WIKIPEDIA
The Free Encyclopedia

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.

As of September 2024, Android is the most popular operating system with a 46% market share, followed by Microsoft Windows at 26%, iOS and iPadOS at 18%, macOS at 5%, and Linux at 1%. Android, iOS, and iPadOS are mobile operating systems, while Windows, macOS, and Linux are desktop operating systems.[3] Linux distributions are dominant in the server and supercomputing sectors. Other specialized classes of operating systems (special-purpose operating systems),[4][5] 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. a LiveUSB from a USB stick).

Definition and purpose

An operating system is difficult to define,[6] but has been called "the layer of software that manages a computer's resources for its users and their applications".[7] Operating systems include the software that is always running, called a kernel—but can include other software as well.[6][8] The two other types of programs that can run on a computer are system programs—which are associated with the operating system, but may not be part of the kernel—and applications—all other software.[8]

There are three main purposes that an operating system fulfills:[9]

- Operating systems allocate resources between different applications, deciding when they will receive central processing unit (CPU) time or space in memory.[9] On modern personal computers, users often want to run several applications at once. In order to ensure that one program cannot monopolize the computer's limited hardware resources, the operating system gives each application a share of the resource, either in time (CPU) or space (memory).[10][11] The operating system also must isolate applications from each other to protect them from errors and security vulnerabilities in another application's code, but enable communications between different applications.[12]
- Operating systems provide an interface that abstracts the details of accessing hardware details (such as physical memory) to make things easier for programmers.[9][13] Virtualization also enables the operating system to mask limited hardware resources; for example, virtual memory can provide a program with the illusion of nearly unlimited memory that exceeds the computer's actual memory.[14]
- Operating systems provide common services, such as an interface for accessing network and disk devices. This enables an application to be run on different hardware without needing to be rewritten.[15] Which services to include in an operating system varies greatly, and this functionality makes up the great majority of code for most operating systems.[16]

Types of operating systems

Multicomputer operating systems

With multiprocessors multiple CPUs share memory. A multicomputer or cluster computer has multiple CPUs, each of which has its own memory. Multicomputers were developed because large multiprocessors are difficult to engineer and prohibitively expensive;[17] they are universal in cloud computing because of the size of the machine needed.[18] The different CPUs often need to send and receive messages to each other;[19] to ensure good performance, the operating systems for these machines need to minimize this copying of packets.[20] Newer systems are often multiqueue—separating groups of users into separate queues—to reduce the need for packet copying and support more concurrent users.[21] Another technique is remote direct memory access, which enables each CPU to access memory belonging to other CPUs.[19] Multicomputer operating systems often support remote procedure calls where a CPU can call a procedure on another CPU,[22] or distributed shared memory, in which the operating system uses virtualization to generate shared memory that does not physically exist.[23]

Distributed systems

A distributed system is a group of distinct, networked computers—each of which might have their own operating system and file system. Unlike multicomputers, they may be dispersed anywhere in the world.[24] Middleware, an additional software layer between the operating system and applications, is often used to improve consistency. Although it functions similarly to an operating system, it is not a true operating system.[25]

Embedded

Embedded operating systems are designed to be used in embedded computer systems, whether they are internet of things objects or not connected to a network. Embedded systems include many household appliances. The distinguishing factor is that they do not load user-installed software. Consequently, they do not need protection between different applications, enabling simpler designs. Very small operating systems might run in less than 10 kilobytes,[26] and the smallest are for smart cards.[27] Examples include Embedded Linux, QNX, VxWorks, and the extra-small systems RIOT and TinyOS.[28]

Real-time

A real-time operating system is an operating system that guarantees to process events or data by or at a specific moment in time. Hard real-time systems require exact timing and are common in manufacturing, avionics, military, and other similar uses.[28] With soft real-time systems, the occasional missed event is acceptable; this category often includes audio or multimedia systems, as well as smartphones.[28] In order for hard real-time systems be sufficiently exact in their timing, often they are just a library with no protection between applications, such as eCos.[28]

Hypervisor

A hypervisor is an operating system that runs a virtual machine. The virtual machine is unaware that it is an application and operates as if it had its own hardware.[14][29] Virtual machines can be paused, saved, and resumed, making them useful for operating systems research, development,[30] and debugging.[31] They also enhance portability by enabling applications to be run on a computer even if they are not compatible with the base operating system.[14]

Library

A *library operating system* (libOS) 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 a single application and configuration code to construct a unikernel: [32] a specialized (only the absolute necessary pieces of code are extracted from libraries and bound together [33]), single address space, machine image that can be deployed to cloud or embedded environments.

