Operating system

An **operating system** (**OS**) is system software that manages computer hardware and software resources, and provides common services for computer programs.

Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, peripherals, and other resources.

For hardware functions such as input and output and memory allocation, the operating system acts as an intermediary between programs and the computer hardware,[1][2] although the application code is usually executed directly by the hardware and frequently makes system calls to an OS function or is interrupted by it. Operating systems are found on many devices that contain a computer – from cellular phones and video game consoles to web servers and supercomputers.

In the personal computer market, as of September 2023, Microsoft Windows holds a dominant market share of around 68%. macOS by Apple Inc. is in second place (20%), and the varieties of Linux, including ChromeOS, are collectively in third place (7%).[3] In the mobile sector (including smartphones and tablets), as of September 2023, Android's share is 68.92%, followed by Apple's iOS and iPadOS with 30.42%, and other operating systems with .66%.[4] Linux distributions are dominant in the server and supercomputing sectors. Other specialized classes of operating systems (special-purpose operating systems),[5][6] such as embedded and real-time systems, exist for many applications. Security-focused operating systems also exist. Some operating systems have low system requirements (e.g. light-weight Linux distribution). Others may have higher system requirements.

Some operating systems require installation or may come pre-installed with purchased computers (OEM-installation), whereas others may run directly from media (i.e. live CD) or flash memory (i.e. USB stick).

Types of operating systems

Single-tasking and multi-tasking

A single-tasking system can only run one program at a time, while a multi-tasking operating system allows more than one program to be running concurrently. This is achieved by time-sharing, where the available processor time is divided between multiple processes. These processes are each interrupted repeatedly in time slices by a task-scheduling subsystem of the operating system. Multi-tasking may be characterized in preemptive and cooperative types. In preemptive multitasking, the operating system slices the CPU time and dedicates a slot to each of the programs. Unix-like operating systems, such as Linux—as well as non-Unix-like, such as AmigaOS—support preemptive multitasking. Cooperative multitasking is achieved by relying on each process to provide time to the other processes in a defined manner. 16-bit versions of Microsoft Windows used cooperative multi-tasking; 32-bit versions of both Windows NT and Win9x used preemptive multi-tasking.

Single- and multi-user

Single-user operating systems have no facilities to distinguish users but may allow multiple programs to run in tandem.[7] A multi-user operating system extends the basic concept of multi-tasking with facilities that identify processes and resources, such as disk space, belonging to multiple users, and the system permits multiple users to interact with the system at the same time. Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources to multiple users.

Distributed

A distributed operating system manages a group of distinct, networked computers and makes them appear to be a single computer, as all computations are distributed (divided amongst the constituent computers).[8]

Embedded

Embedded operating systems are designed to be used in embedded computer systems. They are designed to operate on small machines with less autonomy (e.g. PDAs). They are very compact and extremely efficient by design and are able to operate with a limited amount of resources. Windows CE and Minix 3 are some examples of embedded operating systems.

Real-time

A real-time operating system is an operating system that guarantees to process events or data by a specific moment in time. A real-time operating system may be single- or multi-tasking, but when multitasking, it uses specialized scheduling algorithms so that a deterministic nature of behavior is achieved. Such an event-driven system switches between tasks based on their priorities or external events, whereas time-sharing operating systems switch tasks based on clock interrupts.

Library

A library operating system is one in which the services that a typical operating system provides, such as networking, are provided in the form of libraries and composed with the application and configuration code to construct a unikernel: a specialized, single address space, machine image that can be deployed to cloud or embedded environments.

History

Early computers were built to perform a series of single tasks, like a calculator. Basic operating system features were developed in the 1950s, such as resident monitor functions that could automatically run different programs in succession to speed up processing. Operating systems did not exist in their modern and more complex forms until the early 1960s.[9] Hardware features were added, that enabled use of runtime libraries, interrupts, and parallel processing. When personal computers became popular in the 1980s, operating systems were made for them similar in concept to those used on larger computers.

In the 1940s, the earliest electronic digital systems had no operating systems. Electronic systems of this time were programmed on rows of mechanical switches or by jumper wires on plugboards. These were special-purpose systems that, for example, generated ballistics tables for the military or controlled the printing of payroll checks from data on punched paper cards. After programmable general-purpose computers were invented, machine languages(consisting of strings of the binary digits 0 and 1 on punched paper tape) were introduced that sped up the programming process (Stern, 1981).

In the early 1950s, a computer could execute only one program at a time. Each user had sole use of the computer for a limited period and would arrive at a scheduled time with their program and data on punched paper cards or punched tape. The program would be loaded into the machine, and the machine would be set to work until the program completed or crashed. Programs could generally be debugged via a front panel using toggle switches and panel lights. It is said that Alan Turing was a master of this on the early Manchester Mark 1 machine, and he was already deriving the primitive conception of an operating system from the principles of the universal Turing machine.[9]

An IBM System 360/65 Operator's Panel. OS/360 was used on most IBM mainframe computers beginning in 1966, including computers used by the Apollo program.

Later machines came with libraries of programs, which would be linked to a user's program to assist in operations such as input and output and compiling (generating machine code from human-readable symbolic code). This was the genesis of the modern-day operating system. However, machines still ran a single job at a time. At Cambridge University in England, the job queue was at one time a washing line (clothesline) from which tapes were hung with different colored clothes-pegs to indicate job priority.

By the late 1950s, programs that one would recognize as an operating system were beginning to appear. Often pointed to as the earliest recognizable example is GM-NAA I/O, released in 1956 on the IBM 704. The first known example that actually referred to itself was the SHARE Operating System, a development of GM-NAA I/O, released in 1959. In a May 1960 paper describing the system, George Ryckman noted:

> The development of computer operating systems have materially aided the problem of getting a program or series of programs on and off the computer efficiently.[10]

One of the more famous examples that is often found in discussions of early systems is the Atlas Supervisor, running on the Atlas in 1962.[11] It was referred to as such in a December 1961 article describing the system, but the context of "the Operating System" is more along the lines of "the system operates in the fashion". The Atlas team itself used the term "supervisor",[12] which was widely used along with "monitor". Brinch Hansen described it as "the most significant breakthrough in the history of operating systems."[13]

Mainframes

Through the 1950s, many major features were pioneered in the field of operating systems on mainframe computers, including batch processing, input/output interrupting, buffering, multitasking, spooling, runtime libraries, link-loading, and programs for sorting records in files. These features were included or not included in application software at the option of application programmers, rather than in a separate operating system used by all applications. In 1959, the SHARE Operating System was released as an integrated utility for the IBM 704, and later in the 709 and 7090 mainframes, although it was quickly supplanted by IBSYS/IBJOB on the 709, 7090 and 7094, which in turn influenced the later 7040-PR-150 (7040/7044) and 1410-PR-155 (1410/7010) operating systems.

During the 1960s, IBM's OS/360 introduced the concept of a single OS spanning an entire product line, which was crucial for the success of the System/360 machines. IBM's current mainframe operating systems are distant descendants of this original system and modern machines are backward compatible with applications written for OS/360.

OS/360 also pioneered the concept that the operating system keeps track of all of the system resources that are used, including program and data space allocation in main memory and file space in secondary storage, and file locking during updates. When a process is terminated for any reason, all of these resources are re-claimed by the operating system.

The alternative CP-67 system for the S/360-67 started a whole line of IBM operating systems focused on the concept of virtual machines. Other operating systems used on IBM S/360 series mainframes included systems developed by IBM: DOS/360[a] (Disk Operating System), TSS/360 (Time Sharing System), TOS/360 (Tape Operating System), BOS/360 (Basic Operating System), and ACP (Airline Control Program), as well as a few non-IBM systems: MTS (Michigan Terminal System), MUSIC (Multi-User System for Interactive Computing), and ORVYL (Stanford Timesharing System).

Control Data Corporation developed the SCOPE operating system in the 1960s, for batch processing. In cooperation with the University of Minnesota, the Kronos and later the NOS operating systems were developed during the 1970s, which supported simultaneous batch and timesharing use. Like many commercial timesharing systems, its interface was an extension of the Dartmouth BASIC operating systems, one of the pioneering efforts in timesharing and programming languages. In the late 1970s, Control Data and the University of Illinois developed the PLATO operating system, which used plasma panel displays and long-distance time sharing networks. Plato was remarkably innovative for its time, featuring real-time chat, and multi-user graphical games.

In 1961, Burroughs Corporation introduced the B5000 with the MCP (Master Control Program) operating system. The B5000 was a stack machine designed to exclusively support high-level languages with no assembler;[b] indeed, the MCP was the first OS to be written exclusively in a high-level language (ESPOL, a dialect of ALGOL). MCP also introduced many other ground-breaking innovations, such as being the first commercial implementation of virtual memory. MCP is still in use today in the Unisys company's MCP/ClearPath line of computers.

UNIVAC, the first commercial computer manufacturer, produced a series of EXEC operating systems.[14][15][16] Like all early main-frame systems, this batch-oriented system managed magnetic drums, disks, card readers and line printers. In the 1970s, UNIVAC produced the Real-Time Basic (RTB) system to support large-scale time sharing, also patterned after the Dartmouth BC system.

General Electric developed General Electric Comprehensive Operating Supervisor (GECOS), which primarily supported batch processing. After its acquisition by Honeywell, it was renamed General Comprehensive Operating System (GCOS).

Bell Labs,[c] General Electric and MIT developed Multiplexed Information and Computing Service (Multics), which introduced the concept of ringed security privilege levels.

Digital Equipment Corporation developed many operating systems for its various computer lines, including TOPS-10 and TOPS-20 time-sharing systems for the 36-bit PDP-10 class systems. Before the widespread use of UNIX, TOPS-10 was a particularly popular system in universities, and in the early ARPANET community. RT-11 was a single-user real-time OS for the PDP-11 class minicomputer, and RSX-11 was the corresponding multi-user OS.

