pervasive and powerful computing is
reshaping the roles and responsibilities of operating systems.

In this book, we review operating system principles and discuss cutting-edge
advances in computing that are redefining operating systems. We investigate the

structure and responsibilities of operating systems. Design considerations, such as performance, fault tolerance, security, modularity and cost, are explored in detail. We
also address more recent operating system design issues arising from the rapid growth
in distributed computing, made possible by the Internet and the World Wide Web.

1.2 What Is an Operating System?

In the 1960s, the definition of an operating system might have been the software that
controls the hardware. But the landscape of computer systems has evolved significantly since then, requiring a richer definition.

Today's hardware executes a great variety of software applications. To increase hardware utilization, applications are designed to execute concurrently. If these applications are not carefully programmed, they might interfere with one another. As a result, a layer of software called an operating system separates applications from the hardware they access and provides services that allow each application to execute safely and effectively.

An operating system is software that enables applications to interact with a computer's hardware. The software that contains the core components of the operating system is called the kernel. Operating systems can be found in devices ranging
from cell phones and automobiles to personal and mainframe computers. In most computer systems, a user requests that the computer perform an action (e.g., execute an application or print a document) and the operating system manages the software and hardware to produce the desired result.

To most users, the operating system is a “black box” between the applications and the hardware they run on that ensures the proper result, given appropriate inputs. Operating systems are primarily resource managers-they manage hardware, including processors, memory, input/output devices and communication devices. They must also manage applications and other software abstractions that, unlike hardware, are not physical objects.

Self Review

1. (T/F) Operating systems manage only hardware.

2. What are the primary purposes of an operating system?

Ans: 1) False. Operating systems manage applications and other software abstractions, such

as virtual machines. 2) The primary purposes of an operating system are to enable applica-

tions to interact with a computer's hardware and to manage a system's hardware and soft-

ware resources.

1.3 Early History: The 1940s and 1950s

Operating systems have evolved over the last 60 years through several distinct phases or generations that correspond roughly to the decades (see the Operating Systems Thinking feature, Innovation). In the 1940s, the earliest electronic digital computers did not include operating systems.4,5,6 Machines of the time were so primitive that programmers often entered their machine-language programs one bit at a time on rows of mechanical switches. Eventually, programmers entered their machine-language programs on punched cards. Then, assembly languages-which used English-like abbreviations to represent the basic operations of the computer-were developed to speed the programming process.

General Motors Research Laboratories implemented the first operating sys-

tem in the early 1950s for its IBM 701 computer. The systems of the 1950s gener-ally executed only one job at a time, using techniques that smoothed the transition between jobs to obtain maximum utilization of the computer system. A job consti-

tuted the set of program instructions corresponding to a particular computational task, such as payroll or inventory. Jobs typically executed without user input for minutes, hours or days. These early computers were called single-stream batch-pro-
cessing systems, because programs and data were submitted in groups or batches by
loading them consecutively onto tape or disk. A job stream processor read the job control language statements that defined each job) and facilitated the setup of the next job. When the current job terminated, the job stream reader read in the con-trol-language statements for the next job and performed appropriate housekeeping chores to ease the transition to the next job. Although operating systems of the 1950s reduced interjob transition times, programmers often were required to directly control system resources such as memory and input/output devices. This was slow, difficult and tedious work. Further, these early systems required that an entire program be loaded into memory for the program to run. This limited pro-grammers to creating small programs with limited capabilities.

1.4 The 19605

The systems of the 1960s were also batch-processing systems, but they used the computer's resources more efficiently by running several jobs at once. Systems included many peripheral devices such as card readers, card punches, printers, tape drives and disk drives. Any one job rarely used all the system's resources efficiently.

A typical job would use the processor for a certain period of time before perform-ing an input/output (I/O) operation on one of the system's peripheral devices. At this point, the processor would remain idle while the job waited for the I/O opera-tion to complete.

The systems of the 1960s improved resource utilization by allowing one job to use the processor while other jobs used peripheral devices. In fact, running a mix-

ture of diverse jobs--some jobs that mainly used the processor (called processor-
bound jobs or compute-bound jobs) and some jobs that mainly used peripheral
devices (called I/O-bound jobs)-appeared to be the best way to optimize resource
utilization. With these observations in mind, operating systems designers developed
multiprogramming systems that jobs at once.10, 11, 12 In a multipro-
gramming environment, the op pidly switches the processor from
job to job, keeping several jobs also keeping peripheral devices in
use. A system's degree of multi o called its level of multiprogram-
ming) indicates how many jobs I at once. Thus, operating systems
evolved from managing one job ral jobs at a time.

