Chapter 2
History

Revised: Jan. 12, 2011

2.1 INTRODUCTION

To gain an effective understanding of the concepts and structures of today's operating systems, it is important to review the lessons we have learned through operating systems of the past. There is not time in this course for a complete historical study of operating systems. We will focus on the origins of concepts that have had a significant influence on the systems that followed.

A number of specific operating systems of historical or current importance are introduced in this chapter. Many of these are used for specific examples throughout the text. To provide a perspective on the evolution of these systems and concepts, a brief chronology is presented in Figure 2-1. This chart shows the dates of first appearance for most of the systems and concepts we will discuss.

The first step in understanding the role of operating systems is to recognize the problems that existed in computer use before operating systems were developed. In this section we will explore the computing process in such an environment. The remaining sections explain the development of important categories of operating systems, in roughly chronological order beginning with the earliest I/O supervisors, proceeding through complex OSs for large and small computers, and concluding with operating systems for personal computers and those serving specialized needs and applications, including distributed environments that are of increasing importance today.

The evolution of operating systems has proceeded in cycles. Three cycles are especially evident, corresponding to the main categories of computers that have evolved: **large mainframe computers**, smaller and less expensive **minicomputers,** and still smaller and less expensive **microcomputers,** which provide a complete computer on a few low-cost integrated circuits. The development of effective **networking** and graphic user interfaces and an increased reliance on **open, distributed services** have had revolutionary impacts on OS development as well. Most recently, **embedded systems** and **mobile systems** have come to play a major role in our computing environment.

To some degree, operating systems have evolved independently for each of these categories. The earliest machines had no operating systems and had to be programmed manually. The first OSs for each category served a single user, provided limited services, had little or no file storage, and were equipped with slow and unreliable I/O devices. Later operating systems have become much more sophisticated, managing large memory spaces, supporting a full range of I/O devices, and providing service for large numbers of interactive users. This evolutionary cycle, which is now repeating for the fourth time, is illustrated in Figure 2-2. Each cycle has proceeded markedly faster than the last, and operating systems for the smallest computers now outperform most of their larger but older siblings.

The cycle of evolution is predominantly market-driven; computer users always want more for less. The result has been continuing pressure to develop ever more powerful operating systems for similar cost, or to greatly reduce the cost, which may be the beginning of a new evolutionary cycle.

Throughout your study, observe the problems that existed to motivate each new OS concept, and note how newer operating systems have been influenced, for better or worse, by those that have gone before.

-

DATE	SYSTEMS	CONCEPTS
1950	IOS for 701	First operating systems
	GM/NAA system for 704	I/O supervision; Job sequencing
1955	SHARE OS	Assembly language support; User-driven-design
	FORTRAN Monitor System	High-level language support
	SAGE	Real-time response
	SABRE	Transaction processing
1960	TOS	Tape I/O & storage
	IBSYS	Disk OS
	DOS	System files; Improved I/O control; Job control; Serial batch processing
	ATLAS	Interrupts; System calls; Virtual memory
1965	MCP/5000	Batch multiprogramming
	OS/MFT	Spooling
	OS/MVT	Dynamic memory allocation
	CTSS	Timesharing
	TOPS-10	Command interface concepts
	DTSS	Computing for the masses
	DDP-116 IOS, OS/8 for PDP-8	First minicomputer OSs
	MULTICS	Segmented virtual memory; Dynamic linking; Hierarchical file system; I/O redirection; Device independence; Protection rings; OS written in high-level language
	T.H.E.	Hierarchical OS structure; Semaphores; Deadlock avoidance
1970	TENEX	Abstract machines; Improved file systems; Improved user interfaces
	TSS/8	Timesharing for minis
	RSX/15	Real-time laboratory control
	CP/CMS, VM/370	Virtual machines
	RT-11, RSTS, RSX-11	OS family for PDP-11
	UNIX	Designed by and for researchers; Cheap processes; Filters and pipes; Replaceable shell; New user interface ideas; New file system ideas; Portability

Figure 2-1: Evolution of Operating Systems and Concepts

-

DATE	SYSTEMS	CONCEPTS
1975	CP/M	First minicomputer OS
	MVS	Virtual memory for IBM 370
	VMS	Multiple-use support for VAX
	MS-DOS	Improved OS for micros
	Xerox ALTO & STAR	Professional workstations; Icons and mouse
	Hydra, Medusa	Parallel operating systems for C.mmp and Cm*
	StarOS	Task forces and coscheduling
1980	PC-DOS	MS-DOS for IBM micros
	Concurrent DOS	Multiuser OS for micros
	MSW & RSEXEC	Network operating systems
	Macintosh	Affordable graphics workstation; User-friendly GUI
	PICK	Integral database system
	VRTX	Portable real-time OS
1985	SUN NFS	Network file sharing
	LOCUS	Distributed transparent OS
	OS/2, AmigaDOS	Concurrency and extended features for micros
	Mach	Unix for distributed systems; Threads; Microkernel
	X Window System	Distributed GUI for Unix
	NeXTStep	Optical storage, improved user interface
	Pen Point	Pen-based OS for portable computer
	ITRON	OS interface model for embedded systems
1990	Microsoft Windows	GUI and cooperative multitasking for the IBM-PC
	Macintosh System 7	Cooperative multitasking for the Mac
	Solaris	Unix with a GUI
	Minix, Linux	Affordable Unix for PCs
	Newton OS	First OS for handheld computers
	Windows NT	Safe multitasking, advanced networking, high security, portability, virtual machine
	POSIX and OSF/1	Portability for UNIX
	OpenVMS	Portability for VMS
	CORBA and DCE	Distributed computing comes of age

Figure 2-1: Evolution of Operating Systems and Concepts (continued)

-

DATE	SYSTEMS	CONCEPTS
1995	Windows 95	Improved GUI and compatible MS-DOS replacement
	BeOS	Alternate OS for PC or MAC
	Windows NT 4.0	NT with Win 95 GUI
	Windows CE	Windows light for PDAs
	Macintosh System 8 and 9	Moving toward true multitasking
	Linux (new versions)	Open source plus support
	PalmOS, Windows CE	New design concepts for handhelds
	Windows 98	Internet on the desktop
	OS/390	MVS plus UNIX plus Internet
	MULTOS	OS for smart cards
	The World Wide Web	Global OS? We're still learning
2000	Windows ME, 2000, XP	Two Windows converge
	Macintosh OSX	Mac meets Unix
	Linux continues	True UNIX for the masses?
	z/OS	New version of OS/390; your personal mainframe?
	PalmOS 5.0, Symbian	Handhelds evolve, meet cell phones
	Java OS	It's a Java world
2005	Windows Vista	Focus on Security, networking, user tools
	Macintosh Snow Leopard	More evolution
	iOS for iPod, iPhone	More shrinking
	Windows 7	Smaller, faster, better
	Google Android	The next big thing?

Figure 2-1: Evolution of Operating Systems and Concepts (continued)

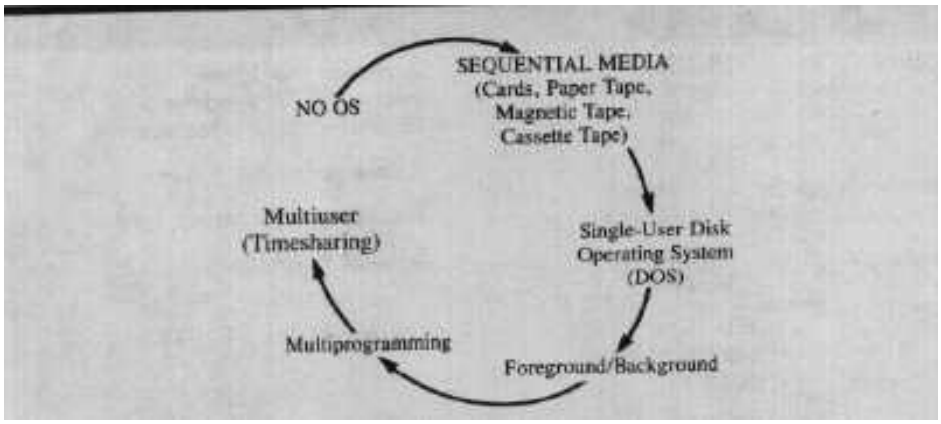

Figure 2-2: Repeated Cycles in Operating System Evolution

2.2 LIFE BEFORE OPERATING SYSTEMS

Programming with Lights and Switches

Present-day computer users may find it difficult to imagine how early computers were programmed with no operating system. However, those who lived through the early days have memories of hours spent entering programs in binary form, word by word, using switches on the computer's front panel.

In this type of programming environment, the first step was to enter the starting address for a sequence of data by setting a row of switches and pressing a button. After that, each data word in the sequence was entered in the same way. Entries could be verified by again setting a starting address and reading a sequence of words in binary form from a row of lights. If errors were found, the same procedure was used for corrections. If a single instruction had to be added to an existing sequence, perhaps dozens of instructions had to be entered again. This cumbersome method of programming is illustrated in Figure 2-3.

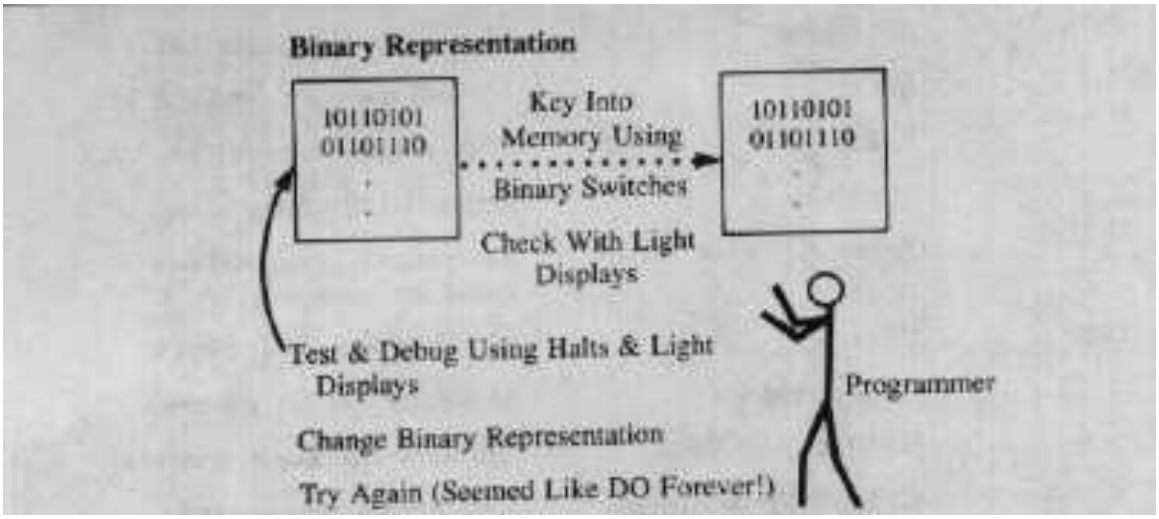

Figure 2-3: Binary Entry of Programs

Debugging of such programs was done without any operating systems or software tools. If necessary, programs could be made to execute one instruction at a time by repeatedly pressing a button. Lights on the front panel displayed the contents of selected registers or memory locations while the program was halted. These displays were studied to discover errors in the program.

While such tedious input and debugging was going on, each programmer necessarily had complete control of the computer. The term "programmer" is appropriate here; no one but trained programmers could possibly use a computer directly. Time was allocated by an **open shop** approach. Programmers signed up for a period of time, and had exclusive use of the computer for that period. Even if the programmer spent much of the time thinking about what to do next, no other programmer could use the computer in the meantime.