The operating system code and application code are not executed in separated protection domains (there is only a single application running, at least conceptually, so there is no need to prevent interference between applications) and OS services are accessed via simple library calls (potentially inlining them based on compiler thresholds), without the usual overhead of context switches, [34] in a way similarly to embedded and real-time OSes. Note that this overhead is not negligible: to the direct cost of mode switching it's necessary to add the indirect pollution of important processor structures (like CPU caches, the instruction pipeline, and so on) which affects both user-mode and kernel-mode performance. [35]

History

The first computers in the late 1940s and 1950s were directly programmed either with plugboards or with machine code inputted on media such as punch cards, without programming languages or operating systems.[36] After the introduction of the transistor in the mid-1950s, mainframes began to be built. These still needed professional operators[36] who manually do what a modern operating system would do, such as scheduling programs to run,[37] but mainframes still had rudimentary operating systems such as Fortran Monitor System (FMS) and IBSYS.[38] In the 1960s, IBM introduced the first series of intercompatible computers (System/360). All of them ran the same operating system—OS/360—which consisted of millions of lines of assembly language that had thousands of bugs. The OS/360 also was the first popular operating system to support multiprogramming, such that the CPU could be put to use on one job while another was waiting on input/output (I/O). Holding multiple jobs in memory necessitated memory partitioning and safeguards against one job accessing the memory allocated to a different one.[39]

Around the same time, teleprinters began to be used as terminals so multiple users could access the computer simultaneously. The operating system MULTICS was intended to allow hundreds of users to access a large computer. Despite its limited adoption, it can be considered the precursor to cloud computing. The UNIX operating system originated as a development of MULTICS for a single user.[40] Because UNIX's source code was available, it became the basis of other, incompatible operating systems, of which the most successful were AT&T's System V and the University of California's Berkeley Software Distribution (BSD).[41] To increase compatibility, the IEEE released the POSIX standard for operating system application programming interfaces (APIs), which is supported by most UNIX systems. MINIX was a stripped-down version of UNIX, developed in 1987 for educational uses, that inspired the commercially available, free software Linux. Since 2008, MINIX is used in controllers of most Intel microchips, while Linux is widespread in data centers and Android smartphones.[42]

Microcomputers

The invention of large scale integration enabled the production of personal computers (initially called microcomputers) from around 1980.[43] For around five years, the CP/M (Control Program for Microcomputers) was the most popular operating system for microcomputers.[44] Later, IBM bought the DOS (Disk Operating System) from Microsoft. After modifications requested by IBM, the resulting system was called MS-DOS (MicroSoft Disk Operating System) and was widely used on IBM microcomputers. Later versions increased their sophistication, in part by borrowing features from UNIX.[44]

Apple's Macintosh was the first popular computer to use a graphical user interface (GUI). The GUI proved much more user friendly than the text-only command-line interface earlier operating systems had used. Following the success of Macintosh, MS-DOS was updated with a GUI overlay called Windows. Windows later was rewritten as a stand-alone operating system, borrowing so many features from another (VAX VMS) that a large legal settlement was paid.[45] In the twenty-first century, Windows continues to be popular on personal computers but has less market share of servers. UNIX operating systems, especially Linux, are the most popular on enterprise systems and servers but are also used on mobile devices and many other computer systems.[46]

On mobile devices, Symbian OS was dominant at first, being usurped by BlackBerry OS (introduced 2002) and iOS for iPhones (from 2007). Later on, the open-source Android operating system (introduced 2008), with a Linux kernel and a C library (Bionic) partially based on BSD code, became most popular.[47]

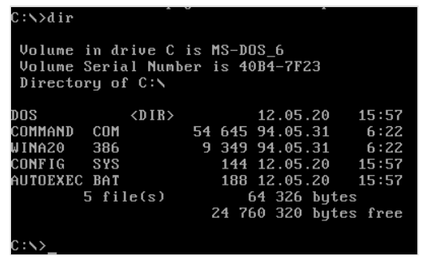
IBM System/360 Model 50 operator's console and CPU; the operator's console is a terminal used by the operating system to communicate with the operator.

Components

The components of an operating system are designed to ensure that various parts of a computer function cohesively. With the de facto obsoletion of DOS, all user software must interact with the operating system to access hardware.

Kernel

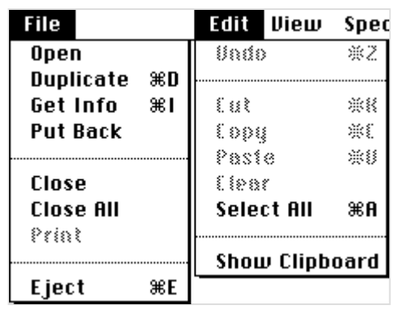
The kernel is the part of the operating system that provides protection between different applications and users. This protection is key to improving reliability by keeping errors isolated to one program, as well as security by limiting the power of malicious software and protecting private data, and ensuring that one program cannot monopolize the computer's resources.[48] Most operating systems have two modes of operation:[49] in user mode, the hardware checks that the software is only executing legal instructions, whereas the kernel has unrestricted powers and is not subject to these checks.[50] The kernel also manages memory for other processes and controls access to input/output devices.[51]



Command-line interface of the MS-DOS operating system

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.