In multiprogrammed computing systems, resource sharing is one of the pri-
mary goals. When resources are shared among a set of processes, each process
maintaining exclusive control over particular resources allocated to it, a process
may be made to wait for a resource that never becomes available. If this occurs, that
process will be unable to complete its task, perhaps requiring the user to restart it,
losing all work that the process had accomplished to that point. In Chapter 7, Dead-

Normally, users of the 1960s were not present at the computing facility when
their jobs were run. Jobs were submitted on punched cards or computer tapes and
remained on input tables until the system's human operator could load them into the
computer for execution. Often, a user's job would sit for hours or even days before it
could be processed. The slightest error in a program, even a missing period or comma,

would "bomb” the job, at which point the (often frustrated) user would correct the

error, resubmit the job and once again wait hours or days for the next attempt at exe-

cution. Software development in that environment was painstakingly slow.

More advanced operating systems were developed to service multiple interac-

tive users at once. Interactive users communicate with their jobs during execution. In

the 1960s, users interacted with the computer via "dumb terminals" (i.e., devices that

supplied a user interface but no processor power) which were online (i.e., directly

attached to the computer via an active connection). Because the user was present and

interacting with it, the computer system needed to respond quickly to user requests;

otherwise, user productivity could suffer. As we discuss in the Operating Systems

Thinking feature, Relative Value of Human and Computer Resources, increased pro-

ductivity has become an important goal for computers because human resources are

extremely expensive compared to computer resources. Timesharing systems were

developed to support simultaneous interactive users.22

Many of the timesharing systems of the 1960s were multimode systems that

supported batch-processing as well as real-time applications (such as industrial pro-

cess control systems).23 Real-time systems attempt to supply a response within a

certain bounded time period. For example, a measurement from a petroleum refin-

ery indicating that temperatures are too high might demand immediate attention to

avert an explosion. The resources of a real-time system are often heavily underuti-

lized - it is more important for such systems to respond quickly than it is for them

to use their resources efficiently. Servicing both batch and real-time jobs meant that
operating systems had to distinguish between types of users and provide each with an appropriate level of service. Batch-processing jobs could suffer reasonable delays, whereas interactive applications demanded a higher level of service and real-time systems demanded extremely high levels of service.

Turnaround time-the time between submission of a job and the return of its results-was reduced to minutes or even seconds. The programmer no longer needed to wait hours or days to correct even the simplest errors. The programmer could enter a program, compile it, receive a list of syntax errors, correct them immediately, recompile and continue this cycle until the program was free of syntax
errors. Then the program could be executed, debugged, corrected and completed with similar time savings.

1.5 The 1970s

The systems of the 1970s were primarily multimode multiprogramming systems that
supported batch processing, timesharing and real-time applications. Personal computing was in its incipient stages, fostered by early and continuing developments in microprocessor technology.43 The experimental timesharing systems of the 1960s evolved into solid commercial products in the 1970s. Communications between computer systems throughout the United States increased as the Department of Defense's TCP/IP communications standards became widely used-especially in military and university computing environments.44,45,46 Communication in local

area networks (LAN) was made practical and economical by the Ethernet stan-
dard developed at Xerox's Palo Alto Research Center (PARC).47,48 In Chapter 16,
Security problems increased with the growing volumes of information passing
over vulnerable communications lines (see the Anecdote, Abraham Lincoln's Tech-
The personal computing revolution began in the late 1970s with such systems
as the Apple II, and exploded in the 1980s.

1.6 The 1980s

The 1980s was the decade of the personal computer and the workstation.49 Micro-
processor technology evolved to the point where high-end desktop computers
called workstations could be built that were as powerful as the mainframes of a
decade earlier. The IBM Personal Computer released in 1981 and the Apple Macin-
tosh personal computer released in 1984 made it possible for individuals and small
businesses to have their own dedicated computers. Communication facilities could
be used to transmit data quickly and economically between systems. Rather than
bringing data to a central, large-scale computer installation for processing, comput-
ing was distributed to the sites at which it was needed. Software such as spreadsheet
programs, word processors, database packages and graphics packages helped drive
the personal computing revolution by creating demand from businesses that could
use these products to increase their productivity.

Personal computers proved to be relatively easy to learn and use, partially
because of graphical user interfaces (GUI) that used graphical symbols such as win-
dows, icons and menus to facilitate user interaction with programs. Xerox's Palo
Alto Research Center (PARC) developed the mouse and GUI (for more on the ori-
gins of the mouse, see the Biographical Note, Doug Engelbart); Apple's release of
the Macintosh personal computer in 1984 popularized their use. In Macintosh com-

puters, the GUI was embedded in the operating system so that all applications would have a similar look and feel.50 Once familiar with the Macintosh GUI, the user could learn to use new applications faster.