-

Use of Punched Cards

Gradually, the use of lights and switches was supplemented by simple input/output devices. Typewriters were attached to the computer, allowing typing of certain input data and printing of output. **Punched cards** provided a medium in which large quantities of information could be prepared in advance and stored for later use. These cards, used with business equipment long before the first electronic computers appeared, were made of stiff paper with information encoded in the form of punched holes. As shown in Figure 2-4, a large deck of punched cards could be loaded into a card reader, which would automatically read, at high speed, the information they contained. Data and instructions for a large program could be encoded into a deck of cards using a separate, typewriter-like "keypunch" device. This information could then be loaded into successive memory locations by pushing a single button. In addition, programs already loaded and running could read data from additional cards.

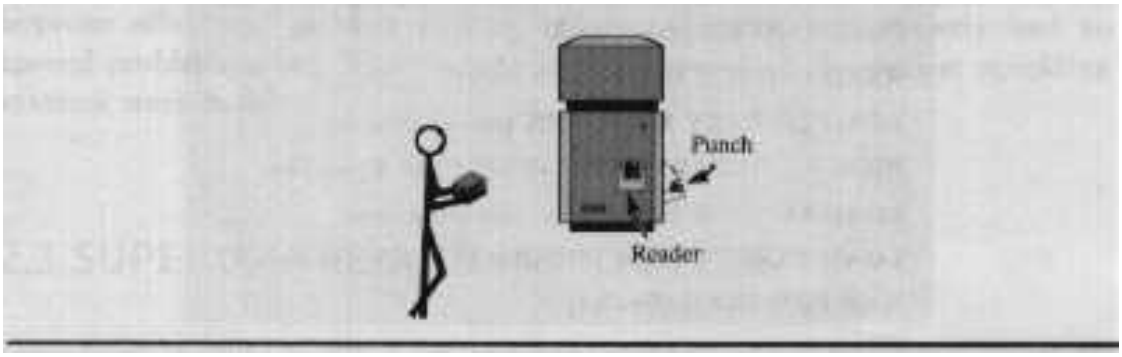

Figure 2-4: Programming with Punched Cards

Early computer systems based on punched cards were further assisted by the development of the concept of assembly language. With assemblers available, other programs could be written in a readable language instead of octal or binary machine language, then translated to machine language by the assembler. Because of the assembly process, though, operation became more complex. First, the assembler itself had to be loaded from cards. The assembler would then run, reading the programmer's **source deck** of cards as its data and producing a translated **object deck** as output. The object deck was then loaded into the computer, and the program could finally be executed.

In some cases, object decks had to be combined with support subroutines before execution. These decks were merged in an additional step by a **linker** program. Assemblers and linkers were the first system programs.

To assist in the management of this process, cards containing system programs were color-coded. The assembler might be red, the linker yellow, and so on. A special loader card was placed between decks to prepare the system to accept the next program.

-

A typical sequence of steps involved in processing a program using a card-based system is listed in Figure 2-5. This process appears cumbersome to a modern observer, but it represented a great advancement over entering programs manually in machine language.

LOAD THE LOADER (white deck)
LOAD ASSEMBLER (red deck)
READ PROGRAM (your choice of color)
READ PROGRAM AGAIN (pass two)
PUNCH OBJECT DECK (how about pink?)
LOAD LOADER (white deck again)
LOAD LINKER (yellow deck)
READ OBJECT DECK (pink deck)
READ LIBRARY ROUTINES (mixed colors)
PUNCH LOADABLE PROGRAM (blue if you like)
LOAD LOADER (white deck, one more time)
LOAD EXECUTABLE PROGRAM DECK (blue deck)
RUN PROGRAM (finally!)

Figure 2-5: Sequence of Operation for a Card Based System

Tapes and Operators

Although the use of punched cards improved the effectiveness of the programming process, computer use continued to be scheduled on an open shop basis. Clearly, these early computers were not being used efficiently; this was a serious problem. Computers were very expensive in the 1950s, and there were very few available. To make more effective use of a very valuable resource, better solutions had to be found.

One improvement became available with the advent of magnetic tape storage. Using machines separate from the computer, programs and data could be transferred from cards to tape. The tapes were then mounted on the computer itself to supply input and receive output. One tape could hold the same amount of data as thousands of cards, and could be processed much faster. This card-to-tape transfer process eventually led to the idea of using separate, simpler computers attached directly to the main computer to manage I/O devices.

A second improvement was to hire trained operators to interact directly with the computer. These operators were supplied in advance with the cards or tapes to be used by a series of programmers. They became adept at loading cards, mounting tapes, and starting the execution of programs. When a program completed, they could rapidly collect its output and start the next one. If anything unusual had to be done, though, the operator required instructions. Frequently it

was necessary for the operator to call the programmer when his job was ready to be run, so the programmer could be present to deal with special situations [Auslander et al. 1981].

The use of operators sped things up, but not enough. There was still plenty of idle time, and system managers began to think about automating the operator. If the cards required for a series of jobs could be "stacked" in the card reader or on a tape, as was becoming common, perhaps a "supervisory program" could be devised that would wait in a corner of the memory while one job ran, and then proceed to load the next one. Such a method should improve efficiency, at least as long as programs behaved properly and no special problems arose. This was the principle on which the earliest operating systems were based.

2.3 SUPERVISORS AND I/O SYSTEMS

First Ideas

During the early 1950s, dozens of computers were delivered to the offices of major corporations. Chief among these systems were the UNIVAC I and II and the IBM 701 and 704. Although large and very expensive, these computers were becoming cost-effective for the largest organizations.

As the number of computer users grew, it became clear that they could benefit by joining together and sharing their experiences. User groups were organized, and conferences were held to discuss issues of concern to programmers and system managers. High on the list of topics for discussion by these groups was the problem of idle time between jobs. Another problem of concern was the repetitive, difficult work every programmer had to do to write routines that effectively controlled I/O devices.

According to tradition [Steel 1964], the first serious discussion of the idea of writing an operating system, or supervisory program, to address such problems took place in the hotel room of computer pioneer Herb Grosch at the 1953 Eastern Joint Computer Conference. Present at this informal discussion were programmers from major companies that used IBM computers; one company represented was General Motors.

Shortly after this time, a rudimentary operating system was designed by General Motors for the IBM 701. The program, called the **Input/Output System**, is generally considered to be the first operating system. This small set of code lived in a small area of the 701's memory, providing a common set of procedures to be used for access to input/output devices. In addition, each program (if properly written) branched to this code when finished, and it accepted and loaded the next program [Ryckman 1983].

Spurred by the GM effort, a number of supervisors were developed by users of the 701's successor, the 704. In fact, GM combined with another user, North American Aviation, to produce one of the first 704 systems.

A Common Supervisor

These early individual efforts led to interest in an improved common supervisory system that would enable easier sharing of programs. An IBM user's group, **SHARE**, had already been formed to promote such sharing. In an unusual cooperative effort, members of SHARE developed specifications for a new operating system, and IBM implemented it. The **SHARE Operating System (SOS)** [Shell et al. 1959] provided supervisory control and buffered I/O management for IBM's next computer, the 709. It also provided support for programming in symbolic assembly language.

While the SHARE effort progressed, a separate group of users developed a different supervisor for the 709, based on the GM/NAA OS for the 704. This effort incorporated a translator for IBM's new FORTRAN language. The result was the **FORTRAN Monitor System (FMS)**, the first operating system to support programming in a high-level language.

The new supervisory software improved operating speed and automated much of the operator's job, but did not make the operator obsolete. Cards and tapes had to be loaded and unloaded frequently. Moreover, operating systems included little or no ability to recover from program errors or other unexpected situations, so the operator acted as system monitor.

Real-Time and Transaction Processing

Two additional events during this period initiated the development of classes of special-purpose operating systems, which have since gained major importance. The **SAGE** software system, developed by IBM for the AN/FSQ7 military computer, was used for direct monitoring of military equipment [Everett et al. 1957]. This was the first **real-time control system**, in which the computer was required to respond to time constraints imposed by external events. A few years later, IBM developed the **SABRE** airline reservation system for American Airlines, the earliest **transaction processing system**, which allowed processing of simple jobs from a large number of remote stations [Jarema & Sussenguth 1981].

2.4 TAPE AND DISK OPERATING SYSTEMS

The introduction of punched cards made the first operating systems possible. The appearance of magnetic tape increased the efficiency of such systems by allowing card input and output to be temporarily stored on tape. Gradually, tapes began to be used for more permanent storage of commonly-used data, such as compilers and other system programs. Important tapes were kept mounted at all times, while others could be changed according to the needs of the program. This strategy provided a rudimentary file storage, and led to a series of **tape operating systems**. Typical examples included TOS/360 for the first S/360 computers, and TOS for the RCA Spectra 70.

The potential for permanent data storage mushroomed with the development of magnetic disks. For the first time, it was possible to keep large amounts of data and programs in permanent storage, and to access this data in any desired order. Tape operating systems gave way to **disk**

operating systems, widely known as **DOS**. Management of this data became an important new challenge for the operating system.

The first disk operating systems treated disk files as substitute tapes. For example, files commonly had to be "rewound" before use, even though this was physically meaningless. Eventually, the greater power of disk systems for random access to data was exploited more effectively.

Many new computer manufacturers entered the field during this period with computers featuring larger memories, more instructions, and faster execution. Most of these computers were equipped with disk operating systems. Unlike their predecessors, these newer and much more complex operating systems were developed and distributed by the manufacturers themselves. Significant operating systems that had their birth during this period included ADMIRAL for the Honeywell 1800, EXEC I for the UNIVAC 1107, SCOPE for the Control Data 6000, and the first Master Control Program for the Burroughs 5000. The direct successor to the earlier IBM systems was IBSYS, developed by IBM for the 709 and its much-improved but program-compatible descendant, the 7090.

Early DOS systems greatly reduced the effort involved in operating and using a computer. Although the disks used by these systems had relatively small capacities by today's standards, the speed of loading both system programs and user programs, and the ease of use, were significant improvements over the earlier card and tape systems. However, cards and tapes were still needed on these systems as a means of original input of information.

An important part of the early disk systems was a **resident loader**, which would load system and user programs into memory, prepare them for execution, and pass control to them. Upon termination, a properly behaved program would pass control to another routine within the operating system, which would prepare to accept the next program.

Each new job was expected to begin with a set of instructions to advise the operating system on the task to be done and tapes or other resources that would be needed. These instructions were written in a special language that came to be known as a **job control language (JCL).** When preparing to accept a new job, the OS would first load a special system routine to read and interpret this JCL and take the necessary actions.

Also included in these DOS systems was an improved **input/output control system (IOCS)**, which provided support for the devices attached to each computer. Device support generally took the form of a set of subroutines. For some critical devices such as the disk or typewriter console, the necessary support routines remained in memory at all times. For other devices, the necessary routines could be merged with application programs as needed by the system linker, or loaded directly by the loader.

Figure 2-6 illustrates the sequence of steps for preparing and executing a program using a simple disk operating system. Compare this sequence to that of card-based systems (Figure 2-5). The first step in execution of a new program is reading and analyzing the instructions given in the job control language. Guided by these instructions, the OS next loads and executes the assembler, which reads the input program and produces an object program that is temporarily

stored on the disk. Again following instructions from the JCL, the linker is invoked to combine the object program with appropriate library subroutines and produce a load module ready for execution. Finally, the load module is loaded into the main memory, and the application program is executed.