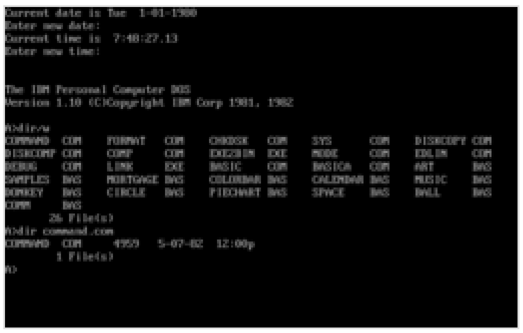
From the late 1960s through the late 1970s, several hardware capabilities evolved that allowed similar or ported software to run on more than one system. Early systems had utilized microprogramming to implement features on their systems in order to permit different underlying computer architectures to appear to be the same as others in a series. In fact, most 360s after the 360/40 (except the 360/44, 360/75, 360/91, 360/95 and 360/195) were microprogrammed implementations.

The enormous investment in software for these systems made since the 1960s caused most of the original computer manufacturers to continue to develop compatible operating systems along with the hardware. Notable supported mainframe operating systems include:

- Burroughs MCP – B5000, 1961 to Unisys Clearpath/MCP, present
- IBM OS/360 – IBM System/360, 1966 to IBM z/OS, present
- IBM CP-67 – IBM System/360, 1967 to IBM z/VM, present
- UNIVAC EXEC 8 – UNIVAC 1108, 1967, to OS 2200 Unisys Clearpath Dorado, present

Microcomputers

The first microcomputers did not have the capacity or need for the elaborate operating systems that had been developed for mainframes and minicomputers; minimalistic operating systems were developed, often loaded from ROM and known as *monitors*. One notable early disk operating system was CP/M, which was supported on many early microcomputers and was closely imitated by Microsoft's MS-DOS, which became widely popular as the operating system chosen for the IBM PC (IBM's version of it was called IBM DOS or PC DOS).



```
Current date is Tue  1-01-1980
Enter new date:
Current time is  7:48:27.13
Enter new time:


The IBM Personal Computer DOS
Version 1.10 (C)Copyright IBM Corp 1981, 1982

A>dir/w
COMMAND  COM    FORMAT   COM    CHKDSK   COM    SYS      COM    DISKCOPY COM
DISKCOMP COM    COMP     COM    EXE2BIN  EXE    MODE     COM    EDLIN    COM
DEBUG    COM    LINK     EXE    BASIC    COM    BASICA   COM    ART      BAS
SAMPLES  BAS    MORTGAGE BAS    COLORBAR BAS    CALENDAR BAS    MUSIC    BAS
DONKEY   BAS    CIRCLE   BAS    PIECHART BAS    SPACE    BAS    BALL     BAS
COMM     BAS
       26 File(s)
A>dir command.com
COMMAND  COM     4959   5-07-82  12:00p
        1 File(s)
A>
```

PC DOS was an early personal computer OS that featured a command-line interface.

In the 1980s, Apple Computer Inc. (now Apple Inc.) introduced the Apple Macintosh alongside its popular Apple II series of microcomputers. The Macintosh had an innovative graphical user interface (GUI) and a mouse; it ran an operating system later known as the (classic) Mac OS.

The introduction of the Intel 80386 CPU chip in October 1985,[17] with 32-bit architecture and paging capabilities, provided personal computers with the ability to run multitasking operating systems like those of earlier superminicomputers and mainframes. Microsoft responded to this progress by hiring Dave Cutler, who had developed the VMS operating system for Digital Equipment Corporation. He would lead the development of the Windows NT operating system, which continues to serve as the basis for Microsoft's operating systems line. Steve Jobs, a co-founder of Apple Inc., started NeXT Computer Inc., which developed the NeXTSTEP operating system. NeXTSTEP would later be acquired by Apple Inc. and used, along with code from FreeBSD as the core of Mac OS X (macOS after latest name change).

The GNU Project was started by activist and programmer Richard Stallman with the goal of creating a complete free software replacement to the proprietary UNIX operating system. While the project was highly successful in duplicating the functionality of various parts of UNIX, development of the GNU Hurd kernel proved to be unproductive. In 1991, Finnish computer science student Linus Torvalds, with cooperation from volunteers collaborating over the Internet, released the first version of the Linux kernel. It was soon merged with the GNU user space components and system software to form a complete operating system. Since then, the combination of the two major components has usually been referred to as simply "Linux" by the software industry, a naming convention that Stallman and the Free Software Foundation remain opposed to, preferring the name GNU/Linux. The Berkeley Software Distribution, known as BSD, is the UNIX derivative distributed by the University of California, Berkeley, starting in the 1970s. Freely distributed and ported to many minicomputers, it eventually also gained a following for use on PCs, mainly as FreeBSD, NetBSD and OpenBSD.

Examples

Unix and Unix-like operating systems

Evolution of Unix systems

Unix was originally written in assembly language.[18] Ken Thompson wrote B, mainly based on BCPL, based on his experience in the MULTICS project. B was replaced by C, and Unix, rewritten in C, developed into a large, complex family of inter-related operating systems which have been influential in every modern operating system (see History).

The *Unix-like* family is a diverse group of operating systems, with several major sub-categories including System V, BSD, and Linux. The name "UNIX" is a trademark of The Open Group which licenses it for use with any operating system that has been shown to conform to their definitions. "UNIX-like" is commonly used to refer to the large set of operating systems which resemble the original UNIX.

Unix-like systems run on a wide variety of computer architectures. They are used heavily for servers in business, as well as workstations in academic and engineering environments. Free UNIX variants, such as Linux and BSD, are popular in these areas.

Five operating systems are certified by The Open Group (holder of the Unix trademark) as Unix. HP's HP-UX and IBM's AIX are both descendants of the original System V Unix and are designed to run only on their respective vendor's hardware. In contrast, Sun Microsystems's Solaris can run on multiple types of hardware, including x86 and SPARC servers, and PCs. Apple's macOS, a replacement for Apple's earlier (non-Unix) classic Mac OS, is a hybrid kernel-based BSD variant derived from NeXTSTEP, Mach, and FreeBSD. IBM's z/OS UNIX System Services includes a shell and utilities based on Mortice Kerns' InterOpen products.

Unix interoperability was sought by establishing the POSIX standard. The POSIX standard can be applied to any operating system, although it was originally created for various Unix variants.

BSD and its descendants

A subgroup of the Unix family is the Berkeley Software Distribution family, which includes FreeBSD, NetBSD, and OpenBSD. These operating systems are most commonly found on webservers, although they can also function as a personal computer OS. The Internet owes much of its existence to BSD, as many of the protocols now commonly used by computers to connect, send and receive data over a network were widely implemented and refined in BSD. The World Wide Web was also first demonstrated on a number of computers running an OS based on BSD called NeXTSTEP.

The first server for the World Wide Web ran on NeXTSTEP, based on BSD.

In 1974, University of California, Berkeley installed its first Unix system. Over time, students and staff in the computer science department there began adding new programs to make things easier, such as text editors. When Berkeley received new VAX computers in 1978 with Unix installed, the school's undergraduates modified Unix even more in order to take advantage of the computer's hardware possibilities. The Defense Advanced Research Projects Agency of the US Department of Defense took interest, and decided to fund the project. Many schools, corporations, and government organizations took notice and started to use Berkeley's version of Unix instead of the official one distributed by AT&T.

Steve Jobs, upon leaving Apple Inc. in 1985, formed NeXT Inc., a company that manufactured high-end computers running on a variation of BSD called NeXTSTEP. One of these computers was used by Tim Berners-Lee as the first webserver to create the World Wide Web.

Developers like Keith Bostic encouraged the project to replace any non-free code that originated with Bell Labs. Once this was done, however, AT&T sued. After two years of legal disputes, the BSD project spawned a number of free derivatives, such as NetBSD and FreeBSD (both in 1993), and OpenBSD (from NetBSD in 1995).

macOS

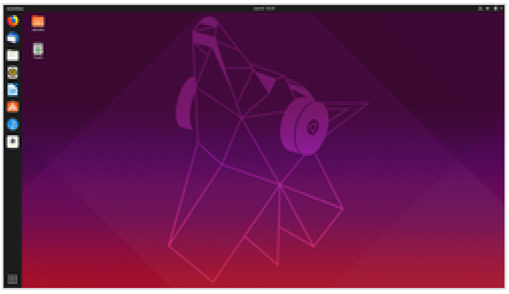
macOS (formerly "Mac OS X" and later "OS X") is a line of open core graphical operating systems developed, marketed, and sold by Apple Inc., the latest of which is pre-loaded on all currently shipping Macintosh computers. macOS is the successor to the original classic Mac OS, which had been Apple's primary operating system since 1984. Unlike its predecessor, macOS is a UNIX operating system built on technology that had been developed at NeXT through the second half of the 1980s and up until Apple purchased the company in early 1997. The operating system was first released in 1999 as Mac OS X Server 1.0, followed in March 2001 by a client version (Mac OS X v10.0 "Cheetah"). Since then, six more distinct "client" and "server" editions of macOS have been released, until the two were merged in OS X 10.7 "Lion".

Prior to its merging with macOS, the server edition – macOS Server – was architecturally identical to its desktop counterpart and usually ran on Apple's line of Macintosh server hardware. macOS Server included work group management and administration software tools that provide simplified access to key network services, including a mail transfer agent, a Samba server, an LDAP server, a domain name server, and others. With Mac OS X v10.7 Lion, all server aspects of Mac OS X Server have been integrated into the client version and the product re-branded as "OS X" (dropping "Mac" from the name). The server tools are now offered as an application.[19]

z/OS UNIX System Services

First introduced as the OpenEdition upgrade to MVS/ESA System Product Version 4 Release 3, announced[20] February 1993 with support for POSIX and other standards.[21][22][23] z/OS UNIX System Services is built on top of MVS services and cannot run independently. While IBM initially introduced OpenEdition to satisfy FIPS requirements, several z/OS component now require UNIX services, e.g., TCP/IP.

Linux


Ubuntu, desktop Linux distribution

The Linux kernel originated in 1991, as a project of Linus Torvalds, while a university student in Finland. He posted information about his project on a newsgroup for computer students and programmers, and received support and assistance from volunteers who succeeded in creating a complete and functional kernel.

Linux is Unix-like, but was developed without any Unix code, unlike BSD and its variants. Because of its open license model, the Linux kernel code is available for study and modification, which resulted in its use on a wide range of computing machinery from supercomputers to smartwatches. Although estimates suggest that Linux is used on only 2.81% of all "desktop" (or laptop) PCs,[3] it has been widely adopted for use in servers[28] and embedded systems[29] such as cell phones.