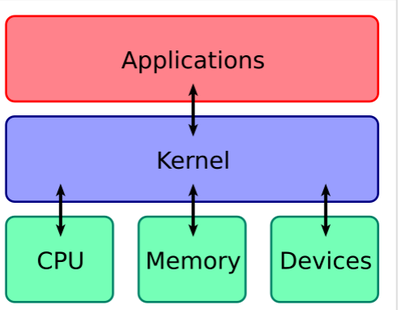


Graphical user interface of a Macintosh

Interrupts

An interrupt (also known as an abort, exception, *fault*, signal,[52] or *trap*)[53] 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).[54][55] An interrupt service routine may cause the central processing unit (CPU) to have a context switch.[56][a] 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.[57] However, several interrupt functions are common.[57] The architecture and operating system must:[57]

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.



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

Software interrupt

A software interrupt is a message to a process that an event has occurred.[52] This contrasts with a *hardware interrupt* — which is a message to the central processing unit (CPU) that an event has occurred.[58] Software interrupts are similar to hardware interrupts — there is a change away from the currently running process.[59] 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.[60] 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.[61]

Software interrupts may be error conditions, such as a malformed machine instruction.[61] However, the most common error conditions are division by zero and accessing an invalid memory address.[61]

Users can send messages to the kernel to modify the behavior of a currently running process.[61] For example, in the command-line environment, pressing the *interrupt character* (usually Control-C) might terminate the currently running process.[61]

To generate *software interrupts* for x86 CPUs, the INT assembly language instruction is available.[62] 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.[63] `pid` is the process identifier of the receiving process. `signum` is the signal number (in mnemonic format)[b] to be sent. (The abrasive name of `kill` was chosen because early implementations only terminated the process.)[64]

In Unix-like operating systems, *signals* inform processes of the occurrence of asynchronous events.[63] To communicate asynchronously, interrupts are required.[65] One reason a process needs to asynchronously communicate to another process solves a variation of the classic reader/writer problem.[66] The writer receives a pipe from the shell for its output to be sent to the reader's input stream.[67] 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.[68] `bravo` will then be moved to the ready queue and soon will read from its input stream.[69] The kernel will generate *software interrupts* to coordinate the piping.[69]

Signals may be classified into 7 categories.[63] 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.[70]

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.[71] (Separate from the architecture, a device may perform direct memory access[c] to and from main memory either directly or via a bus.)[72][d]

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[72] or word[73] 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.[74]

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.[75]
- Create an entry in the device-status table.[76] 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.[65]
- Set the memory address of the memory buffer to a predetermined device register.[77]
- Set the buffer size (an integer) to another predetermined register.[77]
- 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.[78] 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.[57]
- Push the contents of the other registers onto the call stack. (Alternatively, the contents of the registers may be placed in a system table.)[78]
- 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:[79]

- 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.[57]

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 anymore, 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

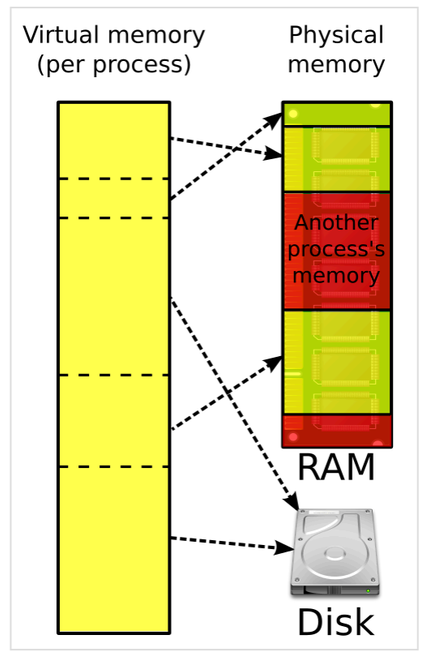
The use of virtual memory addressing (such as paging or segmentation) means that the kernel can choose what memory each program may use at any given time, allowing the operating system to use the same memory locations for multiple tasks.

If a program tries to access memory that is not accessible[e] memory, but nonetheless has been allocated to it, the kernel is interrupted . This kind of interrupt is typically 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 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.[80]



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.