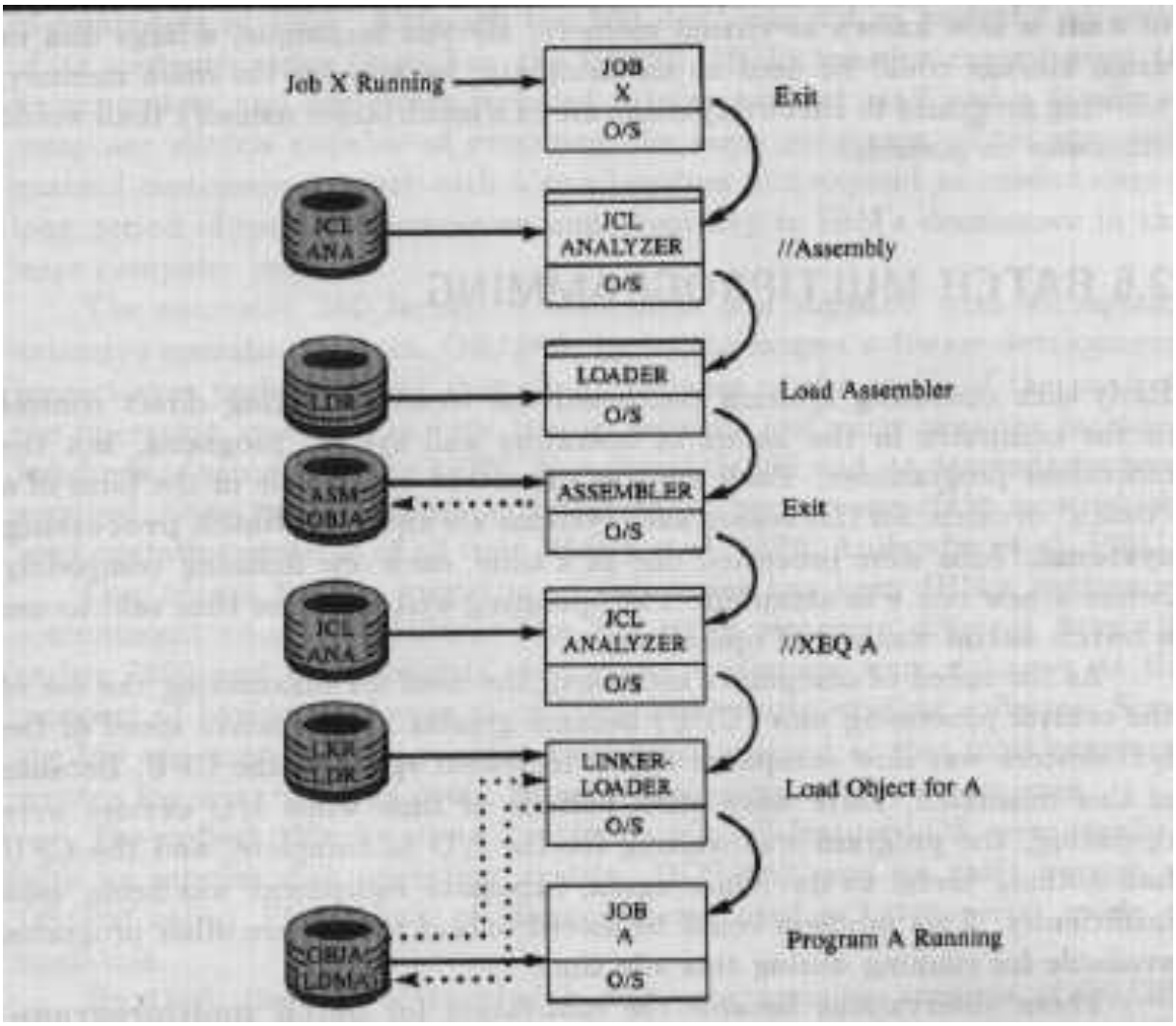

Figure 2-6: Operation of a Disk Operating System

2.5 ATLAS IS BORN

Most of the computers that supported the early DOS systems had very similar traditional designs (a notable exception was the stack-oriented architecture of the Burroughs computers). At Manchester University in England, a very different type of computer was being developed that would have a profound effect on the evolution of operating systems. The **Atlas**, developed jointly by Manchester University and Ferranti Ltd., became operational around 1961 [Howarth et al. 1961].

Atlas was not a large or fast computer compared to its contemporaries, but its architectural innovations were far ahead of its time. Atlas was the first computer designed to support an operating system. This viewpoint led to significant innovations. Atlas introduced the concept of **interrupts**, making it possible for a program in normal operation to be interrupted by an external event, such as completion of an I/O activity. This concept greatly aided the efficient management of I/O concurrent with normal processing. In addition, Atlas provided a special instruction called an **extracode**, which caused a trap to invoke special routines written in software. Although used originally for simple library functions such as math procedures, extracode was the forerunner of the system call instruction used by later computers to request services from the operating system-an instruction that plays a critical role in making multiuser systems possible.

The innovation for which Atlas is best known, however, is its **one-level store**. This novel hardware/software system included all the basic elements of what is now known as virtual memory. By this technique, a large disk or drum storage could be used as an automatic backup for the main memory, allowing programs to effectively make use of a much larger memory than would otherwise be possible.

2.6 BATCH MULTIPROGRAMMING

Early disk operating systems continued the trend of placing direct control of the computer in the hands of operators and system programs, not the individual programmer. Each job was submitted as a whole in the form of a "batch" of cards; for this reason such systems are known as **batch processing systems**. Jobs were processed one at a time, each one finishing completely before a new one was accepted. The operating systems were thus said to use a **batch serial** method of operation.

As the speed of computers increased, the need for maximizing the use of the central processing unit (CPU) became greater. The relative speed of the I/O devices was slow compared to the increased speed of the CPU. Because of this mismatch, there were often periods of time when I/O devices were operating, the program was waiting for the I/O to complete, and the CPU had nothing useful to do. Once again, expensive equipment was being used inefficiently. This problem could be solved only if there were other programs available for running during this idle time.

These observations became the motivation for **batch multiprogramming**, the shared use of the CPU, memory, I/O devices, and other resources of a computer system by more than one program executing in an interleaved manner. With this technique, several user programs were loaded into memory at the same time, if sufficient memory was available. When a running program had to wait for I/O completion or some other event, the CPU could work on one of the other programs.

A pioneer multiprogramming operating system was the **Master Control Program (MCP)** for the Burroughs 5000 [Oliphint 1964]. Building on the Atlas experience, this system used virtual memory techniques to provide sufficient storage for several programs, which could be chosen alternately for execution. Programs were assigned priorities-that is, numerical values

-

representing the relative urgency of each program. These priorities influenced the choice of programs to be run. The Burroughs system was unique in many other respects. In particular, both hardware and software were designed to support programs written in high-level languages; all user's programs were expected to be written in either ALGOL or COBOL, and translated by special system compilers.

The most significant development in the evolution of batch multiprogramming, however, was the announcement of the IBM System/360 family of computers in 1964. Although the 360 itself was not as powerful as some of its contemporaries (including the B5000), IBM's massive commitment to its computers and customers included a large support staff and a family of computer models capable of executing the same programs. This approach enabled customers to start with a small system and expand as needed over a long period of time. It became an important key to IBM's dominance in the large computer industry.

The extensive 360 family of computers was supplied with an equally extensive operating system, **OS/360**, by far the largest software development project ever undertaken at that time. Because of the scope of the project, the operating system was very late in delivery and early versions included hundreds of errors [Brooks 1975]. However, OS/360 and its descendants have survived, along with the S/360 architecture, to become one of the most widely used operating systems of all time [Mealy et al. 1966; Auslander et al. 1981].

One reason for the longevity of this series has been IBM's continuing commitment to compatibility. The 360 series was very different from the earlier 7090 and other models, and many customers were unhappy at the prospect of having to change procedures and rewrite existing software. Since the 360 appeared, all new versions have been designed so that most programs written for older versions could be used unchanged on the newer ones.

The earliest 360s, awaiting completion of a full-featured OS, were supplied with an interim disk operating system, DOS/360, and an early version of OS/360 called PCP. Both of these systems used a batch serial mode of operation.

By 1965, the first of two batch multiprogramming versions of OS/360 appeared. **OS/MFT**, for smaller S/360 models, and the more powerful **OS/MVT** became the dominant operating systems for the 360 architecture.

One feature of the OS/360 series was the introduction of a large and powerful job-control language, known simply as **JCL**. Much could be done with IBM's JCL; however, its use was complex even for simple jobs, and many installations had to employ full-time JCL experts to aid in writing and debugging the JCL for each application.

In order to reduce delays caused by slow input and output devices, batch multiprogramming systems introduced the concept of **spooling**. The term was both a description and an acronym; it stood for Simultaneous Peripheral Operation On-Line. One special program was given the responsibility of reading input jobs from cards or tapes and storing complete copies of them on a disk. This program tried to read jobs as fast as they became available. The operating system then obtained its input rapidly from the fast disk instead of the slower cards and tapes. A similar technique was introduced for output. Information to be printed was initially

placed on disk; a spooling program then copied data from the disk to printers or other output devices.

Figure 2-7 presents an overview of the flow of activity for a typical job using a batch multiprogramming OS (in this case, OS/MVT). A job for this system was made up of a series of job steps. Each step specified the program to be run and the data files and/or devices to be used for the job step. Cards were read and temporarily stored in a job queue, where they remained until selected for execution by the job management portion of MVT. Hence, batch processing was central to the operation of these early 360 systems.

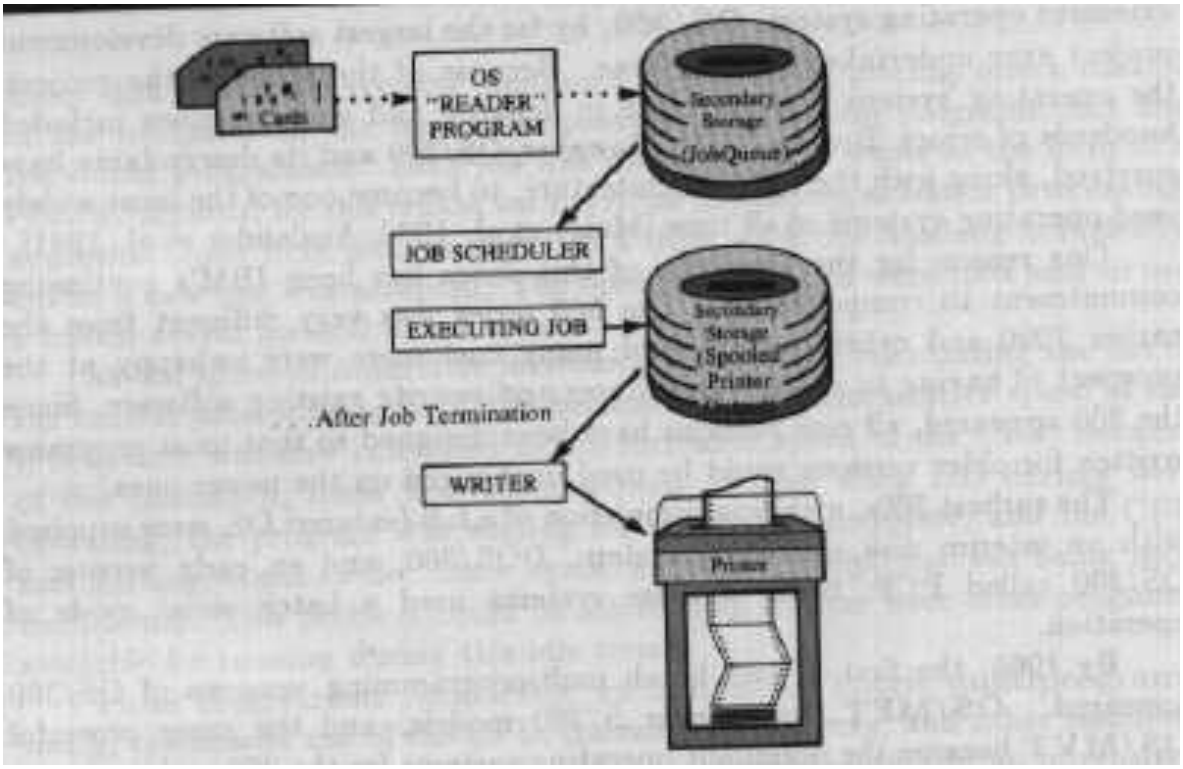

Figure 2-7: Overview of a Batch Multiprogramming System

2.7 TIMESHARING SYSTEMS

Although powerful batch operating systems had greatly improved the rate at which work could be done by each computer, users of the new computer systems were not always pleased. In the early days of computing, each programmer had direct control of the computer while his program was running. He was able to observe the progress of the job in detail. If things went wrong, bugs could be located immediately. In many cases, the program could be fixed and allowed to continue its run. With batch processing, this type of interaction was no longer possible.