As technology costs declined, transferring information between computers in computer networks became more economical and practical. Electronic mail, file transfer and remote database access applications proliferated. Distributed computing (i.e., using multiple independent computers to perform a common task) became widespread under the client/server model. Clients are user computers that request various services; servers are computers that perform the requested services. Servers often are dedicated to one type of task, such as rendering graphics, managing databases or serving Web pages.

1.7 History of the Internet and World Wide Web

In the late 1960s ARPA-the Advanced Research Projects Agency of the Department of Defense rolled out the blueprints for networking the main computer systems of about a dozen ARPA-funded universities and research institutions. They were to be connected with communications lines operating at a then-stunning 56 kilobits per second (Kbps)-1 Kbps is equal to 1,000 bits per second-at a time when most people (of the few who could be) were connecting over telephone lines to computers at a rate of 110 bits per second. HMD vividly recalls the excitement at

that conference. Researchers at Harvard talked about communicating with the Univac 1108 “supercomputer" across the country at the University of Utah to handle the massive computations related to their computer graphics research. Academic research was about to take a giant leap forward. Shortly after this conference, ARPA proceeded to implement what quickly became called the ARPAnet - the grandparent of today's Internet.

The protocols (i.e., sets of rules) for communicating over the ARPAnet became known as the Transmission Control Protocol/Internet Protocol (TCP/IP). TCP/IP was used to manage communication between applications. The protocols ensured that messages were routed properly from sender to receiver and that those messages arrived intact. The advent of TCP/IP promoted worldwide computing growth. Initially, Internet use was limited to universities and research institutions; later, the military adopted the technology.

The World Wide Web (WWW) allows computer users to locate and view multimedia-based documents (i.e., documents with text, graphics, animation, audio or video) on almost any subject. Although the Internet was developed more than three decades ago, the introduction of the World Wide Web (WWW) was a relatively recent event. In 1989, Tim Berners-Lee of CERN (the European Center for Nuclear Research) began to develop a technology for sharing information via hyperlinked text documents (see the Biographical Note, Tim Berners-Lee). To

1.8 The 1990s

Hardware performance continued to improve exponentially in the 1990s.69 By the end of the decade, a typical personal computer could execute several hundred million instructions per second (MIPS) and store over a gigabyte of information on a hard disk; some supercomputers could execute over a trillion operations per second.70 Inexpensive processing power and storage enabled users to execute large, complex programs on personal computers and enabled small- to mid-size companies to use these economical machines for the extensive database and processing jobs that were once delegated to mainframe systems. Falling technology costs also led to an increase in the number of home computers, which were used both for work and for entertainment.

In the 1990s, the creation of the World Wide Web led to an explosion in the popularity of distributed computing. Originally, operating systems performed isolated resource management inside a single computer. With the creation of the

World Wide Web and increasingly fast Internet connections, distributed computing became commonplace among personal computers. Users could request data stored at remote locations or request that programs run on distant processors. Large organizations could use distributed multiprocessors (i.e., networks of computers containing more than one processor) to scale resources and increase efficiency. 71 Distributed applications, however, were still limited by the fact that communication over a network occurred at relatively slow speeds compared to the internal processing speeds of individual computers. Distributed computing is discussed in detail in Microsoft Corporation became dominant in the 1990s. In 1981, Microsoft released the first version of its DOS operating system for the IBM personal computer. In the mid-1980s, Microsoft developed the Windows operating system, a graphical user interface built on top of the DOS operating system. Microsoft released Windows 3.0 in 1990; this new version featured a user-friendly interface and rich functionality. The Windows operating system became incredibly popular after the 1993 release of Windows 3.1, whose successors, Windows 95 and Windows 98, virtually cornered the desktop operating system market by the late 90s. These operating systems, which borrowed from many concepts (such as icons, menus and windows) popularized by early Macintosh operating systems, enabled users to navigate multiple concurrent applications with ease. Microsoft also entered the corporate operating system market with the 1993 release of Windows NT, which quickly became the operating system of choice for corporate workstations. 72 Windows XP,

Open-Source Movement

Another development in the computing community (particularly in the area of oper-

ating systems) during the 1990s was the movement toward open-source software.
Most software is created by writing source code in a high-level programming language. However, most commercial software is sold as object code (also called machine code or binaries)-the compiled source code that computers can understand. The source code is not included, enabling vendors to hide proprietary information and programming techniques. However, free and open-source software became increasingly common in the 1990s. Open-source software is distributed with the source code, allowing individuals to examine and modify the software before compiling and executing it. For example, the Linux operating system and the Apache Web server, both of which are free and open source, were downloaded and installed by millions of users during the 1990s, and the number of downloads is increasing rapidly in the new millennium.74 Linux, created by Linus Torvalds (see the Biographical