Concurrency

Concurrency refers to the operating system's ability to carry out multiple tasks simultaneously.[81] Virtually all modern operating systems support concurrency.[82]

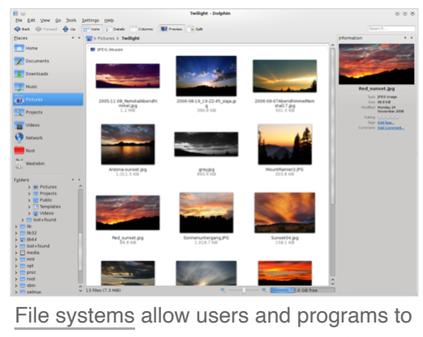
Threads enable splitting a process' work into multiple parts that can run simultaneously.[83] The number of threads is not limited by the number of processors available. If there are more threads than processors, the operating system kernel schedules, suspends, and resumes threads, controlling when each thread runs and how much CPU time it receives.[84] During a context switch a running thread is suspended, its state is saved into the thread control block and stack, and the state of the new thread is loaded in.[85] Historically, on many systems a thread could run until it relinquished control (cooperative multitasking). Because this model can allow a single thread to monopolize the processor, most operating systems now can interrupt a thread (preemptive multitasking).[86]

Threads have their own thread ID, program counter (PC), a register set, and a stack, but share code, heap data, and other resources with other threads of the same process.[87][88] Thus, there is less overhead to create a thread than a new process.[89] On single-CPU systems, concurrency is switching between processes. Many computers have multiple CPUs.[90] Parallelism with multiple threads running on different CPUs can speed up a program, depending on how much of it can be executed concurrently.[91]

File system

Permanent storage devices used in twenty-first century computers, unlike volatile dynamic random-access memory (DRAM), are still accessible after a crash or power failure. Permanent (non-volatile) storage is much cheaper per byte, but takes several orders of magnitude longer to access, read, and write.[92][93] The two main technologies are a hard drive consisting of magnetic disks, and flash memory (a solid-state drive that stores data in electrical circuits). The latter is more expensive but faster and more durable.[94][95]

File systems are an abstraction used by the operating system to simplify access to permanent storage. They provide human-readable filenames and other metadata, increase performance via amortization of accesses, prevent multiple threads from accessing the same section of memory, and include checksums to identify corruption.[96] File systems are composed of files (named collections of data, of an arbitrary size) and directories (also called folders) that list human-readable filenames and other directories.[97] An absolute file path begins at the root directory and lists subdirectories divided by punctuation, while a relative path defines the location of a file from a directory.[98][99]


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

System calls (which are sometimes wrapped by libraries) enable applications to create, delete, open, and close files, as well as link, read, and write to them. All these operations are carried out by the operating system on behalf of the application.[100] The operating system's efforts to reduce latency include storing recently requested blocks of memory in a cache and prefetching data that the application has not asked for, but might need next.[101] Device drivers are software specific to each input/output (I/O) device that enables the operating system to work without modification over different hardware.[102][103]

Another component of file systems is a dictionary that maps a file's name and metadata to the data block where its contents are stored.[104] Most file systems use directories to convert file names to file numbers. To find the block number, the operating system uses an index (often implemented as a tree).[105] Separately, there is a free space map to track free blocks, commonly implemented as a bitmap.[105] Although any free block can be used to store a new file, many operating systems try to group together files in the same directory to maximize performance, or periodically reorganize files to reduce fragmentation.[106]

Maintaining data reliability in the face of a computer crash or hardware failure is another concern.[107] File writing protocols are designed with atomic operations so as not to leave permanent storage in a partially written, inconsistent state in the event of a crash at any point during writing.[108] Data corruption is addressed by redundant storage (for example, RAID—redundant array of inexpensive disks)[109][110] and checksums to detect when data has been corrupted. With multiple layers of checksums and backups of a file, a system can recover from multiple hardware failures. Background processes are often used to detect and recover from data corruption.[110]

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.[111] 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).[112] 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.[113] 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.[114]

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.[115] Unikernels are another approach that improves security by minimizing the kernel and separating out other operating systems functionality by application.[115]

Most operating systems are written in C or C++, which create potential vulnerabilities for exploitation. Despite attempts to protect against them, vulnerabilities are caused by buffer overflow attacks, which are enabled by the lack of bounds checking.[116] Hardware vulnerabilities, some of them caused by CPU optimizations, can also be used to compromise the operating system.[117] There are known instances of operating system programmers deliberately implanting vulnerabilities, such as back doors.[118]