A more severe problem was the long period of time that elapsed between submission of a job to the operator and results returned to the computer user. This delay was often due to both administrative and technical problems; it was typically several hours and could be as long as several days. Since most jobs had to be run several times before all the errors were corrected, these delays quickly multiplied. The situation was frustrating both to experienced programmers and to new users who wanted to take advantage of the computer for data processing or scientific applications.

Dissatisfaction with the problems of batch computing led many system designers to search for new modes of operation that would restore the advantages of direct interaction with the computer and provide much more immediate results.

The development of disk file systems often made it possible to reduce drastically the amount of direct input needed for a typical job. Large programs and sets of data could initially be prepared and stored in disk files, which could then be edited if necessary by special programs. All system programs now were stored in disk files. Often the information that a programmer had to provide to carry out a job consisted mainly of a few lines of JCL. In such cases, it seemed possible that programmers could type this information on a typewriter attached directly to the computer, and receive at least limited results on the same typewriter. This could relieve many of the frustrations of batch operation, replacing it with a new kind of **interactive computing**.

This attractive idea, however, was not without problems. If interactive computing was carried out by assigning the computer to one user at a time, the computer would be used as inefficiently as in the old open shop arrangements. However, early experience with batch multiprogramming showed that computers could hold several jobs in memory, and divide their time among the various jobs. If the multiprogramming concept could be used to serve a number of programmers at typewriters at the same time, the computer could still be kept busy. In addition, many more programmers might be able to use the computer in a given period of time.

The remaining difficulty was the rate at which the computer would have to switch from job to job. In batch multiprogramming, a switch was made only when the current job reached a stopping point. Often a single job would run for minutes or hours. To support multiple interactive users in a reasonable way, the computer had to be forced to switch from job to job at a much higher rate, a task accomplished by assigning a short time limit, typically one or two seconds, to each job. This limit was called a **time slice** or **time quantum**. A timer was set to interrupt the job automatically when the time was up. Once interrupted, each job had to wait for the others to have a turn before finally receiving more service.

This type of multiuser interactive computing system became known as a **timesharing system** (see Figure 2-8). Timesharing operation was the focus of much of the operating system development in the 1960s and 1970s.

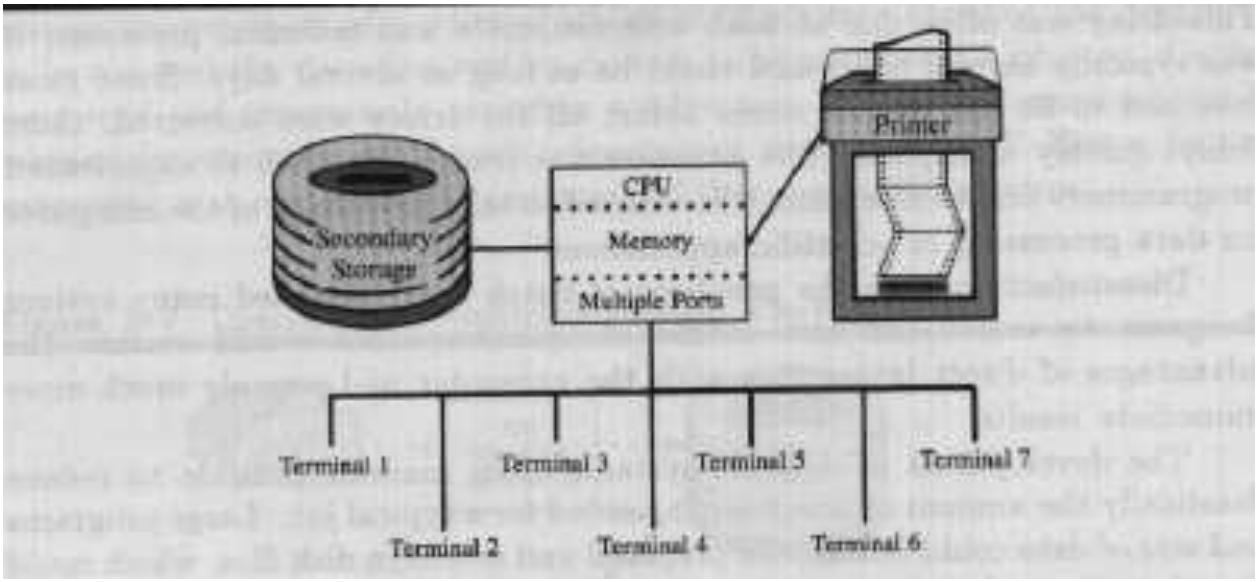

Figure 2-8: Organization of a Timesharing System

A great many timesharing systems were developed in the early and mid-1960s. Perhaps the first such system was designed by Bolt, Beranek and Newman for the Digital Equipment PDP-1 computer. Another early system that had a significant influence was the **Compatible Timesharing System (CTSS)** developed on a modified IBM 7094 at the Massachusetts Institute of Technology [Crisman et al. 1964]. Originally completed in 1961, CTSS established the effectiveness of the timesharing concept, and also exposed some of its potential problems. CTSS was heavily favored by a large community of users until the mid-1970s.

A noteworthy timesharing system of the mid-1960s was developed by Kemeny and Kurtz [1968] in cooperation with General Electric at Dartmouth, a small liberal arts college in Hanover, New Hampshire. The **Dartmouth Timesharing System (DTSS)** was born of its authors' conviction that computer use should form a part of the education of all liberal arts students. As a further step in making the computer easier to use for nonprofessionals, Kemeny and Kurtz developed the BASIC programming language for use on DTSS. The history of BASIC and DTSS is recounted by the creators in an interesting book [Kemeny & Kurtz 1985]. (They report that DTSS was originally called "Phase II," but that name was discarded because it sounded too much like a bath soap.)

In short order, experimental timesharing systems began appearing for a variety of computer systems. Manufacturers were a little slow to enter the field, but some pursued it vigorously. Digital Equipment Corporation introduced the TOPS-10 timesharing system for its medium-sized PDP-10 computer. IBM produced a virtual memory version of the 360-the 360/67-and began development of TSS/360. Although tremendous effort was expended on this system, which was planned as a likely successor to OS/360, it was never very popular.

Experience with CTSS and similar systems soon showed that timesharing was workable but could lead to serious difficulties. A great deal more time was spent by the operating system

in switching between programs. With a large number of users, this switching overhead could leave little time for useful work. Computers with better performance, and carefully designed supervisors, were needed. In addition, the simultaneous use of files and other resources created a need to protect users from each other. An error in one program could damage data of many users and bring the system to a halt. The dangers of deliberate misuse of one user's programs and data by other users was also beginning to be realized.

These problems were studied closely by the members of the computer research group at MIT, Project MAC. They determined that both a new operating system design and an improved computer design were required. In 1964 a project was launched jointly by MIT, AT&T/Bell Laboratories, and General Electric to develop a next-generation successor to CTSS. The new system was envisioned as a "computing utility," which would make the power of computers as accessible to a large community as other utilities, such as electric power or telephone service. The result of this effort was the **MULTICS** operating system.

MULTICS was one of the largest and most ambitious operating systems ever attempted. It explored a wide range of new concepts, including segmented virtual memory, linking and loading segments on demand, a novel method of resource protection, a hierarchical file system, device independence, I/O redirection, and a powerful user interface [Brown et al. 1984]. Along with the software development for MULTICS, extensive hardware development was carried out in the form of modifications to the GE 635 computer. These modifications were designed to support new ideas in virtual memory management and in protecting access to resources. The resulting computer was called the GE 645. MULTICS became operational as the 1960s drew to a close.

MULTICS is known especially for its "rings of protection" scheme, its segmentation system that treated files and main memory uniformly, and for being the first major operating system written almost entirely in a high-level language (a variant of PL/I). MULTICS is described in detail in a book by Organick [1972]. Although it did not fully meet its original goals, the system made significant contributions to computer and operating system technology. MULTICS was later acquired by Honeywell, along with GE's line of computers, and continued in use through much of the 1980s.

2.8 ABSTRACT AND VIRTUAL MACHINES

In the late 1960s, an operating system was developed as a research project by a team led by Edsgar Dijkstra at the Technische Hoogeschool (Technological University) in Eindhoven, Holland. The operating system was named T.H.E. [Dijkstra 1968; McKeag et al. 1976] for the Dutch initials of the University. It was a modest batch multiprogramming OS with no user file storage, and requiring all programs to be processed by the system's ALGOL compiler. **T.H.E.** would have a profound effect on the structure of all operating systems to follow.

The principal contributions of T.H.E. lay in its treatment of interacting processes and in its hierarchical structure. Because of competition for the use of resources, the presence of multiple processes in simultaneous execution in a multiprogrammed system can lead to a variety of problems. T.H.E. built the foundation for solving these problems, and introduced a number of

ideas that are still widely used today. Among them are some effective approaches to the problem of deadlock and the synchronization mechanism of semaphores.

An equally important contribution was made by T.H.E. in the form of a method for structuring the OS itself. The operating system had a hierarchical structure. Its components formed a series of layers, each one interacting only with the layers above and below. The first layer, for example, managed timers, interrupts, and process scheduling, while the second layer provided virtual memory. Each layer could be said to form an **abstract machine**---that is, an apparent extension of the real machine. All components of the OS above the second layer, as well as application programs, could treat process management and virtual memory as "built-in" features of the machine itself. Moreover, once the lowest layers were correctly implemented, they could be relied on in developing and testing higher layers. Problems at these outer layers could not affect the lower-level software.

A number of hierarchically structured operating systems followed soon after T.H.E. One that made a number of important contributions was **TENEX**, developed in the early 1970s at Bolt, Beranek and Newman for the PDP-10 [Bobrow et al. 1972]. Following the tradition of earlier BBN systems, TENEX was a timesharing OS, supporting a well-designed user interface and file system. Its ideas in these two areas directly influenced later Digital Equipment operating systems for the PDP-10 and PDP-11.

TENEX provided an abstract machine structure including the file system, extended machine instructions, multiple processes for each user, and virtual memory (although the PDP-10 required hardware extensions to support this feature). Its designers called this structure, with some justification, a "virtual machine," but that term has now been focused more narrowly on a special type of abstract machine that was being quietly developed in the laboratories of IBM.