A picture of Tux the penguin, the mascot of Linux. Linux is a Unix-like operating system that was first released on September 17, 1991 by Linus Torvalds.[24][25][26][27]

Linux has superseded Unix on many platforms and is used on most supercomputers, including all 500 most powerful supercomputers on the TOP500 list — having displaced all competitors by 2017.[30] Linux is also commonly used on other small energy-efficient computers, such as smartphones and smartwatches. The Linux kernel is used in some popular distributions, such as Red Hat, Debian, Ubuntu, Linux Mint and Google's Android, ChromeOS, and ChromiumOS.

Microsoft Windows

Microsoft Windows is a family of proprietary operating systems designed by Microsoft Corporation and primarily targeted to x86 architecture based computers. As of 2022, its worldwide market share on all platforms was approximately

30%,[31] and on the desktop/laptop platforms, its market share was approximately 75%.[32] The latest version is Windows 11.

Microsoft Windows was first released in 1985, as an operating environment running on top of MS-DOS, which was the standard operating system shipped on most Intel architecture personal computers at the time. In 1995, Windows 95 was released which only used MS-DOS as a bootstrap. For backwards compatibility, Win9x could run real-mode MS-DOS[33][34] and 16-bit Windows 3.x[35] drivers. Windows ME, released in 2000, was the last version in the Win9x family. Later versions have all been based on the Windows NT kernel. Current client versions of Windows run on IA-32, x86-64 and Arm microprocessors.[36] In the past, Windows NT supported additional architectures.

Server editions of Windows are widely used, however, Windows' usage on servers is not as widespread as on personal computers as Windows competes against Linux and BSD for server market share.[37][38]

ReactOS is a Windows-alternative operating system, which is being developed on the principles of Windows – without using any of Microsoft's code.

Other

There have been many operating systems that were significant in their day but are no longer so, such as AmigaOS; OS/2 from IBM and Microsoft; classic Mac OS, the non-Unix precursor to Apple's macOS; BeOS; XTS-300; RISC OS; MorphOS; Haiku; BareMetal and FreeMint. Some are still used in niche markets and continue to be developed as minority platforms for enthusiast communities and specialist applications.

The z/OS operating system for IBM z/Architecture mainframe computers is still being used and developed, and OpenVMS, formerly from DEC, is still under active development by VMS Software Inc. The IBM i operating system for IBM AS/400 and IBM Power Systems midrange computers is also still being used and developed.

Yet other operating systems are used almost exclusively in academia, for operating systems education or to do research on operating system concepts. A typical example of a system that fulfills both roles is MINIX, while for example Singularity is used purely for research. Another example is the Oberon System designed at ETH Zürich by Niklaus Wirth, Jürg Gutknecht and a group of students at the former Computer Systems Institute in the 1980s. It was used mainly for research, teaching, and daily work in Wirth's group.

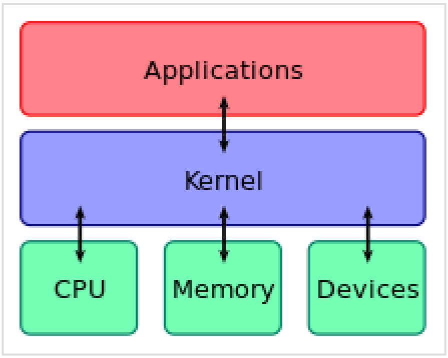
Other operating systems have failed to win significant market share, but have introduced innovations that have influenced mainstream operating systems, not least Bell Labs' Plan 9.

Components

The components of an operating system all exist in order to make the different parts of a computer work together. All user software needs to go through the operating system in order to use any of the hardware, whether it be as simple as a mouse or keyboard or as complex as an Internet component.

Kernel

With the aid of firmware and device drivers, the kernel provides the most basic level of control over all of the computer's hardware devices. It manages memory access for programs in the RAM, it determines which programs get access to which hardware resources, it sets up or resets the CPU's operating states for optimal operation at all times, and it organizes the data for long-term non-volatile storage with file systems on such media as disks, tapes, flash memory, etc.



A kernel connects the application software to the hardware of a computer.

Program execution

The operating system provides an interface between an application program and the computer hardware, so that an application program can interact with the hardware only by obeying rules and procedures programmed into the operating system. The operating system is also a set of services which simplify development and execution of application programs. Executing an application program typically involves the creation of a process by the operating system kernel, which assigns memory space and other resources, establishes a priority for the process in multi-tasking systems, loads program binary code into memory, and initiates execution of the application program, which then interacts with the user and with hardware devices. However, in some systems an application can request that the operating system execute another application within the same process, either as a subroutine or in a separate thread, e.g., the **LINK** and **ATTACH** facilities of OS/360 and successors.

Interrupts

An interrupt (also known as an abort, exception, *fault*, signal,[39] or *trap*)[40] provides an efficient way for most operating systems to react to the environment. Interrupts cause the central processing unit (CPU) to have a control flow change away from the currently running program to an interrupt handler, also known as an interrupt service routine (ISR).[41][42] An interrupt service routine may cause the central processing unit (CPU) to have a context switch.[43][d] The details of how a computer processes an interrupt vary from architecture to architecture, and the details of how interrupt service routines behave vary from operating system to operating system.[44] However, several interrupt functions are common.[44] The architecture and operating system must:[44]

1. transfer control to an interrupt service routine.
2. save the state of the currently running process.
3. restore the state after the interrupt is serviced.

Software interrupt

A software interrupt is a message to a process that an event has occurred.[39] This contrasts with a *hardware interrupt* — which is a message to the central processing unit (CPU) that an event has occurred.[45] Software interrupts are similar to hardware interrupts — there is a change away from the currently running process.[46] Similarly, both hardware and software interrupts execute an interrupt service routine.

Software interrupts may be normally occurring events. It is expected that a time slice will occur, so the kernel will have to perform a context switch.[47] A computer program may set a timer to go off after a few seconds in case too much data causes an algorithm to take too long.[48]

Software interrupts may be error conditions, such as a malformed machine instruction.[48] However, the most common error conditions are division by zero and accessing an invalid memory address.[48]

Users can send messages to the kernel to modify the behavior of a currently running process.[48] For example, in the command-line environment, pressing the *interrupt character* (usually Control-C) might terminate the currently running process.[48]

To generate *software interrupts* for x86 CPUs, the INT assembly language instruction is available.[49] The syntax is `INT X`, where `X` is the offset number (in hexadecimal format) to the interrupt vector table.

Signal

To generate *software interrupts* in Unix-like operating systems, the `kill(pid,signum)` system call will send a signal to another process.[50] `pid` is the process identifier of the receiving process. `signum` is the signal number (in mnemonic format)[e] to be sent. (The abrasive name of `kill` was chosen because early implementations only terminated the process.)[51]

In Unix-like operating systems, *signals* inform processes of the occurrence of asynchronous events.[50] To communicate asynchronously, interrupts are required.[52] One reason a process needs to asynchronously communicate to another process solves a variation of the classic reader/writer problem.[53] The writer receives a pipe from the shell for its output to be sent to the reader's input stream.[54] The command-line syntax is `alpha | bravo`. `alpha` will write to the pipe when its computation is ready and then sleep in the wait queue.[55] `bravo` will then be moved to the ready queue and soon will read from its input stream.[56] The kernel will generate *software interrupts* to coordinate the piping.[56]

Signals may be classified into 7 categories.[50] The categories are:

1. when a process finishes normally.
2. when a process has an error exception.
3. when a process runs out of a system resource.
4. when a process executes an illegal instruction.
5. when a process sets an alarm event.
6. when a process is aborted from the keyboard.
7. when a process has a tracing alert for debugging.

Hardware interrupt

Input/Output (I/O) devices are slower than the CPU. Therefore, it would slow down the computer if the CPU had to wait for each I/O to finish. Instead, a computer may implement interrupts for I/O completion, avoiding the need for polling or busy waiting.[57]

Some computers require an interrupt for each character or word, costing a significant amount of CPU time. Direct memory access (DMA) is an architecture feature to allow devices to bypass the CPU and access main memory directly.[58] (Separate from the architecture, a device may perform direct memory access[f] to and from main memory either directly or via a bus.)[59][g]

Input/Output

Interrupt-driven I/O

When a computer user types a key on the keyboard, typically the character appears immediately on the screen. Likewise, when a user moves a mouse, the cursor immediately moves across the screen. Each keystroke and mouse movement generates an *interrupt* called *Interrupt-driven I/O*. An interrupt-driven I/O occurs when a process causes an interrupt for every character[59] or word[60] transmitted.

Direct Memory Access

Devices such as hard disk drives, solid state drives, and magnetic tape drives can transfer data at a rate high enough that interrupting the CPU for every byte or word transferred, and having the CPU transfer the byte or word between the device and memory, would require too much CPU time. Data is, instead, transferred between the device and memory independently of the CPU by hardware such as a channel or a direct memory access controller; an interrupt is delivered only when all the data is transferred.[61]

If a computer program executes a system call to perform a block I/O *write* operation, then the system call might execute the following instructions:

- Set the contents of the CPU's registers (including the program counter) into the process control block.[62]
- Create an entry in the device-status table.[63] The operating system maintains this table to keep track of which processes are waiting for which devices. One field in the table is the memory address of the process control block.
- Place all the characters to be sent to the device into a memory buffer.[52]
- Set the memory address of the memory buffer to a predetermined device register.[64]
- Set the buffer size (an integer) to another predetermined register.[64]
- Execute the machine instruction to begin the writing.
- Perform a context switch to the next process in the ready queue.

While the writing takes place, the operating system will context switch to other processes as normal. When the device finishes writing, the device will *interrupt* the currently running process by *asserting* an interrupt request. The device will also place an integer onto the data bus.[65] Upon accepting the interrupt request, the operating system will:

- Push the contents of the program counter (a register) followed by the status register onto the call stack.[44]
- Push the contents of the other registers onto the call stack. (Alternatively, the contents of the registers may be placed in a system table.)[65]
- Read the integer from the data bus. The integer is an offset to the interrupt vector table. The vector table's instructions will then:

- Access the device-status table.
- Extract the process control block.
- Perform a context switch back to the writing process.