Operating systems security is hampered by their increasing complexity and the resulting inevitability of bugs.[119] Because formal verification of operating systems may not be feasible, developers use operating system hardening to reduce vulnerabilities,[120] e.g. address space layout randomization, control-flow integrity,[121] access restrictions,[122] and other techniques.[123] There are no restrictions on who can contribute code to open source operating systems; such operating systems have transparent change histories and distributed governance structures.[124] Open source developers strive to work collaboratively to find and eliminate security vulnerabilities, using code review and type checking to expunge malicious code.[125][126] Andrew S. Tanenbaum advises releasing the source code of all operating systems, arguing that it prevents developers from placing trust in secrecy and thus relying on the unreliable practice of security by obscurity.[127]

User interface

A user interface (UI) is essential to support human interaction with a computer. The two most common user interface types for any computer are

- command-line interface, where computer commands are typed, line-by-line,
- graphical user interface (GUI) using a visual environment, most commonly a combination of the window, icon, menu, and pointer elements, also known as WIMP.

For personal computers, including smartphones and tablet computers, and for workstations, user input is typically from a combination of keyboard, mouse, and trackpad or touchscreen, all of which are connected to the operating system with specialized software.[128] Personal computer users who are not software developers or coders often prefer GUIs for both input and output; GUIs are supported by most personal computers.[129] The software to support GUIs is more complex than a command line for input and plain text output. Plain text output is often preferred by programmers, and is easy to support.[130]

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.[131]

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 hobby operating systems 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.

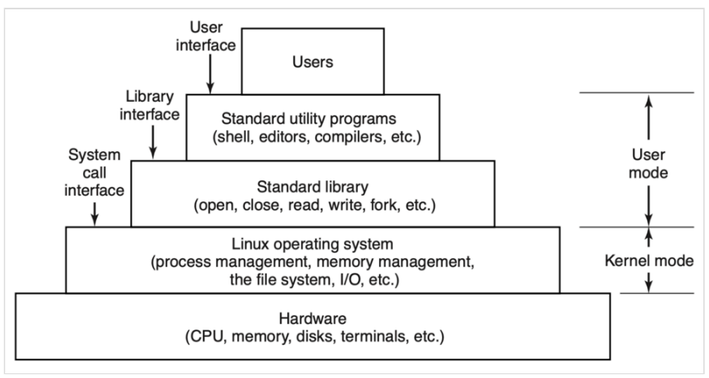
Popular operating systems

As of September 2024, Android (based on the Linux kernel) is the most popular operating system with a 46% market share, followed by Microsoft Windows at 26%, iOS and iPadOS at 18%, macOS at 5%, and Linux at 1%. Android, iOS, and iPadOS are mobile operating systems, while Windows, macOS, and Linux are desktop operating systems.[3]

Linux

Linux is a free software distributed under the GNU General Public License (GPL), which means that all of its derivatives are legally required to release their source code.[132] Linux was designed by programmers for their own use, thus emphasizing simplicity and consistency, with a small number of basic elements that can be combined in nearly unlimited ways, and avoiding redundancy.[133]

Its design is similar to other UNIX systems not using a microkernel.[134] It is written in C[135] and uses UNIX System V syntax, but also supports BSD syntax. Linux supports standard UNIX networking features, as well as the full suite of UNIX tools, while supporting multiple users and employing preemptive multitasking. Initially of a minimalist design, Linux is a flexible system that can work in under 16 MB of RAM, but still is used on large multiprocessor systems.[134] Similar to other UNIX systems, Linux distributions are composed of a kernel, system libraries, and system utilities.[136] Linux has a graphical user interface (GUI) with a desktop, folder and file icons, as well as the option to access the operating system via a command line.[137]



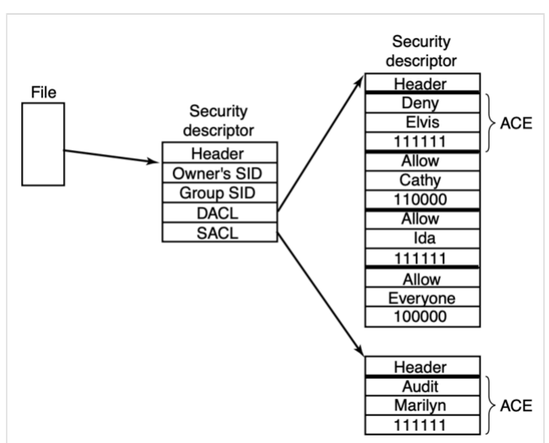
Layers of a Linux system

Android is a partially open-source operating system closely based on Linux and has become the most widely used operating system by users, due to its popularity on smartphones and, to a lesser extent, embedded systems needing a GUI, such as "smart watches, automotive dashboards, airplane seatbacks, medical devices, and home appliances".[138] Unlike Linux, much of Android is written in Java and uses object-oriented design.[139]