Beginning in the mid-1960s, a series of experimental systems were developed at IBM's research center in Cambridge, Massachusetts. These systems drew on the experience of CTSS at nearby MIT to create a timesharing system most useful for internal development work by IBM engineers. Researchers were beginning to explore virtual memory concepts, and IBM had produced an experimental virtual memory system, the M44/44X, at another research center in New York. The new system was designed to incorporate both time-sharing and virtual memory.

The OS developed at Cambridge was called **CP/CMS** to reflect its two parts. CP stood for Control Program, while CMS could be interpreted as either Cambridge Monitor System or Conversational Monitor System. Its objective was to support all of the research and development activities of IBM engineers at Cambridge; one of these activities was operating systems research. To allow OS work to be timeshared with other activities required a radical view of the services provided by the abstract machine to each user. It would have to be a **virtual machine** in the truest sense, providing apparent access to all of the machine features required by an operating system. Just as virtual memory could provide each user with the illusion of a complete private virtual address space, a virtual machine must extend this illusion to a private CPU and possibly I/O devices as well.

The first CP/CMS system at Cambridge ran on a modified S/360 model 40, and was designated CP/40. Shortly afterwards, IBM developed the model 67 with virtual memory

-

support, and CP/40 was succeeded by CP/67. When the System 370, successor to the 360 family, was introduced in the early 1970s, the experimental OS at Cambridge was transferred to the new environment and christened VM/370, or Virtual Machine for the 370. At this time it became an official product of IBM.

2.9 MINICOMPUTER OPERATING SYSTEMS

Ever since the early IBM and UNIVAC systems entered the commercial world in the 1950s, interest blossomed in developing computers that were smaller and cheaper, and thus more accessible to many potential users. As early as 1955 several "small-scale" computers were on the market, moderately priced at under $50,000 [Koudela 1973]. These computers, which included the Burroughs E-101, Bendix G-15, and Librascope LGP-30, were large machines using bulky, unreliable vacuum tubes. They were extremely slow, with several milliseconds required for an arithmetic operation. Their memories were limited to a few thousand words at most, and they had very limited instruction sets. They were programmed in machine language, with no operating systems of any type. Still, some useful software libraries were developed for the small-scale machines, and the LGP-30 was useful enough to support early work leading to the development of BASIC and DTSS at Dartmouth.

The early 1960s saw an expansion of the small-computer field to include machines like the CDC-160 and the IBM-1620. These were the first small computers that took advantage of the new solid-state technology to greatly reduce their size and increase reliability. Both data and instructions on the CDC-160 were limited to 12 bits in length. To address a reasonable amount of memory with such small instructions, relative and indirect addressing were introduced. A rich set of addressing modes has been a characteristic of minicomputers since this time.

Digital Equipment Corporation (DEC), a new player in the field in the early 1960s, introduced the PDP-1 (for Programmed Data Processor), followed in short order by several more models in the PDP series. The 12-bit PDP-8, priced under $18,000, opened up the minicomputer field in earnest and became one of the best-selling computers of all time. The PDP-8 introduced a number of features important for real-time applications, including interrupts, a clock, and direct memory access for high-speed I/O devices. These features gave it the ability to successfully control real-time devices. Laboratory and process control became an important application area for this and many later minicomputers.

Early minicomputers were supplied with limited system software, and there was at first little concept of an operating system. Programming style followed an evolution quite similar to that of the larger computers, with early interaction performed using front panel switches and display lights. Again, the most common medium that emerged for program and data storage was punched paper, but the minicomputer environment favored long rolls of punched-paper tape. Although less convenient and harder to handle than cards, paper tape could be read and punched with much less expensive equipment.

Soon after the appearance of the PDP-8, a new round of minicomputer systems was introduced including the DDP-116 from Computer Control Company (later Honeywell), The DATA-620 from Data Machines, Inc. (later Varian), and the IBM 1800 and 1130. These were 16-bit architectures featuring such innovations as multiple accumulators, vectored priority interrupts, and index registers. Some of these machines were outfitted with magnetic tapes and disks.

The beginnings of a minicomputer operating system appeared with the DDP-116 "Input/Output Selector," paralleling the early Input/Output System for the IBM 701. Soon after, IBM introduced a disk operating system for the 1800. Many of these early minicomputer systems had names like "keyboard monitor" and "real-time monitor." They provided an interactive user interface for a single user, and ran one program at a time. Their principal roles were seen as accepting simple commands from a teletype console, loading and running programs from a disk or tape file system, and perhaps monitoring the operation of real-time laboratory devices.

The popular PDP-8 minicomputer was originally supplied only with paper tape-based software. However, the development by DEC of a low-cost, bidirectional tape system (DECtape) made limited monitors and operating systems possible. Several OSs then appeared for the PDP-8, including OS/8, a single-user monitor, and TSS/8, which provided a limited form of timesharing. Digital's large-scale PDP-10 had its TOPS-10 OS, and the medium-sized PDP-9 and PDP-15 were provided with both single-user monitors and an experimental real-time OS, RSX-15 [Krejci 1971].

In 1970 DEC introduced the PDP-11 minicomputer series. Adopting the strategy that IBM found so successful for its System/360, DEC planned the PDP-11 as a family of compatible models with varying size and performance. This strategy was equally successful for the PDP-11 series. By the early 1970s, no fewer than three important operating systems were available for the PDP-11: a clean and simple single-user OS (RT-11), a timesharing system (RSTS), and a real-time executive (RSX-11).

RT-11 was based primarily on OS-8 but used important ideas from TOPS-10 and TENEX, especially in its command interface and file system. Designed to serve a single user, **RT-11** normally executed only one program at a time. However, a limited form of multiprogramming was available in the **foreground/background** version of RT-11. This OS allowed one process currently not communicating with the terminal, such as a long compilation, to continue in the "background" while the user interacted with a different "foreground" program.

Although the "RT" in RT-11 stands for "real time," the true real-time executive for the PDP-11 was **RSX-11**, derived from RSX-15. RSX-11 was the most advanced of the PDP-11 operating systems, supporting a powerful command language and file system, memory management, and multiprogramming of a number of distinct programs. RSX-11 became the principal ancestor of the powerful operating system that would appear with the next generation of DEC computers.

The early to mid 1970s was the heyday of minicomputers. The PDP-11 had many competitors; perhaps the best known was the Data General Nova with its RDOS operating system. Other players included Xerox, Interdata, Hewlett-Packard, Datapoint, Honeywell,

PR1ME, etc. IBM also introduced a line of "midrange" systems, especially the AS/400 with its operating system OS/400, but IBM did not achieve the dominance they enjoyed in the large computer arena.

2.10 THE ORIGINS OF UNIX

Around 1970, Bell Labs began to wind down its involvement in the MULTICS project. Two researchers who had been active on that project, Ken Thompson and Dennis Ritchie, were ready to move on to other projects in their software research group. However, they were becoming very dissatisfied with the capabilities of the operating systems available for the minicomputers in their lab. Convinced that there was a better way, Thompson set about to design a new single-user OS more suited to his own research needs. Ritchie soon joined him in this effort. Their design drew upon concepts from their MULTICS experience. The result was an operating system for the DEC PDP-7, which the designers christened **UNIX** [Ritchie & Thompson 1974; Ritchie 1980].

The environment in which UNIX was developed was unique. The Bell engineers were a research group, with wide freedom to pursue projects of their own choosing. UNIX was not an official company project; indeed, AT&T at this time was not permitted to develop computer-related products. Thompson and Ritchie had no requirements to adhere to except their own needs.

The operating system that emerged from this effort was quite different from most that preceded it. Running on the limited PDP-7 hardware, UNIX provided its creators with the effective working environment that MULTICS had so far failed to deliver. It became widely used within Bell Laboratories, and was reimplemented on the PDP-11 when that newer system became available.

Key aspects of the UNIX environment were a simple yet powerful file system with a hierarchical system of directories, and a novel command interface called the shell. The file system treated I/O devices as special cases of files, and the shell made it possible to direct the output produced by any command to any selected file or device. Similarly, input could be taken from any source in response to a shell command.

Although the earliest UNIX systems supported only a single user, the work of one user could be carried out by multiple concurrent processes. The system and the shell supported an effective mechanism for communication between processes, in which an output stream produced by one process could become the input stream of another.

These features made possible some new styles of operation. For example, one process might manage an interactive editing session while another process printed a file and yet another carried out a time-consuming sort or compilation. A selective directory listing might become input to a separate command, identifying files to be printed or processed in some particular way. Although many of the UNIX concepts had been present in predecessors such as MULTICS,

-

CTSS and DTSS, UNIX was an improvement on the previous ideas and tied them together in simple and effective ways.

The first versions of UNIX were written in assembly language. The designers were accustomed to the more sophisticated PL/I language variant used to implement MULTICS. However, PL/I was unavailable on the PDP computers, and the available languages were unsatisfactory for system programming. Ritchie addressed this problem by creating a new language, simply called "C." C was based on an earlier experimental language "B," which in turn was derived from BCPL. A C compiler for the PDP-11 was developed, and eventually almost all of UNIX was rewritten in C.

UNIX was widely used inside Bell Laboratories, but it did not become an official product for many years. As the parent company of Bell, AT&T was prevented by antitrust regulations from selling software. Instead, UNIX became available for a nominal cost to universities, and eventually to independent developers who were able to convert it to a commercial product. During this period the UNIX system, complete with source code, became familiar to a large community of students and researchers. Several enhanced versions were developed, especially at the University of California at Berkeley. The widespread use of UNIX at universities created a demand for commercial versions. UNIX was one of the first operating systems to be distributed separately from the computer system on which it ran; it was eventually implemented on many different computers, becoming one of the first portable operating systems.

2.11 OPERATING SYSTEMS FOR MICROS

Solid-state electronics was the technology that drove the steady improvement and miniaturization of computer components. Starting with the invention of transistors to replace bulky and unreliable vacuum tubes, solid-state advances allowed increasingly complex circuits to be created on small, inexpensive integrated circuit microprocessors, or "chips." The economics of this technology were remarkable: although steady improvements allowed more and more complexity on each chip, the cost of producing the chip did not substantially increase.

This evolution was spurred in the late 1960s by the rise of the pocket calculator. These small devices had many of the elements of complete computers. They could perform many complex calculations, store data, and even execute simple programs. With price tags of a few hundred dollars, they were enormously popular, and there was great competition to further increase their capabilities and decrease their costs by refining integrated circuit technology.

As calculators became increasingly sophisticated, the complexity of the special-purpose chips used to implement them was becoming unmanageable. The first developer to see the possible advantages of developing a more systematic, general-purpose chip for calculators was Intel Corporation. In 1971 Intel announced the 4004, implementing a complete 4-bit CPU on a few circuit chips. Although this first CPU had very limited capabilities, it was adequate for use in calculators; researchers envisioned a few other applications as well. The 4004 was followed shortly by a more powerful 8-bit version, the 8008 [Noyce & Hoff 1981].

-

After two years of experience with the 8008, Intel released the much-improved 8080. This chip became very widely adopted as the potential of microprocessors became apparent to more and more users. Aided by the heavy demand, the cost of the 8080 dropped to a few dollars per chip. Its architecture has been a major influence on later Intel CPUs.

Other manufacturers soon followed with devices such as Zilog's Z-80, Motorola's 6800, and MOS Technology's 6502. The age of the **microprocessor**, or "CPU-on-a-chip," had arrived.