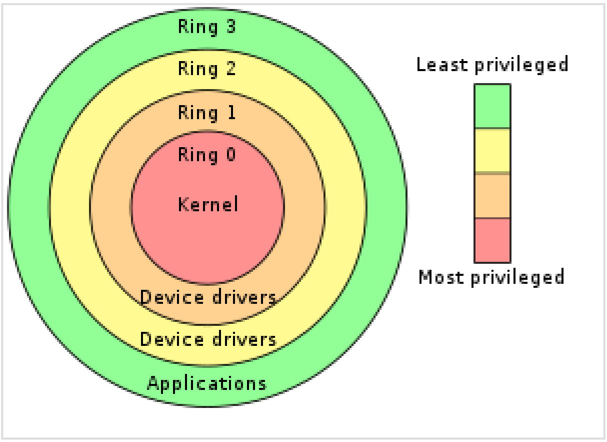
When the writing process has its time slice expired, the operating system will:[66]

- Pop from the call stack the registers other than the status register and program counter.
- Pop from the call stack the status register.
- Pop from the call stack the address of the next instruction, and set it back into the program counter.

With the program counter now reset, the interrupted process will resume its time slice.[44]

Modes

Modern computers support multiple modes of operation. CPUs with this capability offer at least two modes: user mode and supervisor mode. In general terms, supervisor mode operation allows unrestricted access to all machine resources, including all MPU instructions. User mode operation sets limits on instruction use and typically disallows direct access to machine resources. CPUs might have other modes similar to user mode as well, such as the virtual modes in order to emulate older processor types, such as 16-bit processors on a 32-bit one, or 32-bit processors on a 64-bit one.



Privilege rings for the x86 microprocessor architecture available in protected mode. Operating systems determine which processes run in each mode.

At power-on or reset, the system begins in supervisor mode. Once an operating system kernel has been loaded and started, the boundary between user mode and supervisor mode (also known as kernel mode) can be established.

Supervisor mode is used by the kernel for low level tasks that need unrestricted access to hardware, such as controlling how memory is accessed, and communicating with devices such as disk drives and video display devices. User mode, in contrast, is used for almost everything else. Application programs, such as word processors and database managers, operate within user mode, and can only access machine resources by turning control over to the kernel, a process which causes a switch to supervisor mode. Typically, the transfer of control to the kernel is achieved by executing a software interrupt instruction, such as the Motorola 68000 `TRAP` instruction. The software interrupt causes the processor to switch from user mode to supervisor mode and begin executing code that allows the kernel to take control.

In user mode, programs usually have access to a restricted set of processor instructions, and generally cannot execute any instructions that could potentially cause disruption to the system's operation. In supervisor mode, instruction execution restrictions are typically removed, allowing the kernel unrestricted access to all machine resources.

The term "user mode resource" generally refers to one or more CPU registers, which contain information that the running program is not allowed to alter. Attempts to alter these resources generally cause a switch to supervisor mode, where the operating system can deal with the illegal operation the program was attempting; for example, by forcibly terminating ("killing") the program.

Memory management

Among other things, a multiprogramming operating system kernel must be responsible for managing all system memory which is currently in use by the programs. This ensures that a program does not interfere with memory already in use by another program. Since programs time share, each program must have independent access to memory.

Cooperative memory management, used by many early operating systems, assumes that all programs make voluntary use of the kernel's memory manager, and do not exceed their allocated memory. This system of memory management is almost never seen any more, since programs often contain bugs which can cause them to exceed their allocated memory. If a program fails, it may cause memory used by one or more other programs to be affected or overwritten. Malicious programs or viruses may purposefully alter another program's memory, or may affect the operation of the operating system itself. With cooperative memory management, it takes only one misbehaved program to crash the system.

Memory protection enables the kernel to limit a process' access to the computer's memory. Various methods of memory protection exist, including memory segmentation and paging. All methods require some level of hardware support (such as the 80286 MMU), which does not exist in all computers.

In both segmentation and paging, certain protected mode registers specify to the CPU what memory address it should allow a running program to access. Attempts to access other addresses trigger an interrupt, which causes the CPU to re-enter supervisor mode, placing the kernel in charge. This is called a segmentation violation or Seg-V for short, and since it is both difficult to assign a meaningful result to such an operation, and because it is usually a sign of a misbehaving program, the kernel generally resorts to terminating the offending program, and reports the error.

Windows versions 3.1 through ME had some level of memory protection, but programs could easily circumvent the need to use it. A general protection fault would be produced, indicating a segmentation violation had occurred; however, the system would often crash anyway.

Virtual memory

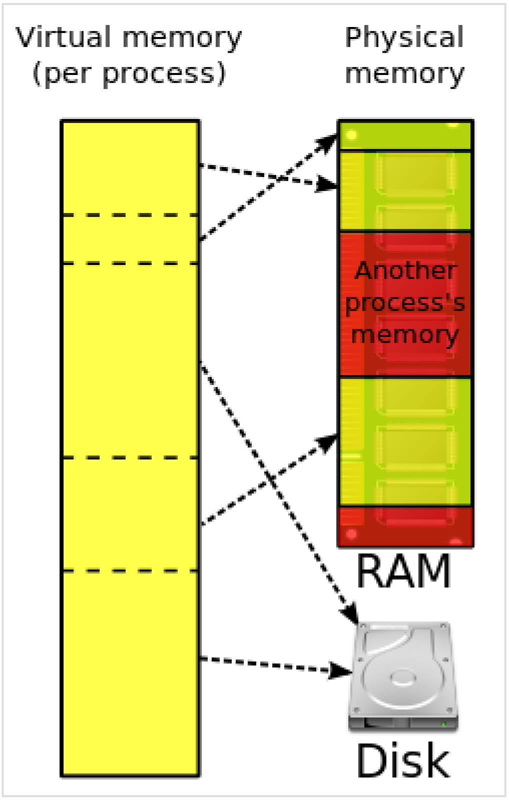
The use of virtual memory addressing (such as paging or segmentation) means that the kernel can choose what memory each program may use at any given time, allowing the operating system to use the same memory locations for multiple tasks.

If a program tries to access memory that is not in its current range of accessible memory, but nonetheless has been allocated to it, the kernel is interrupted in the same way as it would if the program were to exceed its allocated memory. (See section on memory management.) Under UNIX this kind of interrupt is referred to as a page fault.

When the kernel detects a page fault it generally adjusts the virtual memory range of the program which triggered it, granting it access to the memory requested. This gives the kernel discretionary power over where a particular application's memory is stored, or even whether or not it has actually been allocated yet.

In modern operating systems, memory which is accessed less frequently can be temporarily stored on a disk or other media to make that space available for use by other programs. This is called swapping, as an area of memory can be used by multiple programs, and what that memory area contains can be swapped or exchanged on demand.

"Virtual memory" provides the programmer or the user with the perception that there is a much larger amount of RAM in the computer than is really there.[67]



Many operating systems can "trick" programs into using memory scattered around the hard disk and RAM as if it is one continuous chunk of memory, called virtual memory.

Multitasking

Multitasking refers to the running of multiple independent computer programs on the same computer, giving the appearance that it is performing the tasks at the same time. Since most computers can do at most one or two things at one time, this is generally done via time-sharing, which means that each program uses a share of the computer's time to execute.

An operating system kernel contains a scheduling program which determines how much time each process spends executing, and in which order execution control should be passed to programs. Control is passed to a process by the kernel, which allows the program access to the CPU and memory. Later, control is returned to the kernel through some mechanism, so that another program may be allowed to use the CPU. This so-called passing of control between the kernel and applications is called a context switch.

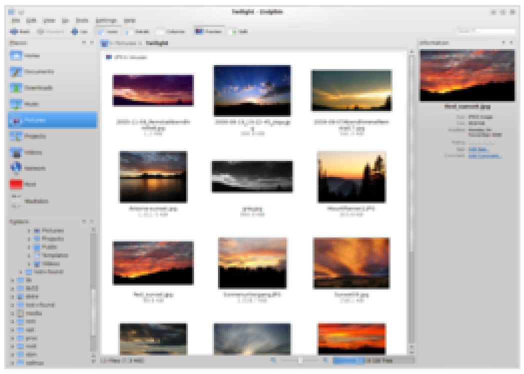
An early model which governed the allocation of time to programs was called cooperative multitasking. In this model, when control is passed to a program by the kernel, it may execute for as long as it wants before explicitly returning control to the kernel. This means that a malicious or malfunctioning program may not only prevent any other programs from using the CPU, but it can hang the entire system if it enters an infinite loop.

Modern operating systems extend the concepts of application preemption to device drivers and kernel code, so that the operating system has preemptive control over internal run-times as well.

The philosophy governing preemptive multitasking is that of ensuring that all programs are given regular time on the CPU. This implies that all programs must be limited in how much time they are allowed to spend on the CPU without being interrupted. To accomplish this, modern operating system kernels make use of a timed interrupt. A protected mode timer is set by the kernel which triggers a return to supervisor mode after the specified time has elapsed. (See above sections on Interrupts and Dual Mode Operation.)

On many single user operating systems cooperative multitasking is perfectly adequate, as home computers generally run a small number of well tested programs. AmigaOS is an exception, having preemptive multitasking from its first version. Windows NT was the first version of Microsoft Windows which enforced preemptive multitasking, but it did not reach the home user market until Windows XP (since Windows NT was targeted at professionals).

Disk access and file systems


File systems allow users and programs to organize and sort files on a computer, often through the use of directories (or "folders").

Access to data stored on disks is a central feature of all operating systems. Computers store data on disks using files, which are structured in specific ways in order to allow for faster access, higher reliability, and to make better use of the drive's available space. The specific way in which files are stored on a disk is called a file system, and enables files to have names and attributes. It also allows them to be stored in a hierarchy of directories or folders arranged in a directory tree.

Early operating systems generally supported a single type of disk drive and only one kind of file system. Early file systems were limited in their capacity, speed, and in the kinds of file names and directory structures they could use. These limitations often reflected limitations in the operating systems they were designed for, making it very difficult for an operating system to support more than one file system.

While many simpler operating systems support a limited range of options for accessing storage systems, operating systems like UNIX and Linux support a technology known as a virtual file system or VFS. An operating system such as UNIX supports a wide array of storage devices, regardless of their design or file systems, allowing them to be accessed through a common application programming interface (API). This makes it unnecessary for programs to have any knowledge about the device they are accessing. A VFS allows the operating system to provide programs with access to an unlimited number of devices with an infinite variety of file systems installed on them, through the use of specific device drivers and file system drivers.