Microsoft Windows

Windows is a proprietary operating system that is widely used on desktop computers, laptops, tablets, phones, workstations, enterprise servers, and Xbox consoles.[141] The operating system was designed for "security, reliability, compatibility, high performance, extensibility, portability, and international support"—later on, energy efficiency and support for dynamic devices also became priorities.[142]

Windows Executive works via kernel-mode objects for important data structures like processes, threads, and sections (memory objects, for example files).[143] The operating system supports demand paging of virtual memory, which speeds up I/O for many applications. I/O device drivers use the Windows Driver Model.[143] The NTFS file system has a master table and each file is represented as a record with metadata.[144] The scheduling includes preemptive multitasking.[145] Windows has many security features;[146] especially important are the use of access-control lists and integrity levels. Every process has an authentication token and each object is given a security descriptor. Later releases have added even more security features.[144]



Security descriptor for a file that is read-only by default, specified no access for Elvis, read/write access for Cathy, and full access for Ida, the owner of the file[140]

See also

- Comparison of operating systems
- DBOS
- Interruptible operating system
- List of operating systems
- List of pioneers in computer science
- Glossary of operating systems terms
- Microcontroller
- Network operating system
- Object-oriented operating system
- Lisp machine
- Operating System Projects
- System Commander

- System image
- Timeline of operating systems

Notes

a. Modern CPUs provide instructions (e.g. SYSENTER) to invoke selected kernel services without an interrupts. Visit https://wiki.osdev.org/SYSENTER for more information.
b. Examples include SIGINT, SIGSEGV, and SIGBUS.
c. often in the form of a DMA chip for smaller systems and I/O channels for larger systems
d. Modern motherboards have a DMA controller. Additionally, a device may also have one. Visit SCSI RDMA Protocol.
e. There are several reasons that the memory might be inaccessible

- The address might be out of range
- The address might refer to a page or segment that has been moved to a backing store
- The address might refer to memory that has restricted access due to, e.g., key, ring.