Intel and other microprocessor developers had little experience with actual computers, and did not consider themselves in the computer business. Others, though, were quick to see the potential of developing complete small-computer systems based on microprocessor chips. If the great cost reductions in CPUs could be matched by similar reductions for memories and other system components, it would be possible to develop **microcomputers** so small and inexpensive that they could be used and afforded by individuals.

As recounted by Freiberger and Swaine [1984], the earliest microcomputers were built by hobbyists. Their experiments led to commercial ventures by pioneer companies like MITS and Imsai. Although many of the early microcomputer companies did not survive, a few found the right combination of good design and sound business decisions. One well-known survivor was Apple, a company originating from the work of two unconventional young computer hobbyists. Apple made use of Motorola 6502 CPUs in novel, self-contained personal computers featuring effective color graphics, sound, and game-playing capabilities. The Apple II became extremely popular.

Computer development had also been undertaken by a few more established companies, notably Tandy (Radio Shack) and Heath. These companies had the advantage of being secure enough to deal with occasional setbacks. As the field began to mature, the future of the microcomputer was assured by the announcement of a personal computer by the biggest survivor of all: IBM.

Like the mainframes and minicomputers before them, the very early microcomputers were programmed with front-panel lights and switches. These computers were equipped with a few thousand bytes of memory and primitive forms of input and output. However, the widespread acceptance of micros created a high demand for better memories and improved, low-cost I/O devices. To a large degree, this demand was fulfilled as the 1970s progressed. Main memories of 64K bytes or more became available at costs of a few cents a bit. The cost of video terminals and printers steadily declined, and their capabilities improved. Printers also decreased greatly in size and became easier to maintain. Affordable storage devices were developed, based first on simple audio cassette tapes and then on the very popular floppy disks.

The new microcomputers thus formed the center of the third evolutionary spiral, following mainframes and minicomputers. Each evolution followed a somewhat similar path, and each moved much more rapidly than the one before. The characteristics of typical systems in each of these families (at similar stages in their evolution) are compared in Figure 2-9.

	MAINFRAMES	**MINIS**	**MICROS (c. 1980)**
Cost (thousands of dollars)	500-5000	5-25	1-5
Memory size (K bytes)	1000-10000	50-500	50-200
Instructions per second (thousands)	1000	250	100 to 250
Number of simultaneous users	10-200	1-20	1-4
Storage devices	Large disks and tapes	Small disks and tapes	Tape cassettes; floppy disks
I/O devices	Card readers and punches; magnetic tape; high speed printers	Paper tape readers and punches, small terminals and printers, laboratory sensors and controllers	Cassette tapes and floppy disks, small terminals and printers, embedded sensors and controllers

Figure 2-9: Mainframes, Minicomputers, and Microcomputers

Note that the information given in the table for microcomputers describes these systems in their early maturity, and is far out of date today. A typical PC or microcomputer system in 2010 may cost less than $500, include hundreds of megabytes (millions of bytes) of main memory (plus hundreds of gigabytes (billions of bytes) of secondary storage), and execute billions of instructions per second! More recent storage and I/O devices include various newer types of magnetic, optical, and solid-state disks, as well as audio, video, and communication devices.

In spite of the rapid advance in microcomputer hardware, system software was somewhat slow to develop. Simple loaders were sometimes provided in read-only memory, capable of reading data from cassette tapes. Implementations of Dartmouth's already popular BASIC language became common. Microcomputers were seen largely as toys, however, and true large-computer ideas like operating systems seemed beyond the scope of their tiny CPUs.

This situation began to change with the work of Gary Kildall at Intel. Working as a consultant to develop software for the 8008 microprocessor (predecessor to the 8080), Kildall developed a compiler for a simplified version of the programming language PL/I. The new language was called PL/M, or Programming Language for Microprocessors. It was designed for writing microprocessor system software. To assist in this development he also created a rudimentary operating system, called Control Program for Microprocessors, or **CP/M**. This single-user operating system for the 8080 included a simple interactive command interface, basic I/O device management, and a floppy disk-based file system. Intel was not interested in CP/M,

and Kildall obtained the rights to distribute it himself. With his wife he formed a company called Digital Research, and CP/M became its first and most successful product [Freiberger & Swaine 1984].

CP/M was the first microprocessor operating system and remained the dominant one for a number of years. Although some microcomputer companies, such as Apple and Tandy, developed their own proprietary operating systems, most offered versions of CP/M as well.

2.12 LARGE SYSTEMS CARRY ON

The microprocessor began a revolution, but during the 1970s larger computer systems (primarily from IBM) continued to fill an important role in supplying high-performance batch and timesharing services to a wide community of users. The future of minicomputers, though, was somewhat open to question. The DEC PDP-11 had been widely accepted when it first appeared; but its limitations were becoming increasingly evident, especially its 64K byte address space. Digital had been able to dominate its competitors in the minicomputer field, but new microprocessors appearing with 16-bit addresses and data paths competed strongly with the PDP-11's capabilities. DEC responded with the introduction of a "superminicomputer" as successor to the PDP-11, which was called the VAX. The VAX architecture included a very large and powerful set of instructions and the ability to address over four billion bytes. Closely tuned to the VAX design was the **VAX/VMS** operating system. A number of VAX CPU instructions, such as some that directly manage queues, were designed especially to support the needs of VMS. VMS provided an extensive assortment of system services, file management techniques, and associated system programs that placed it second perhaps only to OS/360 in its size, scope and complexity [Kenah & Bate 1984].

The introduction of the VAX cemented Digital's dominance of the (super)minicomputer field. Data General introduced a competing system, the Eclipse, with its Advanced Operating System (AOS). This system still survives but did not achieve a fraction of the popularity of the VAX. Of the many other minicomputer companies of the 1970s, only a handful, notably IBM and HP, have remained significant players.

The VAX architecture became very widely used, but acceptance of VMS as the VAX operating system was not universal. UNIX and its users were rapidly outgrowing the PDP-11, and the VAX was seen as a natural new host for this OS as well. By 1978 UNIX had been implemented on the VAX and greatly enhanced by researchers and students at the University of California at Berkeley.

In the meantime, IBM remained dominant in mainframe computers, although alternative systems from Burroughs, Control Data, and a few other companies maintained a limited following. With superminis like VAX challenging the low-performance systems in its series, IBM placed emphasis on its high-performance systems. The S/370 series had added virtual memory capability to the aging S/360 architecture. VM/370 offered a novel mechanism for exploiting the 370, but VM served a specialized purpose and was not a complete operating system for general use. The mainstream operating systems for IBM hardware evolved instead

from OS/MFT and OS/MVT. The most important successor to these popular systems became known as **OS/MVS** (Multiple Virtual Spaces).

OS/MVS supported an address space of up to 16 million bytes for each process in the system, and was a thorough redesign and updating of MVT for timesharing, based on the earlier timesharing option (TSO) developed for MVT. It continued to support most MVT files and programs. MVS offered effective control for the newer and larger 370 successors [Auslander et al. 1981].

By the mid 1980s, increasing memory demands and declining costs were catching up even to the once forward-looking S/360 architecture. 24-bit addressing was no longer enough; larger address spaces were often necessary. This need led to some new extensions of the mainframe architectures, referred to as Extended System Architecture (ESA), and a revised operating system designated MVS/ESA.

2.13 PERSONAL COMPUTERS AND WORKSTATIONS

In 1970 Xerox Corporation, which had already purchased a small minicomputer company, opened a research facility in California called the Palo Alto Research Center (PARC). Its charter was to carry on pure research at the frontiers of technology; the results of this research had a profound effect on computer hardware and software design.

One of the more striking products of the PARC research was a small computer called the **Alto**. Designed for personal use by professionals, the Alto introduced a new style of communication between the computer and its user. As an alternative to typing commands at a keyboard and receiving results as text displayed on a screen, Alto users could instruct their system by selecting phrases or graphical symbols displayed on the screen by moving a small device called a "mouse." The Alto provided very high performance for its size and sophisticated graphics.

The Alto was expensive, and it was not a microcomputer. Intel's 8008 was still a novelty as the Alto was being developed. Instead, Alto was described as a professional **workstation** that became a model for many microcomputers that followed.

Alto was not a commercial product, but it was sold to a number of government agencies and inspired other Xerox products that appeared during the late 1970s, including the **Star**. The Star featured a large display with an Alto-style graphical user interface. It was designed for information-handling tasks in business offices, and was intended to be easy to use by office workers who might be uncomfortable with other types of computers. The Star, an office automation system, was the heart of Xerox's planned "office of the future."

The Xerox Star was expensive and not very successful. But system developers began thinking about this new style of user interaction. Similar products soon appeared, aiming for similar or enhanced capabilities at lower costs. One developer aiming for high-performance was Sun, which produced a series of expensive, sophisticated workstations aimed at professional

users, such as scientists and engineers. Apple was also paying attention to the Alto and Star, and this led to the introduction of Apple's **Lisa** computer in the early 1980s. The world was not quite ready for Lisa, but it was more than ready for Apple's next version, the **Macintosh**, released in 1984. Taking advantage of the newest microprocessor technology, the Macintosh was by far the lowest-cost computer to follow in the Alto tradition. It provided a complete, self-contained workstation with impressive capabilities for about $2,000. The Apple Macintosh became a great success.

The model of a single-user workstation, responsive to the directions of its user, led to a different conception of operating system design. The OS was long considered the master of the computer system; now it needed to become the servant of its users. This led to the concept of an **open operating system**, one that does not place barriers between application programs and the OS itself. The focus of this type of operating system is on convenience, not control. The open OS provides services as desired, but does not interfere with a program. The OS may be viewed as little more than a collection of subprograms. An open operating system does not get in the way of its users, but it may make programs more vulnerable if errors occur.

2.14 NETWORKS AND DISTRIBUTED SYSTEMS

The idea of direct communication among computers was an attractive goal of researchers for many years. Such communication could make information from one system available quickly to another, without manual transfer or transcription. Various types of data communication equipment had been developed by the 1960s, allowing information to be transmitted long distances in digital form between computers connected by wires, or even temporarily linked by telephone lines.

Long-distance data transmission, though, is very prone to errors, and sophisticated techniques were needed to cope with these problems and provide reliable communication in spite of them. In the early 1970s the Defense Department, through its Advanced Research Projects Agency (DARPA), provided an important stimulus to solving these problems by setting up a dedicated computer network, the ARPANET, to connect computers throughout the country that were performing DARPA research. The ARPANET used a combination of hardware and software techniques to provide reliable and efficient data transfer among its attached computers.

Each computer linked on the ARPANET had its own operating system. Communication was viewed as a special I/O activity. A user wanting to transfer data to or from a remote computer had to understand the names and formats required by that computer, as well as have the proper authorization for the transfer. ARPANET allowed for direct interactive use of remote computers as well, but accounts, log-in procedures, and other activities were the responsibility of individual users.

In an effort to allow users to treat the network more as a unified set of resources, several experimental **network operating systems** were developed. One example was known as the National Software Works [Millstein 1977], a project based on a central database of user information maintained at a master site on the network. Users were able to work through this site

to gain access to any other resource participating in the system. Although the goals of these network operating systems were important ones, the procedures were still cumbersome, and the systems themselves were not widely accepted.