A connected storage device, such as a hard drive, is accessed through a device driver. The device driver understands the specific language of the drive and is able to translate that language into a standard language used by the operating system to access all disk drives. On UNIX, this is the language of block devices.

When the kernel has an appropriate device driver in place, it can then access the contents of the disk drive in raw format, which may contain one or more file systems. A file system driver is used to translate the commands used to access each specific file system into a standard set of commands that the operating system can use to talk to all file systems. Programs can then deal with these file systems on the basis of filenames, and directories/folders, contained within a hierarchical structure. They can create, delete, open, and close files, as well as gather various information about them, including access permissions, size, free space, and creation and modification dates.

Various differences between file systems make supporting all file systems difficult. Allowed characters in file names, case sensitivity, and the presence of various kinds of file attributes makes the implementation of a single interface for every file system a daunting task. Operating systems tend to recommend using (and so support natively) file systems specifically designed for them; for example, NTFS in Windows and ReiserFS, Reiser4, ext3, ext4 and Btrfs in Linux. However, in practice, third party drivers are usually available to give support for the most widely used file systems in most general-purpose operating systems (for example, NTFS is available in Linux through NTFS-3g, and ext2/3 and ReiserFS are available in Windows through third-party software).

Support for file systems is highly varied among modern operating systems, although there are several common file systems which almost all operating systems include support and drivers for. Operating systems vary on file system support and on the disk formats they may be installed on. Under Windows, each file system is usually limited in application to certain media; for example, CDs must use ISO 9660 or UDF, and as of Windows Vista, NTFS is the only file system which the operating system can be installed on. It is possible to install Linux onto many types of file systems. Unlike other operating systems, Linux and UNIX allow any file system to be used regardless of the media it is stored in, whether it is a hard drive, a disc (CD, DVD...), a USB flash drive, or even contained within a file located on another file system.

Device drivers

A device driver is a specific type of computer software developed to allow interaction with hardware devices. Typically this constitutes an interface for communicating with the device, through the specific computer bus or communications subsystem that the hardware is connected to, providing commands to or receiving data from the device, and on the other end, the requisite interfaces to the operating system and software applications. It is a specialized hardware-dependent computer program which is also operating system specific that enables another program, typically an operating system or applications software package or computer program running under the operating system kernel, to interact transparently with a hardware device, and usually provides the requisite interrupt handling necessary for any necessary asynchronous time-dependent hardware interfacing needs.

The key design goal of device drivers is abstraction. Every model of hardware (even within the same class of device) is different. Newer models also are released by manufacturers that provide more reliable or better performance and these newer models are often controlled differently. Computers and their operating systems cannot be expected to know how to control every device, both now and in the future. To solve this problem, operating systems essentially dictate how every type of device should be controlled. The function of the device driver is then to translate these operating system mandated function calls into device specific calls. In theory a new device, which is controlled in a new manner, should function correctly if a suitable driver is available. This new driver ensures that the device appears to operate as usual from the operating system's point of view.

Under versions of Windows before Vista and versions of Linux before 2.6, all driver execution was co-operative, meaning that if a driver entered an infinite loop it would freeze the system. More recent revisions of these operating systems incorporate kernel preemption, where the kernel interrupts the driver to give it tasks, and then separates itself from the process until it receives a response from the device driver, or gives it more tasks to do.

Networking

Currently most operating systems support a variety of networking protocols, hardware, and applications for using them. This means that computers running dissimilar operating systems can participate in a common network for sharing resources such as computing, files, printers, and scanners using either wired or wireless connections. Networks can essentially allow a computer's operating system to access the resources of a remote computer to support the same functions as it could if those resources were connected directly to the local computer. This includes everything from simple communication, to using networked file systems or even sharing another computer's graphics or sound hardware. Some network services allow the resources of a computer to be accessed transparently, such as SSH which allows networked users direct access to a computer's command line interface.

Client/server networking allows a program on a computer, called a client, to connect via a network to another computer, called a server. Servers offer (or host) various services to other network computers and users. These services are usually provided through ports or numbered access points beyond the server's IP address. Each port number is usually associated with a maximum of one running program, which is responsible for handling requests to that port. A daemon, being a user program, can in turn access the local hardware resources of that computer by passing requests to the operating system kernel.

Many operating systems support one or more vendor-specific or open networking protocols as well, for example, SNA on IBM systems, DECnet on systems from Digital Equipment Corporation, and Microsoft-specific protocols (SMB) on Windows. Specific protocols for specific tasks may also be supported such as NFS for file access. Protocols like ESound, or esd can be easily extended over the network to provide sound from local applications, on a remote system's sound hardware.

Security

Security means protecting users from other users of the same computer, as well as from those who seeking remote access to it over a network.[68] Operating systems security rests on achieving the CIA triad: confidentiality (unauthorized users cannot access data), integrity (unauthorized users cannot modify data), and availability (ensuring that the system remains available to authorized users, even in the event of a denial of service attack).[69] As with other computer systems, isolating security domains—in the case of operating systems, the kernel, processes, and virtual machines—is key to achieving security.[70] Other ways to increase security include simplicity to minimize the attack surface, locking access to resources by default, checking all requests for authorization, principle of least authority (granting the minimum privilege essential for performing a task), privilege separation, and reducing shared data.[71]

Some operating system designs are more secure than others. Those with no isolation between the kernel and applications are least secure, while those with a monolithic kernel like most general-purpose operating systems are still vulnerable if any part of the kernel is compromised. A more secure design features microkernels that separate the kernel's privileges into many separate security domains and reduce the consequences of a single kernel breach.[72] Unikernels are another approach that improves security by minimizing the kernel and separating out other operating systems functionality by application.[72]

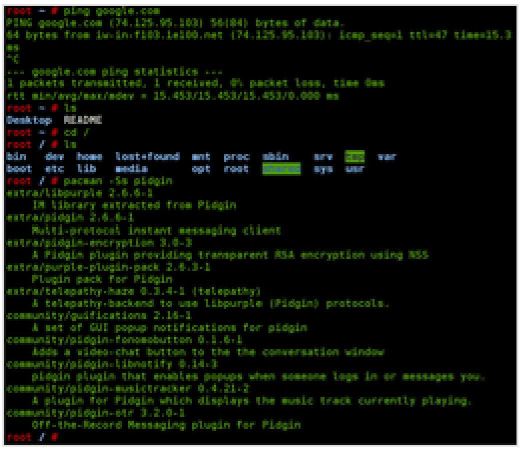
Most operating systems are written in C or C++, which can cause vulnerabilities. Despite various attempts to protect against them, a substantial number of vulnerabilities are caused by buffer overflow attacks, which are enabled by the lack of bounds checking.[73] Hardware vulnerabilities, some of them caused by CPU optimizations, can also be used to compromise the operating system.[74] Programmers coding the operating system may have deliberately implanted vulnerabilities, such as back doors.[75]

Operating systems security is hampered by their increasing complexity and the resulting inevitability of bugs.[76] Because formal verification of operating systems may not be feasible, operating systems developers use hardening to reduce vulnerabilities,[77] such as address space layout randomization, control-flow integrity,[78] access restrictions,[79] and other techniques.[80] Anyone can contribute code to open source operating systems, which have transparent code histories and distributed governance structures.[81]

Their developers work together to find and eliminate security vulnerabilities, using techniques such as code review and type checking to avoid malicious code.[82][83] Andrew S. Tanenbaum advises releasing the source code of all operating systems, arguing that it prevents the developer from falsely believing it is secret and relying on security by obscurity.[84]

User interface

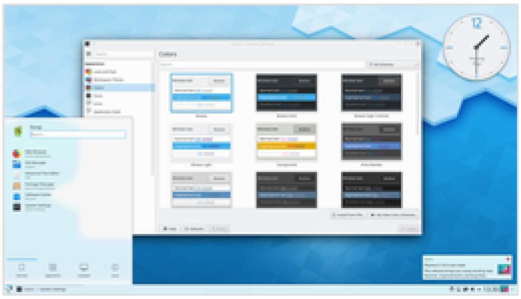
Every computer that is to be operated by an individual requires a user interface. The user interface is usually referred to as a shell and is essential if human interaction is to be supported. The user interface views the directory structure and requests services from the operating system that will acquire data from input hardware devices, such as a keyboard, mouse or credit card reader, and requests operating system services to display prompts, status messages and such on output hardware devices, such as a video monitor or printer. The two most common forms of a user interface have historically been the command-line interface, where computer commands are typed out line-by-line, and the graphical user interface, where a visual environment (most commonly a WIMP) is present.



```
root ~ # ping google.com
PING google.com (74.125.95.103) 56(84) bytes of data.
64 bytes from iw-in-f103.1e100.net (74.125.95.103): icmp_seq=1 ttl=47 time=15.3
ms
^C
--- google.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 15.453/15.453/15.453/0.000 ms
root ~ # ls
Desktop  README
root ~ # cd /
root / # ls
bin   dev  home  lost+found  mnt  proc  sbin    srv  tmp  var
boot  etc  lib   media       opt  root  shared  sys  usr
root / # pacman -Ss pidgin
extra/libpurple 2.6.6-1
    IM library extracted from Pidgin
extra/pidgin 2.6.6-1
    Multi-protocol instant messaging client
extra/pidgin-encryption 3.0-3
    A Pidgin plugin providing transparent RSA encryption using NSS
extra/purple-plugin-pack 2.6.3-1
    Plugin pack for Pidgin
extra/telepathy-haze 0.3.4-1 (telepathy)
    A telepathy-backend to use libpurple (Pidgin) protocols.
community/guifications 2.16-1
    A set of GUI popup notifications for pidgin
community/pidgin-fonomobutton 0.1.6-1
    Adds a video-chat button to the the conversation window
community/pidgin-libnotify 0.14-3
    pidgin plugin that enables popups when someone logs in or messages you.
community/pidgin-musictracker 0.4.21-2
    A plugin for Pidgin which displays the music track currently playing.
community/pidgin-otr 3.2.0-1
    Off-the-Record Messaging plugin for Pidgin
root / #
```

A screenshot of the bash command line. Each command is typed out after the 'prompt', and then its output appears below, working its way down the screen. The current command prompt is at the bottom.