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. "Operating System Market Share Worldwide" (https://gs.statcounter.com/os-market-share). *StatCounter Global Stats*. Retrieved 20 December 2024.
4. "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.
5. "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.
6. Tanenbaum & Bos 2023, p. 4.
7. Anderson & Dahlin 2014, p. 6.
8. Silberschatz et al. 2018, p. 6.
9. Anderson & Dahlin 2014, p. 7.
10. Anderson & Dahlin 2014, pp. 9–10.
11. Tanenbaum & Bos 2023, pp. 6–7.
12. Anderson & Dahlin 2014, p. 10.
13. Tanenbaum & Bos 2023, p. 5.
14. Anderson & Dahlin 2014, p. 11.
15. Anderson & Dahlin 2014, pp. 7, 9, 13.
16. Anderson & Dahlin 2014, pp. 12–13.
17. Tanenbaum & Bos 2023, p. 557.
18. Tanenbaum & Bos 2023, p. 558.
19. Tanenbaum & Bos 2023, p. 565.
20. Tanenbaum & Bos 2023, p. 562.
21. Tanenbaum & Bos 2023, p. 563.
22. Tanenbaum & Bos 2023, p. 569.
23. Tanenbaum & Bos 2023, p. 571.
24. Tanenbaum & Bos 2023, p. 579.
25. Tanenbaum & Bos 2023, p. 581.
26. Tanenbaum & Bos 2023, pp. 37–38.
27. Tanenbaum & Bos 2023, p. 39.
28. Tanenbaum & Bos 2023, p. 38.
29. Silberschatz et al. 2018, pp. 701.
30. Silberschatz et al. 2018, pp. 705.
31. Anderson & Dahlin 2014, p. 12.
32. Madhavapeddy, Anil; Scott, David J (November 2013). "Unikernels: Rise of the Virtual Library Operating System: What if all the software layers in a virtual appliance were compiled within the same safe, high-level language framework?" (https://doi.org/10.1145/2557963.2566628). *Queue*. Vol. 11, no. 11. New York, NY, USA: ACM. pp. 30–44. doi:10.1145/2557963.2566628 (https://doi.org/10.1145%2F2557963.2566628). ISSN 1542-7730 (https://search.worldcat.org/issn/1542-7730). Retrieved 7 August 2024.
33. "Build Process - Unikraft" (https://unikraft.org/docs/concepts/build-process). Archived (https://web.archive.org/web/20240422183734/https://unikraft.org/docs/concepts/build-process) from the original on 22 April 2024. Retrieved 8 August 2024.
34. "Leave your OS at home: the rise of library operating systems" (https://www.sigarch.org/leave-your-os-at-home-the-rise-of-library-operating-systems/). ACM SIGARCH. 14 September 2017. Archived (https://web.archive.org/web/20240301072916/https://www.sigarch.org/leave-your-os-at-home-the-rise-of-library-operating-systems/) from the original on 1 March 2024. Retrieved 7 August 2024.
35. Soares, Livio Baldini; Stumm, Michael (4 October 2010). *FlexSC: Flexible System Call Scheduling with Exception-Less System Calls* (https://www.usenix.org/conference/osdi10/flexsc-flexible-system-call-scheduling-exception-less-system-calls). OSDI '10, 9th USENIX Symposium on Operating System Design and Implementation (https://www.usenix.org/legacy/events/osdi10/). USENIX. Retrieved 9 August 2024. p. 2: "Synchronous implementation of system calls negatively impacts the performance of system intensive workloads, both in terms of the *direct* costs of mode switching and, more interestingly, in terms of the *indirect* pollution of important processor structures which affects both user-mode and kernel-mode performance. A motivating example that quantifies the impact of system call pollution on application performance can be seen in Figure 1. It depicts the user-mode instructions per cycles (kernel cycles and instructions are ignored) of one of the SPEC CPU 2006 benchmarks (Xalan) immediately before and after a `pwrite` system call. There is a significant drop in instructions per cycle (IPC) due to the system call, and it takes up to 14,000 cycles of execution before the IPC of this application returns to its previous level. As we will show, this performance degradation is mainly due to interference caused by the kernel on key processor structures."
36. Tanenbaum & Bos 2023, p. 8.
37. 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.
38. Tanenbaum & Bos 2023, p. 10.
39. Tanenbaum & Bos 2023, pp. 11–12.
40. Tanenbaum & Bos 2023, pp. 13–14.
41. Tanenbaum & Bos 2023, pp. 14–15.
42. Tanenbaum & Bos 2023, p. 15.
43. Tanenbaum & Bos 2023, pp. 15–16.
44. Tanenbaum & Bos 2023, p. 16.
45. Tanenbaum & Bos 2023, p. 17.
46. Tanenbaum & Bos 2023, p. 18.
47. Tanenbaum & Bos 2023, pp. 19–20.
48. Anderson & Dahlin 2014, pp. 39–40.
49. Tanenbaum & Bos 2023, p. 2.
50. Anderson & Dahlin 2014, pp. 41, 45.
51. Anderson & Dahlin 2014, pp. 52–53.
52. 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."
53. 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."
54. 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."
55. 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."
56. 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."
57. Silberschatz, Abraham (1994). *Operating System Concepts, Fourth Edition*. Addison-Wesley. p. 31. ISBN 978-0-201-50480-4.
58. 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."
59. 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."
60. 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[.]"
61. Kerrisk, Michael (2010). *The Linux Programming Interface*. No Starch Press. p. 388. ISBN 978-1-59327-220-3.
62. "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.
63. Bach, Maurice J. (1986). *The Design of the UNIX Operating System*. Prentice-Hall. p. 200. ISBN 0-13-201799-7.
64. Kerrisk, Michael (2010). *The Linux Programming Interface*. No Starch Press. p. 400. ISBN 978-1-59327-220-3.
65. 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.
66. Silberschatz, Abraham (1994). *Operating System Concepts, Fourth Edition*. Addison-Wesley. p. 182. ISBN 978-0-201-50480-4.
67. Haviland, Keith; Salama, Ben (1987). *UNIX System Programming*. Addison-Wesley Publishing Company. p. 153. ISBN 0-201-12919-1.
68. Haviland, Keith; Salama, Ben (1987). *UNIX System Programming*. Addison-Wesley Publishing Company. p. 148. ISBN 0-201-12919-1.
69. Haviland, Keith; Salama, Ben (1987). *UNIX System Programming*. Addison-Wesley Publishing Company. p. 149. ISBN 0-201-12919-1.
70. 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.
71. 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
72. 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.
73. "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.
74. *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.
75. Silberschatz, Abraham (1994). *Operating System Concepts, Fourth Edition*. Addison-Wesley. p. 32. ISBN 978-0-201-50480-4.
76. Silberschatz, Abraham (1994). *Operating System Concepts, Fourth Edition*. Addison-Wesley. p. 34. ISBN 978-0-201-50480-4.
77. 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.
78. 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.
79. 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.
80. Stallings, William (2008). *Computer Organization & Architecture*. New Delhi: Prentice-Hall of India Private Limited. p. 267. ISBN 978-81-203-2962-1.
81. Anderson & Dahlin 2014, p. 129.
82. Silberschatz et al. 2018, p. 159.
83. Anderson & Dahlin 2014, p. 130.
84. Anderson & Dahlin 2014, p. 131.
85. Anderson & Dahlin 2014, pp. 157, 159.
86. Anderson & Dahlin 2014, p. 139.
87. Silberschatz et al. 2018, p. 160.
88. Anderson & Dahlin 2014, p. 183.
89. Silberschatz et al. 2018, p. 162.
90. Silberschatz et al. 2018, pp. 162–163.
91. Silberschatz et al. 2018, p. 164.
92. Anderson & Dahlin 2014, pp. 492, 517.
93. Tanenbaum & Bos 2023, pp. 259–260.
94. Anderson & Dahlin 2014, pp. 517, 530.
95. Tanenbaum & Bos 2023, p. 260.
96. Anderson & Dahlin 2014, pp. 492–493.
97. Anderson & Dahlin 2014, p. 496.
98. Anderson & Dahlin 2014, pp. 496–497.
99. Tanenbaum & Bos 2023, pp. 274–275.
100. Anderson & Dahlin 2014, pp. 502–504.