1.9 2000 and Beyond

In the current decade, middleware, which is software that links two separate applications (often over a network), has become vital as applications are published on the World Wide Web and consumers use them via affordable, high-speed Internet connections over cable television lines and digital subscriber lines (DSL). Middleware is common in Web applications, in which a Web server (the application that sends data to the user's Web browser) must generate content to satisfy a user's request with the help of a database. The middleware acts as a courier to pass messages between the Web server and the database, simplifying communication between multiple different architectures. Web services encompass a set of related standards that can enable any

two computer applications to communicate and exchange data via the Internet. A

Web service communicates over a network to supply a specific set of operations that

other applications can invoke. The data is passed back and forth using standard pro-

tocols such as HTTP, the same protocol used to transfer ordinary Web pages. Web

services operate using open, text-based standards that enable components written in

different languages and on different platforms to communicate. They are ready-to-

use pieces of software on the Internet.

Multiprocessor and network architectures are creating numerous opportuni-

ties for research and development of new hardware and software design techniques.

Sequential programming languages that specify one computation at a time are now

complemented by concurrent programming languages, such as Java, that enable the

specification of parallel computations; in Java the units of parallel computing are

specified via threads. We discuss threads and the technique of multithreading in

1.10 Application Bases

When the IBM Personal Computer (often called simply “the PC”) appeared in

1981, it immediately spawned a huge software industry in which independent soft-

ware vendors (ISVS) were able to market packages for the IBM PC to run under

the MS-DOS operating system (IBM's version was called DOS). Operating systems

free applications software developers from having to deal with the messy details of

manipulating computer hardware to manage memory, perform input/output, deal

with communication lines, and so on. The operating system provides a series of

application programming interface (API) calls which applications programmers use
to accomplish detailed hardware manipulations and other operations. The API provides system calls by which a user program instructs the operating system to do the work; the application developer simply has to know what routines to call to accomplish specific tasks (Fig. 1.1). Note that in Fig. 1.1, the area above the dashed line, user space, indicates software components that are not part of the operating system and cannot directly access the system's physical resources. The area below the dashed line, kernel space, indicates software components that are part of the operating system and have unrestricted access to system resources. We frequently use this convention in our diagrams to indicate the privilege with which software components execute. If an application attempts to misuse system resources, or if the application attempts to use resources that it has not been granted, the operating system must intervene to prevent the application from damaging the system or interfering with other user applications.

1.12 Operating System Components and Goals

Computer systems have evolved from early systems containing no operating system, to multiprogramming machines, to timesharing machines, to personal computers and finally to truly distributed systems. As the demand for new features and improved efficiency grew and hardware changed, operating systems evolved to fill new roles. This section describes various core operating system components and explains several goals of operating systems.

1.121 Core Operating System Components

A user interacts with the operating system via one or more user applications, and often through a special application called a shell, or command interpreter.105 Most of today's shells are implemented as text-based interfaces that enable the user to issue commands from a keyboard or as GUIs that allow the user to point and click

and drag and drop icons to request services from the operating system (e.g., to open
an application). For example, Microsoft Windows XP provides a GUI through
which users can issue commands; alternatively, the user can open a command
prompt window that accepts typed commands.

The software that contains the core components of the operating system is
referred to as the kernel. Typical operating system core components include:

- the process scheduler, which determines when and for how long a process executes on a processor.
- the memory manager, which determines when and how memory is allo- cated to processes and what to do when main memory becomes full.
- the 1/0 manager, which services input and output requests from and to hardware devices, respectively.
- the interprocess communication (IPC) manager, which allows processes to communicate with one another.

the file system manager, which organizes named collections of data on stor-
age devices and provides an interface for accessing data on those devices.

Almost all modern operating systems support a multiprogrammed environ-
ment in which multiple applications can execute concurrently. One of the most fun-
damental responsibilities of an operating system is to determine which processor
executes a process and for how long that process executes.

Self Review

1. Which operating system components perform each of the following operations?
   a. Write to disk.
   b. Determine which process will run next.
   c. Determine where in memory a new process should be placed.

d. Organize files on disk.

e. Enable one process to send data to another.

2. Why is it dangerous to allow users to perform read or write operations to any region of

disk at will?