Graphical user interfaces



A screenshot of the KDE Plasma 5 graphical user interface. Programs take the form of images on the screen, and the files, folders (directories), and applications take the form of icons and symbols. A mouse is used to navigate the computer.

Most of the modern computer systems support graphical user interfaces (GUI), and often include them. In some computer systems, such as the original implementation of the classic Mac OS, the GUI is integrated into the kernel.

While technically a graphical user interface is not an operating system service, incorporating support for one into the operating system kernel can allow the GUI to be more responsive by reducing the number of context switches required for the GUI to perform its output functions. Other operating systems are modular, separating the graphics subsystem from the kernel and the Operating System. In the 1980s UNIX, VMS and many others had operating systems that were built this way. Linux and macOS are also built this way. Modern releases of Microsoft Windows such as Windows Vista implement a graphics subsystem that is mostly in user-space; however the graphics drawing routines of versions between Windows NT 4.0 and Windows Server 2003 exist mostly in kernel space. Windows 9x had very little distinction between the interface and the kernel.

Many computer operating systems allow the user to install or create any user interface they desire. The X Window System in conjunction with GNOME or KDE Plasma 5 is a commonly found setup on most Unix and Unix-like (BSD, Linux, Solaris) systems. A number of Windows shell replacements have been released for Microsoft Windows, which offer alternatives to the included Windows shell, but the shell itself cannot be separated from Windows.

Numerous Unix-based GUIs have existed over time, most derived from X11. Competition among the various vendors of Unix (HP, IBM, Sun) led to much fragmentation, though an effort to standardize in the 1990s to COSE and CDE failed for various reasons, and were eventually eclipsed by the widespread adoption of GNOME and K Desktop Environment. Prior to free software-based toolkits and desktop environments, Motif was the prevalent toolkit/desktop combination (and was the basis upon which CDE was developed).

Graphical user interfaces evolve over time. For example, Windows has modified its user interface almost every time a new major version of Windows is released, and the Mac OS GUI changed dramatically with the introduction of Mac OS X in 1999.[85]

Real-time operating systems

A real-time operating system (RTOS) is an operating system intended for applications with fixed deadlines (real-time computing). Such applications include some small embedded systems, automobile engine controllers, industrial robots, spacecraft, industrial control, and some large-scale computing systems.

An early example of a large-scale real-time operating system was Transaction Processing Facility developed by American Airlines and IBM for the Sabre Airline Reservations System.

Embedded systems that have fixed deadlines use a real-time operating system such as VxWorks, PikeOS, eCos, QNX, MontaVista Linux and RTLinux. Windows CE is a real-time operating system that shares similar APIs to desktop Windows but shares none of desktop Windows' codebase.[86] Symbian OS also has an RTOS kernel (EKA2) starting with version 8.0b.

Some embedded systems use operating systems such as Palm OS, BSD, and Linux, although such operating systems do not support real-time computing.

Operating system development as a hobby

A hobby operating system may be classified as one whose code has not been directly derived from an existing operating system, and has few users and active developers.

In some cases, hobby development is in support of a "homebrew" computing device, for example, a simple single-board computer powered by a 6502 microprocessor. Or, development may be for an architecture already in widespread use. Operating system development may come from entirely new concepts, or may commence by modeling an existing operating system. In either case, the hobbyist is her/his own developer, or may interact with a small and sometimes unstructured group of individuals who have like interests.

Examples of a hobby operating system include Syllable and TempleOS.

Diversity of operating systems and portability

If an application is written for use on a specific operating system, and is ported to another OS, the functionality required by that application may be implemented differently by that OS (the names of functions, meaning of arguments, etc.) requiring the application to be adapted, changed, or otherwise maintained.

This cost in supporting operating systems diversity can be avoided by instead writing applications against software platforms such as Java or Qt. These abstractions have already borne the cost of adaptation to specific operating systems and their system libraries.

Another approach is for operating system vendors to adopt standards. For example, POSIX and OS abstraction layers provide commonalities that reduce porting costs.

Market share

See also

- Comparison of operating systems
- Crash (computing)
- DBOS
- Hypervisor
- Interruptible operating system
- List of important publications in operating systems
- List of operating systems
- List of pioneers in computer science
- Live CD
- Glossary of operating systems terms
- Microcontroller
- Mobile device
- Mobile operating system
- Network operating system
- Object-oriented operating system
- Operating System Projects
- System Commander
- System image
- Timeline of operating systems

Notes

a. A combination of DOS/360 and emulation software was known as Compatibility Operating System (COS).

b. However, ESPOL did allow source programs to specify all of the operations of the instruction repertoire.

c. Bell Labs quickly dropped out, leaving GE and MIT.

d. Modern CPUs provide instructions (e.g. SYSENTER) to invoke selected kernel services without an interrupts. Visit https://wiki.osdev.org/SYSENTER for more information.

e. Examples include SIGINT, SIGSEGV, and SIGBUS.

f. often in the form of a DMA chip for smaller systems and I/O channels for larger systems

g. Modern motherboards have a DMA controller. Additionally, a device may also have one. Visit SCSI RDMA Protocol.

References

1. Stallings (2005). *Operating Systems, Internals and Design Principles*. Pearson: Prentice Hall. p. 6.
2. Dhotre, I.A. (2009). *Operating Systems*. Technical Publications. p. 1.
3. "Desktop Operating System Market Share Worldwide" (https://gs.statcounter.com/os-market-share/desktop/worldwide/#monthly-202309-202309-bar). *StatCounter Global Stats*. Archived (https://web.archive.org/web/20231002223546/https://gs.statcounter.com/os-market-share/desktop/worldwide/) from the original on 2 October 2023. Retrieved 3 October 2023.
4. "Mobile & Tablet Operating System Market Share Worldwide" (https://gs.statcounter.com/os-market-share/mobile-tablet/worldwide/#monthly-202309-202309-bar). *StatCounter Global Stats*. Retrieved 2 October 2023.
5. "VII. Special-Purpose Systems - Operating System Concepts, Seventh Edition [Book]" (https://www.oreilly.com/library/view/operating-system-concepts/9780471694663/pt07.html). *www.oreilly.com*. Archived (https://web.archive.org/web/20210613190049/https://www.oreilly.com/library/view/operating-system-concepts/9780471694663/pt07.html) from the original on 13 June 2021. Retrieved 8 February 2021.
6. "Special-Purpose Operating Systems - RWTH AACHEN UNIVERSITY Institute for Automation of Complex Power Systems - English" (https://www.acs.eonerc.rwth-aachen.de/cms/E-ON-ERC-ACS/Studium/Lehrveranstaltungen/~lrhs/Spezial-Betriebssysteme/?lidx=1). *www.acs.eonerc.rwth-aachen.de*. Archived (https://web.archive.org/web/20210614034001/https://www.acs.eonerc.rwth-aachen.de/cms/E-ON-ERC-ACS/Studium/Lehrveranstaltungen/~lrhs/Spezial-Betriebssysteme/?lidx=1) from the original on 14 June 2021. Retrieved 8 February 2021.
7. Lorch, Jacob R.; Smith, Alan Jay (1996). "Reducing processor power consumption by improving processor time management in a single-user operating system". *Proceedings of the 2nd annual international conference on Mobile computing and networking*. New York, NY, US: ACM. pp. 143–154. doi:10.1145/236387.236437 (https://doi.org/10.1145%2F236387.236437). ISBN 089791872X.
8. Mishra, B.; Singh, N.; Singh, R. (2014). "Master-slave group based model for co-ordinator selection, an improvement of bully algorithm". *International Conference on Parallel, Distributed and Grid Computing (PDGC)*. pp. 457–460. doi:10.1109/PDGC.2014.7030789 (https://doi.org/10.1109%2FPDGC.2014.7030789). ISBN 978-1-4799-7682-9. S2CID 13887160 (https://api.semanticscholar.org/CorpusID:13887160).
9. Hansen, Per Brinch, ed. (2001). *Classic Operating Systems* (https://books.google.com/books?id=-PDPBvIPYBkC&pg=PP1). Springer. pp. 4–7. ISBN 0-387-95113-X. Archived (https://web.archive.org/web/20230111061716/https://books.google.com/books?id=-PDPBvIPYBkC&pg=PP1) from the original on 11 January 2023. Retrieved 19 December 2020.
10. Ryckman, George (1960). "The computer operation language" (https://doi.org/10.1145/1460361.1460406). *IRE-AIEE-ACM '60 (Western)*: 341. doi:10.1145/1460361.1460406 (https://doi.org/10.1145%2F1460361.1460406). S2CID 30745551 (https://api.semanticscholar.org/CorpusID:30745551).
11. Lavington, Simon (1998). *A History of Manchester Computers* (2nd ed.). Swindon: The British Computer Society. pp. 50–52. ISBN 978-1-902505-01-5.

