Operating Systems

RAPHAEL A. FINKEL

University of Kentucky, Lexington ⟨raphael@cs.engr.uky.edu⟩

WHAT IS AN OPERATING SYSTEM?

An operating system is the set of programs that control a computer. Many books on operating systems describe operating systems concepts [Deitel 1992; Finkel 1988; Krakowiak 1988; Lane and Mooney 1988; Milenkovic 1992; Silberschatz and Galvin 1994; Stallings 1992; Tanenbaum 1992] and specific operating systems [Kenah and Bate 1994; Leffler et al. 1989].

Operating system software includes several levels: kernel services, library services, and application-level services. Applications are run by *processes*, programs linked together with libraries, that are running on the computer. The *kernel* supports the processes by providing them the resources they need. It responds to *service calls* from the processes and interrupts from the devices.

Operating systems accomplish three major goals. They hide details of hardware by creating abstractions; they manage resources; and they provide a pleasant user interface.

Abstractions. An *abstraction* is software (often implemented as a subroutine or a library of subroutines) that hides lower-level details and provides a set of higher-level functions. Programs that use the abstraction can safely ignore the lower-level (*physical*) details; they need deal only with the higher-level (*virtual*) structures.

Abstraction is used to standardize the appearance of peripheral devices into uniform interfaces, to turn unstructured disk blocks into files, and to transform the hardware into multiple virtual computers, each belonging to a different process.

The kernel generally enforces these abstractions by running all processes in a non-privileged state. Instructions such as those that perform input/output and those that change processor state cause traps when executed in non-privileged state. The kernel runs in privileged state.

Resources. A resource is a commodity necessary to get work done. The computer's hardware provides a number of low-level resources. Working programs need to reside somewhere in the main store (the computer's memory), to execute instructions, and to accept data and present results. These needs are related to the fundamental resources of *memory, time*, and *input/output*. In addition, the operating system creates new abstract resources, such as files and pseudo-files (objects that appear to be data files and represent devices, processes, or inter-process communication ports).

The resource needs of processes often interfere with each other. Resource managers in the operating system include policies that try to be fair in giving resources to the processes and allow as much computation to proceed as possible. These goals often conflict.

Each resource has its own manager, typically in the kernel. The memory manager allocates regions of main memory for processes. Most modern operating systems use address-translation hardware that maps between a process's *virtual addresses* and the underlying *physical addresses*. Only the currently active part of a process's virtual

space needs to be physically resident; the rest is kept on backing store (usually a disk) and brought in on demand. The virtual spaces of processes do not usually overlap except, perhaps, to share read-only instruction regions. Some operating systems also provide *lightweight processes* that share a single virtual space. The memory manager includes policies that determine how much physical memory to grant to each process and which region of physical memory to swap out to make room for other memory that must be swapped in.

The time manager is called the *scheduler*. Schedulers usually implement a preemptive policy that forces the processes to take turns at running. Schedulers categorize processes according to whether they are currently runnable (they may not be if they are waiting for other resources) and by their priority.

The file manager mediates process requests such as creating, reading, and writing files. It validates access based on the identity of the user running the process and the permissions associated with the file. The file manager also prevents conflicting accesses by multiple processes to the same file. It translates input/output requests into device accesses, usually to a disk, but often to networks (for remote files) or other devices (for pseudo-files).

The device managers convert standard-format requests into the particular commands appropriate for the individual devices, which vary widely across device types and manufacturers. These managers may maintain caches of data in memory to reduce the frequency of access to physical devices.

User interface. The human user interacts with the operating system through the user interface. Where the user used to communicate with the computer by setting switches and, later, entering punched cards and reading line-printer output, most users today use graphics terminals. Programs that interact with users can either use a text-based interface or employ color, fonts, and interactive menus, controlled largely by the mouse or other pointing devices. The display includes sound, pictures, and video in addition to text.

The look and feel of the operating system is affected by many components of the user interface. Some of the most important components are the command interpreter, the file system, online help, and application integration.

INSIDE THE KERNEL

Operating systems used to be written as single large programs encompassing hundreds of thousands of lines of assembler instructions. Modern operating systems are written in high-level languages (often C) in a modular, layered fashion. Layers include process control, device drivers, resource managers, file system support, and the service call interpreter. Each module has its own data structures. For example, the process control module keeps a list of all processes, where it stores information on resource allocation and the current scheduling state.

When all the layers are privileged, the organization is called a *macrokernel*. Most commercially available operating systems use this design. If the kernel contains code only for process creation, interprocess communication, the mechanisms (but not the policies) for memory management, scheduling, and the lowest level of device control, the result is a *microkernel*. Services such as the file system and policy modules for scheduling and memory are relegated to processes called *servers*; the ordinary processes that need those services are called *clients*.

SUMMARY

Modern operating systems generally have three goals: to hide details of hardware by creating abstractions, to allocate resources to processes, and to provide a pleasant user interface. Operating systems generally accomplish these goals by running processes in low privilege and providing service calls

that invoke the operating system kernel in a high-privilege state. The recent trend has been toward increasingly integrated graphical user interfaces that represent the activities of multiple processes on networks of computers. These increasingly sophisticated application programs are supported by increasingly small operating system kernels.

REFERENCES

DEITEL, H. M. 1992. *Operating Systems.* Addison-Wesley, Reading, MA.

FINKEL, R. 1988. *An Operating Systems Vade Mecum*. Prentice-Hall, Englewood Cliffs, NJ.

KENAH, L. J. AND BATE, S. F. 1984. *VAX/VMS Internals and Data Structures*. Digital Equipment Corporation.

KRAKOWIAK, S. 1988. *Principles of Operating Systems*. MIT Press, Cambridge, MA.

LANE, M. G. AND MOONEY, J. D. 1988. *A Practical Approach to Operating Systems*. Boyd and Fraser.

LEFFLER, S. J., MCKUSICK, M. K., KARELS, M. J., AND QUARTERMAN, J. S. 1989. *4.3BSD UNIX Operating System*. Addison-Wesley, Reading, MA.

MILENKOVIC, M. 1992. *Operating Systems: Concepts and Design,* 2nd ed. McGraw-Hill, New York.

SILBERSCHATZ, A. AND GALVIN, P. B. 1994. *Operating Systems Concepts: Fourth Edition*. Addison-Wesley, Reading, MA.

STALLINGS, W. 1992. *Operating Systems*. Macmillan, New York.

TANENBAUM, A. S. 1992. *Modern Operating Systems*. Prentice-Hall, Englewood Cliffs, NJ.