Ans: 1) a) 1/0 manager; b) processor scheduler; c) memory manager; d) file system man-

ager; e) interprocess communication (IPC) manager. 2) It is dangerous because users could

1.12.2 Operating System Goals

Users have come to expect certain characteristics of operating systems, such as:

- efficiency
- robustness
- scalability
- extensibility
- portability
- security
- interactivity
- usability

An efficient operating system achieves high throughput and low average turn-

around time. Throughput measures the amount of work a processor can complete

within a certain time period. Recall that one role of an operating system is to provide

services to many applications. An efficient operating system minimizes the time spent

providing these services (see the Operating Systems Thinking feature, Performance).

A robust operating system is fault tolerant and reliable - the system will not

fail due to isolated application or hardware errors, and if it fails, it does so gracefully

(i.e., by minimizing loss of work and by preventing damage to the system's hardware). Such an operating system will provide services to each application unless the

hardware it relies on fails.

A scalable operating system is able to use resources as they are added. If an

operating system is not scalable, then it will quickly reach a point where additional

resources will not be fully utilized. A scalable operating system can readily adjust its

degree of multiprogramming. Scalability is a particularly important attribute of

multiprocessor systems - as more processors are added to a system, ideally the pro-

cessing capacity should increase in proportion to the number of processes, though,

in practice, that does not happen. Multiprocessing is discussed in Chapter 15, Multi-

processor Management.

An extensible operating system will adapt well to new technologies and pro-

vide capabilities to extend the operating system to perform tasks beyond its original

design.

A portable operating system is designed such that it can operate on many

hardware configurations. Application portability is also important, because it is

costly to develop applications, so the same application should run on a variety of

hardware configurations to reduce development costs. The operating system is cru-

cial to achieving this kind of portability.

A secure operating system prevents users and software from accessing ser-

vices and resources without authorization. Protection refers to the mechanisms that

implement the system's security policy.

An interactive operating system allows applications to respond quickly to user

actions, or events. A usable operating system is one that has the potential to serve a significant user base. These operating systems generally provide an easy-to-use user

interface. Operating systems such as Linux, Windows XP and MacOS X are charac-

terized as usable operating systems, because each supports a large set of applica-

tions and provides standard user interfaces. Many experimental and academic

operating systems do not support a large number of applications or provide user-

friendly interfaces and therefore are not considered to be usable.

Key Terms

interactive operating system-Operating system that allows

applications to respond quickly to user input.

interactive users - Users that are present when the system pro-

cesses their jobs. Interactive users communicate with their

jobs during execution.

Internet - Network of communication channels that provides

the backbone for telecommunication and the World Wide

Web. Each computer on the Internet determines which

services it uses and which it makes available to other com-

puters connected to the Internet.

interprocess communication (IPC) manager-Operating sys-

tem component that governs communication between

processes.

job-Set of work to be done by a computer.

kernel-Software that contains the core components of an

operating system.

layered operating system-Modular operating system that

places similar components in isolated layers. Each layer

accesses the services of the layer below and returns results
to the layer above.
level of multiprogramming-See degree of multiprogramming.
massive parallelism-Property of a system containing large
numbers of processors so that many parts of computations
can be performed in parallel.
memory manager-Operating system component that controls
physical and virtual memory.
middleware-Layer of software that enables communication
between different applications. Middleware simplifies
application programming by performing work such as net-
work communication and translation between different
data formats.
single-stream batch-processing system-Early computer sys-
tem that executed a series of noninteractive jobs sequen-
tially, one at a time.
system call-Call from a user process that invokes a service of
the kernel.
real-time system-System that attempts to service requests
within a specified (usually short) time period. In mission-
critical real-time systems (e.g., air traffic control and petro-
leum refinery monitors), money, property or even human
life could be lost if requests are not serviced on time.
operating system-Software that manages system resources to
provide services that allow applications to execute prop-
erly. An operating system may manage both hardware and
software resources. Operating systems provide an applica-

tion programming interface to facilitate application development. They also help make system resources conveniently available to users while providing a reliable, secure and responsive environment to applications and users.

thread-Entity that describes an independently executable stream of program instructions (also called a thread of execution or thread of control). Threads facilitate parallel execution of concurrent activities within a process.

throughput - Amount of work performed per unit time.

timesharing system-Operating system that enables multiple simultaneous interactive users.

Transmission Control Protocol/Internet Protocol (TCP/IP)-Family of protocols that provide a framework for networking on the Internet.

distributed operating system-Single operating system that provides transparent access to resources spread over multiple computers.

distributed system-Collection of computers that cooperate to perform a common task.

application programming interface (API)—Specification that allows applications to request services from the kernel by making system calls.