12. Kilburn, T.; Payne, R. B.; Howarth, D. J. (December 1961). "The Atlas Supervisor" (https://dl.acm.org/doi/pdf/10.1145/1460764.1460786). *Institute of Electrical Engineers*: 279–294. doi:10.1145/1460764.1460786 (https://doi.org/10.1145%2F1460764.1460786). S2CID 16466990 (https://api.semanticscholar.org/CorpusID:16466990). Archived (https://web.archive.org/web/20220609221919/https://dl.acm.org/doi/pdf/10.1145/1460764.1460786) from the original on 9 June 2022. Retrieved 17 June 2022.
13. Brinch Hansen, Per (2000). *Classic Operating Systems: From Batch Processing to Distributed Systems*. Springer-Verlag.
14. *EXEC I UNIVAC 1107 Executive System* (http://bitsavers.org/pdf/univac/1107/UP-2577r1_EXEC_I_Tech.pdf) (PDF) (rev. 1 ed.). Sperry Rand. UP-2577. Archived (https://web.archive.org/web/20220615003346/http://bitsavers.org/pdf/univac/1107/UP-2577r1_EXEC_I_Tech.pdf) (PDF) from the original on 15 June 2022. Retrieved 15 May 2022.
15. *UNIVAC Data Processing Division Multi-processor System EXEC II Programmers Reference Manual* (http://bitsavers.org/pdf/univac/1100/exec/UP-4058_EXEC_II_Programmers_Ref_Man_1966.pdf) (PDF). Sperry Rand. 1966. UP-4058. Archived (https://web.archive.org/web/20220615003849/http://bitsavers.org/pdf/univac/1100/exec/UP-4058_EXEC_II_Programmers_Ref_Man_1966.pdf) (PDF) from the original on 15 June 2022. Retrieved 15 May 2022.
16. *UNIVAC 1108 Multi-processor System Operating System EXEC 8 Programmers Reference* (http://bitsavers.org/pdf/univac/1100/exec/UP-4144_Rev1_1108_Exec_8_Programmers_Reference_1968.pdf) (PDF) (rev. 1 ed.). Sperry Rand. 1968. UP-4144. Archived (https://web.archive.org/web/20220525145022/http://bitsavers.org/pdf/univac/1100/exec/UP-4144_Rev1_1108_Exec_8_Programmers_Reference_1968.pdf) (PDF) from the original on 25 May 2022. Retrieved 15 May 2022.
17. "Intel® Microprocessor Quick Reference Guide - Year" (http://www.intel.com/pressroom/kits/quickrefyr.htm#1985). *Intel*. Archived (https://web.archive.org/web/20160425001839/http://www.intel.com/pressroom/kits/quickrefyr.htm#1985) from the original on 25 April 2016. Retrieved 24 April 2016.
18. Ritchie, Dennis. "Unix Manual, first edition" (https://web.archive.org/web/20080518013206/http://cm.bell-labs.com/cm/cs/who/dmr/1stEdman.html). Lucent Technologies. Archived from the original (http://cm.bell-labs.com/cm/cs/who/dmr/1stEdman.html) on 18 May 2008. Retrieved 22 November 2012.
19. "OS X Mountain Lion – Move your Mac even further ahead" (https://www.apple.com/macosx/lion/). Apple. Archived (https://web.archive.org/web/20110523090205/http://www.apple.com/macosx/lion/) from the original on 23 May 2011. Retrieved 7 August 2012.
20. "Openedition Services on MVS/ESA SP Version 4 Release 3 Announced and Availability of MVS/ESA SP Version 4 Release 3 With Additional Enhancements" (https://www.ibm.com/docs/en/announcements/archive/ENUS293-060). *Announcement Letters*. IBM. 9 February 1993. 293-060. Retrieved 16 July 2023.
21. *Introducing OpenEdition MVS*. First Edition. IBM. December 1993. GC23-3010-00.
22. *OpenEdition MVS POSIX.1 Conformance Document*. First Edition. IBM. February 1993. GC23-3011-00.
23. *OpenEdition MVS POSIX.2 Conformance Document*. First Edition. IBM. December 1993. GC23-3012-00.
24. "Twenty Years of Linux according to Linus Torvalds" (https://www.zdnet.com/article/twenty-years-of-linux-according-to-linus-torvalds/). ZDNet. April 13, 2011. Archived (https://web.archive.org/web/20160919232940/http://www.zdnet.com/article/twenty-years-of-linux-according-to-linus-torvalds/) from the original on September 19, 2016. Retrieved September 19, 2016.

25. Linus Benedict Torvalds (5 October 1991). "Free minix-like kernel sources for 386-AT" (https://groups.google.com/group/comp.os.minix/msg/2194d253268b0a1b?pli=1). Newsgroup: comp.os.minix (news:comp.os.minix). Archived (https://web.archive.org/web/20130302010902/http://groups.google.com/group/comp.os.minix/msg/2194d253268b0a1b?pli=1) from the original on 2 March 2013. Retrieved 30 September 2011.
26. "What Is Linux: An Overview of the Linux Operating System" (https://medium.com/@theinfovalley097/what-is-linux-an-overview-of-the-linux-operating-system-77bc7421c7e5). Medium. 11 April 2020. Retrieved 16 July 2023.
27. Linux Online (2008). "Linux Logos and Mascots" (https://web.archive.org/web/20100815085106/http://www.linux.org/info/logos.html). Archived from the original (http://www.linux.org/info/logos.html) on 15 August 2010. Retrieved 11 August 2009.
28. "IDC report into Server market share" (https://web.archive.org/web/20120927155332/http://www.idc.com/about/viewpressrelease.jsp?containerId=prUS22360110&sectionId=null&elementId=null&pageType=SYNOPSIS). Idc.com. Archived from the original (http://www.idc.com/about/viewpressrelease.jsp?containerId=prUS22360110&sectionId=null&elementId=null&pageType=SYNOPSIS) on 27 September 2012. Retrieved 7 August 2012.
29. LinuxDevices Staff (23 April 2008). "Linux still top embedded OS" (https://web.archive.org/web/20160419010154/http://archive.linuxgizmos.com/linux-still-top-embedded-os/). *LinuxGizmos.com*. Archived from the original (http://archive.linuxgizmos.com/linux-still-top-embedded-os/) on 19 April 2016. Retrieved 5 April 2016.
30. "Operating system Family / Linux | TOP500" (https://www.top500.org/statistics/details/osfam/1/). *www.top500.org*. Retrieved 30 July 2023.
31. "Operating System Market Share Worldwide Jan–Dec 2022" (https://gs.statcounter.com/os-market-share/all/worldwide/2022). 2022. Retrieved 4 November 2023.
32. "Desktop Operating System Market Share Worldwide Jan–Dec 2022" (https://gs.statcounter.com/os-market-share/desktop/worldwide/2022). 2022. Retrieved 4 November 2023.
33. "Troubleshooting MS-DOS Compatibility Mode on Hard Disks" (https://web.archive.org/web/20120810073641/http://support.microsoft.com/kb/130179/EN-US). Microsoft Support. Archived from the original (http://support.microsoft.com/kb/130179/EN-US) on 10 August 2012. Retrieved 7 August 2012.
34. "Using NDIS 2 PCMCIA Network Card Drivers in Windows 95" (https://web.archive.org/web/20130217043405/http://support.microsoft.com/kb/134748/en). Microsoft Support. Archived from the original (http://support.microsoft.com/kb/134748/en) on 17 February 2013. Retrieved 7 August 2012.
35. "INFO: Windows 95 Multimedia Wave Device Drivers Must be 16 bit" (https://web.archive.org/web/20130217043412/http://support.microsoft.com/kb/163354/en). Microsoft Support. Archived from the original (http://support.microsoft.com/kb/163354/en) on 17 February 2013. Retrieved 7 August 2012.
36. Arthur, Charles (5 January 2011). " 'Windows 8' will run on ARM chips - but third-party apps will need rewrite" (https://www.theguardian.com/technology/2011/jan/05/microsoft-windows-8-arm-processors). *The Guardian*. Archived (https://web.archive.org/web/20161012022725/https://www.theguardian.com/technology/2011/jan/05/microsoft-windows-8-arm-processors) from the original on 12 October 2016.
37. "Operating System Share by Groups for Sites in All Locations January 2009" (https://web.archive.org/web/20090706135203/http://news.netcraft.com/SSL-Survey/CMatch/osdv_all). Archived from the original (http://news.netcraft.com/SSL-Survey/CMatch/osdv_all) on 6 July 2009. Retrieved 3 May 2010.
38. "Behind the IDC data: Windows still No. 1 in server operating systems" (http://blogs.zdnet.com/microsoft/?p=5408). *ZDNet*. 26 February 2010. Archived (https://web.archive.org/web/20100301032456/http://blogs.zdnet.com/microsoft/?p=5408) from the original on 1 March 2010.

39. Kerrisk, Michael (2010). *The Linux Programming Interface*. No Starch Press. p. 388. ISBN 978-1-59327-220-3. "A signal is a notification to a process that an event has occurred. Signals are sometimes described as software interrupts."
40. Hyde, Randall (1996). "Chapter Seventeen: Interrupts, Traps and Exceptions (Part 1)" (https://www.plantation-productions.com/Webster/www.artofasm.com/DOS/ch17/CH17-1.html#HEADING1-0). *The Art Of Assembly Language Programming*. No Starch Press. Archived (https://web.archive.org/web/20211222205623/https://www.plantation-productions.com/Webster/www.artofasm.com/DOS/ch17/CH17-1.html#HEADING1-0) from the original on 22 December 2021. Retrieved 22 December 2021. "The concept of an interrupt is something that has expanded in scope over the years. The 80x86 family has only added to the confusion surrounding interrupts by introducing the int (software interrupt) instruction. Indeed, different manufacturers have used terms like exceptions, faults, aborts, traps and interrupts to describe the phenomena this chapter discusses. Unfortunately there is no clear consensus as to the exact meaning of these terms. Different authors adopt different terms to their own use."
41. Tanenbaum, Andrew S. (1990). *Structured Computer Organization, Third Edition* (https://archive.org/details/structuredcomput00tane/page/308). Prentice Hall. p. 308 (https://archive.org/details/structuredcomput00tane/page/308). ISBN 978-0-13-854662-5. "Like the trap, the interrupt stops the running program and transfers control to an interrupt handler, which performs some appropriate action. When finished, the interrupt handler returns control to the interrupted program."
42. Silberschatz, Abraham (1994). *Operating System Concepts, Fourth Edition*. Addison-Wesley. p. 32. ISBN 978-0-201-50480-4. "When an interrupt (or trap) occurs, the hardware transfers control to the operating system. First, the operating system preserves the state of the CPU by storing registers and the program counter. Then, it determines which type of interrupt has occurred. For each type of interrupt, separate segments of code in the operating system determine what action should be taken."
43. Silberschatz, Abraham (1994). *Operating System Concepts, Fourth Edition*. Addison-Wesley. p. 105. ISBN 978-0-201-50480-4. "Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. This task is known as a context switch."
44. Silberschatz, Abraham (1994). *Operating System Concepts, Fourth Edition*. Addison-Wesley. p. 31. ISBN 978-0-201-50480-4.
45. Silberschatz, Abraham (1994). *Operating System Concepts, Fourth Edition*. Addison-Wesley. p. 30. ISBN 978-0-201-50480-4. "Hardware may trigger an interrupt at any time by sending a signal to the CPU, usually by way of the system bus."
46. Kerrisk, Michael (2010). *The Linux Programming Interface*. No Starch Press. p. 388. ISBN 978-1-59327-220-3. "Signals are analogous to hardware interrupts in that they interrupt the normal flow of execution of a program; in most cases, it is not possible to predict exactly when a signal will arrive."
47. Kerrisk, Michael (2010). *The Linux Programming Interface*. No Starch Press. p. 388. ISBN 978-1-59327-220-3. "Among the types of events that cause the kernel to generate a signal for a process are the following: A software event occurred. For example, ... the process's CPU time limit was exceeded[.]"
48. Kerrisk, Michael (2010). *The Linux Programming Interface*. No Starch Press. p. 388. ISBN 978-1-59327-220-3.