The work at Xerox PARC had also laid the groundwork for a different type of network. With the rise in popularity of personal computers and workstations, many small computers were appearing within a single company or university campus. The new type of network was designed to provide more effective connection of a group of computers and related devices located within a limited area. Because the distance was not great, data transfer could be supported at a much higher rate than with conventional networks. These networks were called Local Area Networks, or LANs, and the specific network strategy developed at PARC was called Ethernet. A new computing environment was emerging, as shown in Figure 2-10. It consisted of many small computers, and a few large computers, together with terminals, printers, and other devices, all connected by a high performance network. Communication devices could also link this network to other long-distance networks and telephone lines.

The environment of a high-capacity network within a single organization, linking a number of usually similar computers with common objectives, gave a new interpretation to the idea of a network operating system. The goal of such an OS within a local area network was to provide a common control system for all of the resources on the network, making remote resources as easy to use as local ones. In the extreme case, users and programs would be unable to tell, and would not be expected to care, where the resources they were using were located. An operating system with these objectives was called a **distributed operating system**.

As the 1980s progressed, operating system research focused intensively on distributed operating systems. Dozens of experimental systems were developed, and the majority of them were based on UNIX. One example is LOCUS, described in a book by Popek and Walker [1985]. LOCUS was one of the first distributed operating systems to be transformed into a commercial product.

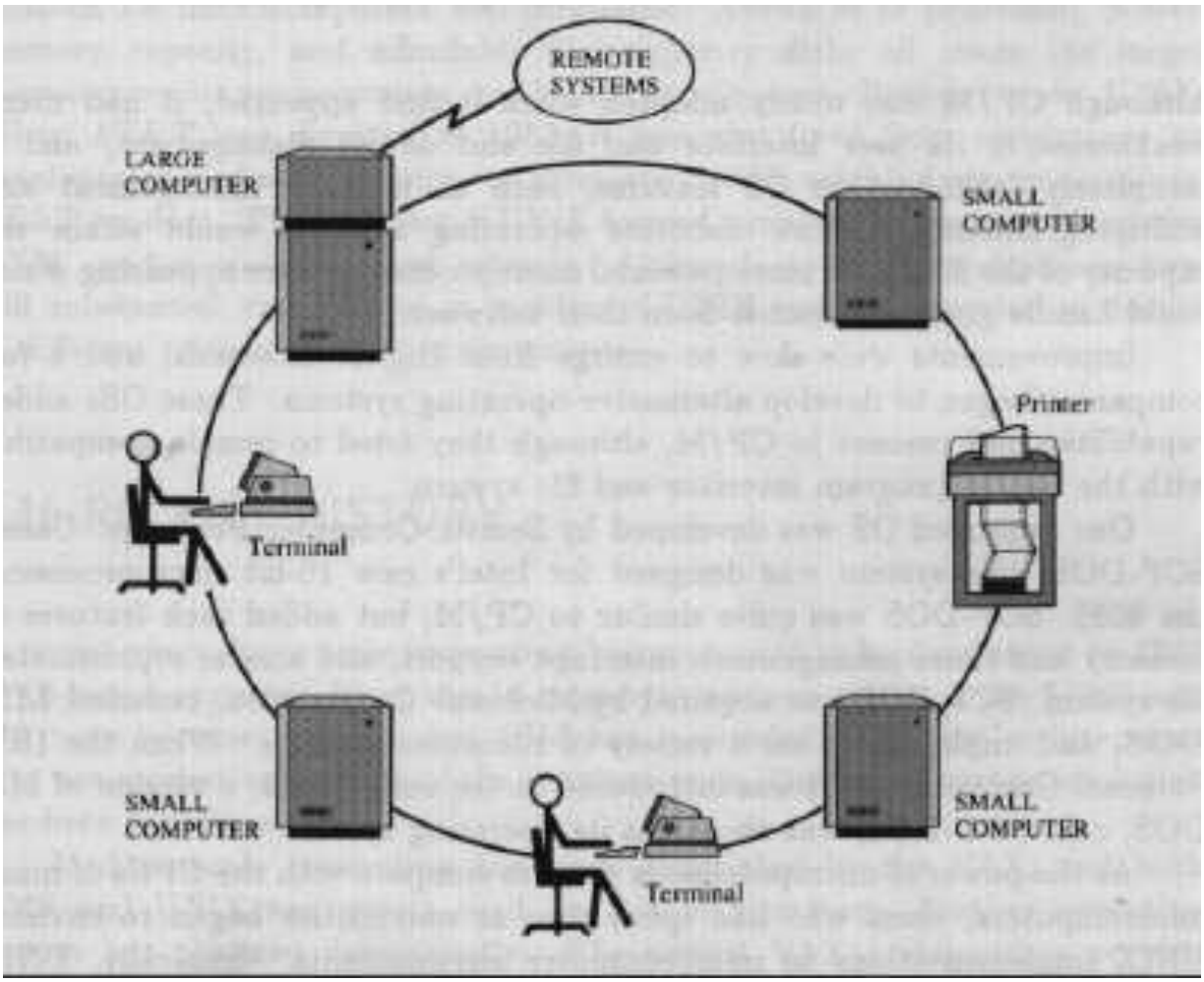

Figure 2-10: A Network of Computing Resources

2.15 CONTROLLING PARALLEL SYSTEMS

Since the advent of widespread commercial computing in the 1960s, many large-scale systems such as the IBM S/360 series have been offered in multiprocessor models. These systems included more than one processor sharing the common memory and communication paths. Clearly, managing multiple processors represents an added challenge for operating systems. These early multiprocessing environments, however, were generally organized as master-slave systems. One processor (the master) ran the OS; the other processors were assigned work to perform by the master. Differences from conventional operating systems were slight.

Beginning in the early 1980's, high-performance parallel computers made their appearance in the research lab and in the marketplace. These computers attempted to exploit multiple processing units in a more distributed and symmetric way, and required significant new approaches for resource control.

-

During the 1980s most commercial parallel computers followed the SIMD model. The Thinking Machines CM-2 was a popular example. This model involves a specialized parallel subsystem driven by a conventional host computer. The parallel processor itself has no operating system. The host runs a conventional OS (usually a version of Unix) and sends jobs to the SIMD processor in a batch mode.

While SIMD systems were entering commercial use, new types of multiprocessors, or MIMD systems, were emerging in research labs. In the early 1990s a number of these systems began to enter commercial use. Examples include the Cray T3D and Thinking Machines CM-5. These systems ran their own operating systems, and required new types of thinking in the control of parallel resources.

An early research OS for a highly-parallel multiprocessor was Hydra, developed at CMU for the C.mmp system. Hydra allowed multiple processors to run the OS, and introduced a locking mechanism to avoid conflicts. Hydra was based on the security concept of capabilities, and was an early example of an object-oriented operating system.

The successor to C.mmp, Cm*, spawned two operating systems, Medusa and StarOS. These systems had some differences in philosophy, but both explored the concept of a task force, that is, a collection of tasks cooperating on a common problem.

Other significant research OSs included Embos for the Elxsi System 6400, Dynix for the Sequent Balance, and Symunix for the NYU Ultracomputer. Commercial MIMD systems, however, continued to use more conservative OSs, typically versions of Unix, with only limited support for explicit control of tasks spanning multiple processors.

2.16 THE RISE OF MACH

During the mid-1970s, many experimental distributed computing environments were developed at various universities. One of these was a project at the University of Rochester called the Rochester Intelligient Gateway (RIG). Among the participants in this project was a PhD student named Rich Rashid.

After graduation Rashid joined the faculty at Carnegie-Mellon University, which was also developing a distributed system environment. The ideas from RIG were a strong influence on the Accent operating system at CMU.

In 1985 a project was launched at CMU to develop a state-of-the-art operating system usable in distributed environments and moving towards new structuring ideas including object-orientation. This became known as the **Mach** system. Mach was influenced significantly by Accent and also by TENEX.

Unix was the de facto standard for programming environments, and compatibility with Unix programs was considered important. Mach followed the plan of a number of other OSs by deliberately maintaining a Unix-compatible program interface even as it added new features and used a completely different internal structure.

-

The fundamental concepts of Mach are based on five types of objects: tasks, threads, ports, messages, and memory objects. The concept of threads was an important innovation that has been adopted by many later systems.

Mach was adopted as the "Unix version" by many commercial systems, especially in distributed environments.

2.17 MOVING TOWARDS PORTABILITY

In the early 1980s many computer users and programmers were experiencing a high level of frustration about arbitrary differences among the various systems that they were commonly encountering. These differences encompassed general philosophies, tools, command languages, and programming conventions. A particular concern was the many unnecessary differences in program interfaces. Various systems had no general model for even simple concepts like how to access a file or control a display terminal.

This problem was certainly acute across dissimilar systems, but it was serious even within the Unix world alone. There were many versions of Unix with subtle differences between them. An effort was launched to develop a standard for the Unix programming interface, which became known as the POSIX standard.

POSIX has made some contributions to simplifying program portability across Unix versions. Moreover, it has had a profound effect on the concept of standards in general. The POSIX work has led to a wide suite of standards projects, and to the general concept of open systems. This concept should not be confused with the open attribute for OSs such as Macintosh, described earlier. An open system in the POSIX sense is one that adheres as far as possible to common standards and so supports a maximum degree of portability and common data interchange.

The market pressure to support the open systems concept has led most prominent computing systems to move towards and promote openness, that is, adherence to industry standards. In most cases this includes the POSIX interface; most operating systems support this API to some degree even if they are not UNIX-like in any other sense. VMS has become OpenVMS (and ported to the Alpha architecture). Other systems from mainframes to micros now also stress their openness. Consortia such as the Open Software Foundation attempt to bring a common flavor to many important system interfaces.

2.18 THE MICROCOMPUTER OS MATURES

Although CP/M was widely adopted when it first appeared, it had many weaknesses in its user interface and file and device management, and it completely lacked other OS features, such as memory management and multiprogramming. More elaborate operating systems would strain

-

the capacity of the 8080, but more powerful microprocessors were appearing which could handle greater demands from their software.

Improvements were slow to emerge from Digital Research, and a few companies began to develop alternative operating systems. These OSs added capabilities not present in CP/M, although they tried to remain compatible with the CP/M program interface and file system.

One improved OS was developed by Seattle Computer Products. Called SCP-DOS, this system was designed for Intel's new 16-bit microprocessor, the 8086. SCP-DOS was quite similar to CP/M, but added such features as memory and timer management, interrupt support, and a more sophisticated file system. SCP-DOS was acquired by Microsoft Corporation, renamed MS-DOS, and implemented on a variety of computer systems. When the IBM Personal Computer (PC) was introduced in the early 1980s, a version of MS-DOS, called PC-DOS, was chosen as its operating system.

As the power of microprocessors grew to compare with the CPUs of many minicomputers, users who had spent time at universities began to envision UNIX implementations in microcomputer environments. Ironically, UNIX was no longer the lean and simple system that Thompson and Ritchie created for the PDP-7. If it had been, a moderately powerful microcomputer could have handled it well. But later versions of UNIX had grown huge in their VAX environments. A typical Berkeley version occupied several hundred thousand bytes of main memory, and required tens of megabytes of file storage for all the associated system programs.

However, if UNIX itself was out of reach, it was still possible for existing microprocessor OSs to adopt some of the features that made UNIX seem attractive. As a response to this need, Microsoft released version 2.0 of MS-DOS, which included a command interface with features from the UNIX shell, a UNIX-like program interface coexisting with the CP/M-style system calls, and a hierarchical file system. MS-DOS (and PC-DOS) became a bridge between the CP/M model and the UNIX model for microcomputer operating systems.

MS-DOS version 2 included some important features of UNIX, but lacked any support for concurrent processes. As microprocessor power has continued to grow, further enhancements were added to both MS-DOS and CP/M. These enhanced versions could support several users and in some cases several processes for each user, each communicating through a separate window on the terminal screen.