101. Anderson & Dahlin 2014, p. 507.
102. Anderson & Dahlin 2014, p. 508.
103. Tanenbaum & Bos 2023, p. 359.
104. Anderson & Dahlin 2014, p. 545.
105. Anderson & Dahlin 2014, p. 546.
106. Anderson & Dahlin 2014, p. 547.
107. Anderson & Dahlin 2014, pp. 589, 591.
108. Anderson & Dahlin 2014, pp. 591–592.
109. Tanenbaum & Bos 2023, pp. 385–386.
110. Anderson & Dahlin 2014, p. 592.
111. Tanenbaum & Bos 2023, pp. 605–606.
112. Tanenbaum & Bos 2023, p. 608.
113. Tanenbaum & Bos 2023, p. 609.
114. Tanenbaum & Bos 2023, pp. 609–610.
115. Tanenbaum & Bos 2023, p. 612.
116. Tanenbaum & Bos 2023, pp. 648, 657.
117. Tanenbaum & Bos 2023, pp. 668–669, 674.
118. Tanenbaum & Bos 2023, pp. 679–680.
119. Tanenbaum & Bos 2023, pp. 605, 617–618.
120. Tanenbaum & Bos 2023, pp. 681–682.
121. Tanenbaum & Bos 2023, p. 683.
122. Tanenbaum & Bos 2023, p. 685.
123. Tanenbaum & Bos 2023, p. 689.
124. Richet & Bouaynaya 2023, p. 92.
125. Richet & Bouaynaya 2023, pp. 92–93.
126. Berntsso, Strandén & Warg 2017, pp. 130–131.
127. Tanenbaum & Bos 2023, p. 611.
128. Tanenbaum & Bos 2023, pp. 396, 402.
129. Tanenbaum & Bos 2023, pp. 395, 408.
130. Tanenbaum & Bos 2023, p. 402.
131. Holwerda, Thom (20 December 2009). "My OS Is Less Hobby than Yours" (https://www.osnews.com/story/22638/my-os-is-less-hobby-than-yours/). *OS News*. Retrieved 4 June 2024.
132. Silberschatz et al. 2018, pp. 779–780.
133. Tanenbaum & Bos 2023, pp. 713–714.
134. Silberschatz et al. 2018, p. 780.
135. Vaughan-Nichols, Steven (2022). "Linus Torvalds prepares to move the Linux kernel to modern C" (https://www.zdnet.com/article/linus-torvalds-prepares-to-move-the-linux-kernel-to-modern-c/). *ZDNET*. Retrieved 7 February 2024.
136. Silberschatz et al. 2018, p. 781.
137. Tanenbaum & Bos 2023, pp. 715–716.
138. Tanenbaum & Bos 2023, pp. 793–794.
139. Tanenbaum & Bos 2023, p. 793.
140. Tanenbaum & Bos 2023, pp. 1021–1022.
141. Tanenbaum & Bos 2023, p. 871.
142. Silberschatz et al. 2018, p. 826.
143. Tanenbaum & Bos 2023, p. 1035.
144. Tanenbaum & Bos 2023, p. 1036.
145. Silberschatz et al. 2018, p. 821.
146. Silberschatz et al. 2018, p. 827.

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). ISSN 0018-8646 (https://search.worldcat.org/issn/0018-8646).
- 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.
- 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) (inactive 1 November 2024).
- Silberschatz, Abraham; Galvin, Peter B.; Gagne, Greg (2018). *Operating System Concepts* (https://archive.org/details/operating-system-concepts-10th) (10 ed.). Wiley. ISBN 978-1-119-32091-3.
- Tanenbaum, Andrew S.; Bos, Herbert (2023). *Modern Operating Systems, Global Edition*. Pearson Higher Ed. ISBN 978-1-292-72789-9.

External links

- Multics History (http://www.cbi.umn.edu/iterations/haigh.html) and the history of operating systems