49. "Intel® 64 and IA-32 Architectures Software Developer's Manual" (https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.pdf) (PDF). Intel Corporation. September 2016. p. 610. Archived (https://web.archive.org/web/20220323231921/https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.pdf) (PDF) from the original on 23 March 2022. Retrieved 5 May 2022.
50. Bach, Maurice J. (1986). *The Design of the UNIX Operating System*. Prentice-Hall. p. 200. ISBN 0-13-201799-7.
51. Kerrisk, Michael (2010). *The Linux Programming Interface*. No Starch Press. p. 400. ISBN 978-1-59327-220-3.
52. Tanenbaum, Andrew S. (1990). *Structured Computer Organization, Third Edition* (https://archive.org/details/structuredcomput00tane/page/308). Prentice Hall. p. 308 (https://archive.org/details/structuredcomput00tane/page/308). ISBN 978-0-13-854662-5.
53. Silberschatz, Abraham (1994). *Operating System Concepts, Fourth Edition*. Addison-Wesley. p. 182. ISBN 978-0-201-50480-4.
54. Haviland, Keith; Salama, Ben (1987). *UNIX System Programming*. Addison-Wesley Publishing Company. p. 153. ISBN 0-201-12919-1.
55. Haviland, Keith; Salama, Ben (1987). *UNIX System Programming*. Addison-Wesley Publishing Company. p. 148. ISBN 0-201-12919-1.
56. Haviland, Keith; Salama, Ben (1987). *UNIX System Programming*. Addison-Wesley Publishing Company. p. 149. ISBN 0-201-12919-1.
57. Tanenbaum, Andrew S. (1990). *Structured Computer Organization, Third Edition* (https://archive.org/details/structuredcomput00tane/page/292). Prentice Hall. p. 292 (https://archive.org/details/structuredcomput00tane/page/292). ISBN 978-0-13-854662-5.
58. IBM (September 1968), "Main Storage" (http://bitsavers.org/pdf/ibm/360/princOps/A22-6821-7_360PrincOpsDec67.pdf#page=8) (PDF), *IBM System/360 Principles of Operation* (http://bitsavers.org/pdf/ibm/360/princOps/A22-6821-7_360PrincOpsDec67.pdf) (PDF), Eighth Edition, p. 7, archived (https://web.archive.org/web/20220319083255/http://bitsavers.org/pdf/ibm/360/princOps/A22-6821-7_360PrincOpsDec67.pdf) (PDF) from the original on 19 March 2022, retrieved 13 April 2022
59. Tanenbaum, Andrew S. (1990). *Structured Computer Organization, Third Edition* (https://archive.org/details/structuredcomput00tane/page/294). Prentice Hall. p. 294 (https://archive.org/details/structuredcomput00tane/page/294). ISBN 978-0-13-854662-5.
60. "Program Interrupt Controller (PIC)" (http://bitsavers.org/pdf/dec/pdp7/F-75_PDP-7userHbk_Jun65.pdf#page=62) (PDF). *Users Handbook - PDP-7* (http://bitsavers.org/pdf/dec/pdp7/F-75_PDP-7userHbk_Jun65.pdf) (PDF). Digital Equipment Corporation. 1965. pp. 48 (http://bitsavers.org/pdf/dec/pdp7/F-75_PDP-7userHbk_Jun65.pdf#page=63). F-75. Archived (https://web.archive.org/web/20220510164742/http://bitsavers.org/pdf/dec/pdp7/F-75_PDP-7userHbk_Jun65.pdf) (PDF) from the original on 10 May 2022. Retrieved 20 April 2022.
61. *PDP-1 Input-Output Systems Manual* (http://bitsavers.org/pdf/dec/pdp1/F25_PDP1_IO.pdf) (PDF). Digital Equipment Corporation. pp. 19–20. Archived (https://web.archive.org/web/20190125050839/http://bitsavers.org/pdf/dec/pdp1/F25_PDP1_IO.pdf) (PDF) from the original on 25 January 2019. Retrieved 16 August 2022.
62. Silberschatz, Abraham (1994). *Operating System Concepts, Fourth Edition*. Addison-Wesley. p. 32. ISBN 978-0-201-50480-4.
63. Silberschatz, Abraham (1994). *Operating System Concepts, Fourth Edition*. Addison-Wesley. p. 34. ISBN 978-0-201-50480-4.

64. Tanenbaum, Andrew S. (1990). *Structured Computer Organization, Third Edition* (https://archive.org/details/structuredcomput00tane/page/295). Prentice Hall. p. 295 (https://archive.org/details/structuredcomput00tane/page/295). ISBN 978-0-13-854662-5.
65. Tanenbaum, Andrew S. (1990). *Structured Computer Organization, Third Edition* (https://archive.org/details/structuredcomput00tane/page/309). Prentice Hall. p. 309 (https://archive.org/details/structuredcomput00tane/page/309). ISBN 978-0-13-854662-5.
66. Tanenbaum, Andrew S. (1990). *Structured Computer Organization, Third Edition* (https://archive.org/details/structuredcomput00tane/page/310). Prentice Hall. p. 310 (https://archive.org/details/structuredcomput00tane/page/310). ISBN 978-0-13-854662-5.
67. Stallings, William (2008). *Computer Organization & Architecture*. New Delhi: Prentice-Hall of India Private Limited. p. 267. ISBN 978-81-203-2962-1.
68. Tanenbaum & Bos 2023, pp. 605–606.
69. Tanenbaum & Bos 2023, p. 608.
70. Tanenbaum & Bos 2023, p. 609.
71. Tanenbaum & Bos 2023, pp. 609–610.
72. Tanenbaum & Bos 2023, p. 612.
73. Tanenbaum & Bos 2023, pp. 648, 657.
74. Tanenbaum & Bos 2023, pp. 668–669, 674.
75. Tanenbaum & Bos 2023, pp. 679–680.
76. Tanenbaum & Bos 2023, pp. 605, 617–618.
77. Tanenbaum & Bos 2023, pp. 681–682.
78. Tanenbaum & Bos 2023, p. 683.
79. Tanenbaum & Bos 2023, p. 685.
80. Tanenbaum & Bos 2023, p. 689.
81. Richet & Bouaynaya 2023, p. 92.
82. Richet & Bouaynaya 2023, pp. 92–93.
83. Berntsso, Strandén & Warg 2017, pp. 130–131.
84. Tanenbaum & Bos 2023, p. 611.
85. Polsson, Ken (8 February 2007). "Chronology of Personal Computer Software (1998-1999)" (https://web.archive.org/web/20080514022217/http://www.islandnet.com/~kpolsson/compsoft/soft1998.htm). *islandnet.com*. Archived from the original (http://www.islandnet.com/~kpolsson/compsoft/soft1998.htm) on 14 May 2008.
86. "Reading: Operating System" (https://courses.lumenlearning.com/sanjacinto-computerapps-v2/chapter/reading-operating-system/). *Lumen*. Archived (https://web.archive.org/web/20190106012248/https://courses.lumenlearning.com/sanjacinto-computerapps-v2/chapter/reading-operating-system/) from the original on 6 January 2019. Retrieved 5 January 2019.

Further reading

- Anderson, Thomas; Dahlin, Michael (2014). *Operating Systems: Principles and Practice*. Recursive Books. ISBN 978-0-9856735-2-9.
- Auslander, M. A.; Larkin, D. C.; Scherr, A. L. (September 1981). "The Evolution of the MVS Operating System". *IBM Journal of Research and Development*. **25** (5): 471–482. doi:10.1147/rd.255.0471 (https://doi.org/10.1147%2Frd.255.0471).
- Berntsson, Petter Sainio; Strandén, Lars; Warg, Fredrik (2017). *Evaluation of Open Source Operating Systems for Safety-Critical Applications*. Springer International Publishing. pp. 117–132. ISBN 978-3-319-65948-0.

- Deitel, Harvey M.; Deitel, Paul; Choffnes, David (25 December 2015). *Operating Systems* (https://archive.org/details/modernoperatings00tane). Pearson/Prentice Hall. ISBN 978-0-13-092641-8.
- Bic, Lubomur F.; Shaw, Alan C. (2003). *Operating Systems*. Pearson: Prentice Hall.
- Silberschatz, Avi; Galvin, Peter; Gagne, Greg (2008). *Operating Systems Concepts*. John Wiley & Sons. ISBN 978-0-470-12872-5.
- O'Brien, J. A., & Marakas, G. M.(2011). *Management Information Systems*. 10e. McGraw-Hill Irwin.
- Leva, Alberto; Maggio, Martina; Papadopoulos, Alessandro Vittorio; Terraneo, Federico (2013). *Control-based Operating System Design*. IET. ISBN 978-1-84919-609-3.
- Arpaci-Dusseau, Remzi; Arpaci-Dusseau, Andrea (2015). *Operating Systems: Three Easy Pieces* (http://pages.cs.wisc.edu/~remzi/OSTEP/). Archived (https://web.archive.org/web/20160725012948/http://pages.cs.wisc.edu/~remzi/OSTEP/) from the original on 25 July 2016. Retrieved 25 July 2016.
- Richet, Jean-Loup; Bouaynaya, Wafa (2023). "Understanding and Managing Complex Software Vulnerabilities: An Empirical Analysis of Open-Source Operating Systems" (https://www.cairn.info/revue-systemes-d-information-et-management-2023-1-page-87.htm.). *Systèmes d'information & management*. **28** (1): 87–114. doi:10.54695/sim.28.1.0087 (https://doi.org/10.54695%2Fsim.28.1.0087).

External links

- Operating Systems (https://curlie.org/Computers/Software/Operating_Systems) at Curlie
- Multics History (http://www.cbi.umn.edu/iterations/haigh.html) and the history of operating systems

-