Despite improvements in MS-DOS and CP/M, the development of UNIX versions for microcomputers was inevitable. Advances in processing power, memory capacity, and affordable high-capacity disks all made the larger microcomputer environments ready to handle even Berkeley-style UNIX. When AT&T was divested in 1983, it was also freed from restrictions on development and sale of computer products. UNIX quickly became an official AT&T product. Their System V UNIX formed a major alternative to Berkeley UNIX, and it was promoted actively. Although its resource demands were still substantial, one version or another of UNIX was implemented in dozens of different microcomputer environments.

These new advances required a new look at the earlier concept of operating systems viewed simply as collections of available service routines. This concept creates clear risks in a multiprogrammed environment, since problems in one process may corrupt the entire system. This approach was suitable for personal computers such as IBM-PC and Macintosh through the late 1980s, since these systems typically ran only one program at a time. Moreover, the processor architecture on which these systems were based did not support protection mechanisms required of the OS was to have stronger control.

This situation began to change in newer personal computing environments. Even without hardware protection, IBM-PCs (running Microsoft Windows) and Apple Macintoshes (running System 7 or higher) now supported multiple applications in at least a limited form of multiprogramming. True concurrency was supported by new microprocessor models, and the demand for better OSs began to overwhelm the demand for compatibility.

One system which supported a true multitasking environment from its inception was the Commodore Amiga. Successor to earlier game-oriented Commodore systems, the Amiga adopted a research OS, TRIPOS, and rechristened it AmigaDOS. For a time the superior graphics and multitasking of the Amiga made it a strong player in the microcomputer field, overwhelming competition from another game computer company, Atari. However, Amiga could not compete with IBM or Apple products in the mainstream market.

IBM and Microsoft early recognized the need for a good multitasking environment on the IBM-PC. In the mid 1980s they jointly developed OS/2, envisioned as the successor to MS-DOS. IBM support for OS/2 continued, but Microsoft moved on to other projects. OS/2 continued for some time, but never achieved widespread popularity.

Apple provided primitive cooperative multitasking in earlier Macintosh operating systems through the Multifinder. This capability was improved with various versions of System 7. Around the dawn of the 2000s, A fundamental change occurred as the Macintosh switched to a full multitasking OS, System X, built upon a UNIX core.

2.19 THE RISE OF WINDOWS

Spurred by the popularity of the Macintosh user interface, Microsoft soon developed a graphic subsystem, called Windows, that provided some similar capabilities. Early versions of Microsoft Windows ran in partnership with MS-DOS. The first version to gain widespread popularity was Windows 3.1.

Microsoft's initial windowing environments offered primitive cooperative multitasking, but were not robust, especially because they were layered on top of MS-DOS. In the early 1990s Microsoft, with the help of some principal developers of VAX/VMS, created Windows NT (for New Technology), a new OS with important business-oriented features for multitasking, safety and security. NT was a direct competitor to OS/2; unfortunately both suffered because of compatibility issues. Most PC software was written for MS-DOS, and neither OS/2 nor NT could run DOS programs.

-

Microsoft's new answer to the problem of robustness plus compatibility was a new version of Windows based on NT but also supporting strong MS-DOS compatibility. This system was developed over several years of the early 1990s, and finally released in 1995 as Windows 95. Win 95 was a true operating system, independent of MS-DOS. To maintain DOS compatibility, a version of MS-DOS was provided that ran within a window as an application. Formerly Windows was running "on top of" DOS; from now on, what remained of DOS would run on top of Windows.

Windows 95 was followed by several additional versions, including Windows 98 and the very unpopular Windows Millenium Edition (ME), released in 2000. Each of these versions sought to improve the user interface and increase compatibility with Windows NT, but the compatibility issues were only partially solved. These versions of Windows were seen as mainly suitable for home use. Meanwhile, Windows NT was succeeded by Windows 2000, a distinctly different OS with more enterprise-oriented features for business use, including robust networking. Further, Windows NT/2000 used a completely different type of file system from other Windows versions.

In the mid-2000s, the competition between the NT and Win 95 series was finally resolved with the introduction of Windows XP. Although XP had several versions, all were based on the NT architecture, used the NT file system, and provided the type of user interface that had been developed for Windows 95/98/ME.

2.20 HERE COMES LINUX

Among the may versions of UNIX that made their appearance in the 1970s and 1980s were several designed primarily for teaching purposes. The most notable of these was MINIX, distributed along with a popular OS textbook by Tanenbaum [1992]. Like various other UNIX incarnations such as Mach or LOCUS, Minix was designed to "look like" (a subset of) Unix from the outside, although it was built quite differently on the inside. This means that Minix offered a user interface and commands very similar to UNIX, and could run certain types of UNIX programs.

MINIX and its textbook were very popular. Among those who were learning from it was a researcher named Linus Torvaalds. Students were encouraged to modify and extend the basic code for MINIX, and Torvaalds accepted this challenge and set out to create a much more complete and robust version, running on PC hardware, which he named Linux. From the start, Linux was developed in an "open source" fashion, with extensive help and participation from the increasingly networked resarch community.

Originally conceived as a UNIX for the masses, Linux has enjoyed phenomenal success, and today competes seriously with established commercial OSs, especially in applications such as Internet servers.

-

2.21 RECENT HISTORY

In recent years, large scale computing has continued to be dominated by IBM MVS/ESA-based systems. These systems promote more openness by the inclusion of large standard subsystems such as database managers. The newest versions of the IBM computers are designated S/390, and the OS has been renamed OS/390. Today's OS/390 supports interactive users as well as batch processing and includes a full Unix environment as a subsystem.

Medium-scale computing, long dominated by the VAX, has moved toward RISC architectures. Sun's SPARC architecture is popular for high-end workstations, running various versions of UNIX. Digital developed the Alpha architecture, where both OpenVMS and UNIX were supported; this has now been retired by HP, which purchased Compaq after Compaq purchased Digital. IBM continues development of mid-range systems such as the AS/400, now using the PowerPC. Other important players include Hewlett-Packard and Silicon Graphics. Over 50 versions of Unix are used on various systems in this class.

Small computer environments are dominated by IBM-PCs running Windows XP, Vista, or Windows 7, and Macintoshes (now using the Intel architecture) running OS X. Versions of Linux also abound in such environments. Increasingly, computers with Intel architectures provide multiboot capability or virtual machine support, allowing different OS types to be used alternately or simultaneously.

In today's OS world, it may appear to some that UNIX and Windows are the only significant players. The reality, however, is far richer.

Other operating systems have continued to develop to meet new categories of needs. Much of the new OS development has focused on improving security and reliability. Advances in this area run the gamut from secure versions of the basically insecure UNIX [Popek et al. 1979] to operating systems for secure object-oriented computer architectures, such as CAP [Wilkes & Needham 1979], or highly reliable architectures like those developed by Tandem [Serlin 1984]. Other systems have focused on improving their information handling capabilities, raising the file system to the level of a database manager. The PICK system [Sisk & VanArsdale 1985] is a small computer OS illustrating this approach.

Operating systems for real-time applications have also developed steadily. Although many such applications require special-purpose system software for the highest possible speed, others benefit from the use of general-purpose real-time operating systems. OSs of increasing sophistication have been supplied with control-oriented minicomputers like IBM's System/34 and Series/1 and similar systems from DEC and other manufacturers. Many control-oriented microcomputers have likewise been supplied with real-time operating systems. A few general-purpose real-time OSs have also been developed independently and implemented on several different computer types. VRTX [Ready 1986] is a prominent example. ITRON, an embedded real-time OS architecture, now has more implementations than any other OS.

More ambitious dynamic real-time systems, in which tasks and resource use cannot be fully predicted, have gained popularity. One example is the Spring Kernel [Stankovic & Ramanrithram 1987].

Finally, operating systems for handheld computers have become a fast growing field, with competition continuing between stripped-down versions of Windows and the PalmOS. The Symbian OS dominates for many cellphone types, while a compressed version of Apple's OSX known as iOS powers the iPhone. The newest player in this field is the Android OS, developed by Google, which runs on a number of mobile devices provided by Google and others.

FOR FURTHER READING

A number of surveys and more extensive treatments of aspects of operating system history have been published. Surveys of operating systems in the early period were published by Rosen [1967, 1969, 1972], Rosin [1969], and Weizer [1981]. Steel [1964] provides a good snapshot of the industry in this early period. For an in-depth treatment of one very early system, the series of papers on SOS [Shell et al. 1959] are worthwhile. A good discussion of early IBM systems is also contained in the extensive history by Bashe et al. [1986].

Various papers have presented the evolution of operating systems for specific computers or classes of computers. These include the IBM S/360 and S/370 [Auslander et al. 1981], DEC PDP-10 [Bell et al. 1978], Univac 1100 series [Borgerson et al. 1978], IBM real-time computers [Harrison et al. 1981], and minicomputers [Koudela 1973].

The subject of virtual machine operating systems is covered in surveys by Buzen and Gagliardi [1973] and Goldberg [1974]. The evolution of VM/370 is presented by Creasy [1981].

The early evolution of UNIX is described by Ritchie [1980]. Quaterman et al. [1985] discuss later UNIX history, covering VAX versions. The early history of microprocessors at Intel is reviewed by Noyce and Hoff [1981]. A detailed and interesting book by Freiberger and Swaine [1984] describes the evolution of microcomputers, from their earliest versions up to the appearance of personal workstations, including coverage of CP/M and MS-DOS.

The history of Parallel Operating Systems is reviewed by Almasi & Gottlieb [1994]. Boykin et al [1993, Chapter 1] describe the history of Mach. The evolution of the POSIX standards is described by Jesperin [1995].

Among more specialized topics, Barron et al. [1972] discuss the evolution of job-control languages, and Fosdick [1979] surveys the use of high-level languages for writing operating systems.

REVIEW QUESTIONS

1. Name two important OS ideas first explored by the T.H.E. operating system.

2. Name two OS features pioneered by the ATLAS system.

-

3. Name two unusual characteristics of the MULTICS system.

4. Explain at least three developments in system hardware and software that had occurred by the late 1960s that made timesharing more feasible than it could have been ten years earlier.

5. What was the greatest problem in designing a timesharing system that was not faced by batch systems?

6. Name an operating system that meets each of the descriptions in the following list:

   i. A very early OS developed by an IBM users group
   ii. A large OS written almost entirely in PL/I
   iii. Introduced the concept of interrupts
   iv. Popular OS originally created by programmers for their own use
   v. Introduced the idea of layered system organization
   vi. The largest and most complex OS ever developed
   vii. First operating system for microcomputers
   viii. A transparent distributed OS based on UNIX
   ix. A pioneer OS for personal workstations

7. Explain briefly why the objectives of a batch operating system conflict with those of an interactive system.

8. What is unusual about the manner in which the UNIX operating system was developed?

9. Give a principal reason why the VAX was developed.

10. Why were the goals of operating systems for local area networks different than those for long-distance networks?

ASSIGNMENTS

1. Draw a graph based on Figure 2-1, showing which early operating systems were a significant influence on each later one. Identify some of the distinct separate lines of development.

2. Discuss the reasons for the widespread popularity of each of the following: MVS, VMS, UNIX, Linux, Windows 2000, Palm OS.

3. Support or refute the following claim: In general, operating systems developed by computer manufacturers have been less influential than those developed independently.

4. Trace the genealogy of a current OS of your choice. Identify several features of this OS that are design innovations, and several others that are included primarily for historical reasons.