Operating System Structures

Chapter 2

An operating system provides the environment within which programs are executed.

2.1 Operating-System Services

An operating system provides an environment for the execution of programs. It provides certain services to programs and to the users of those programs.

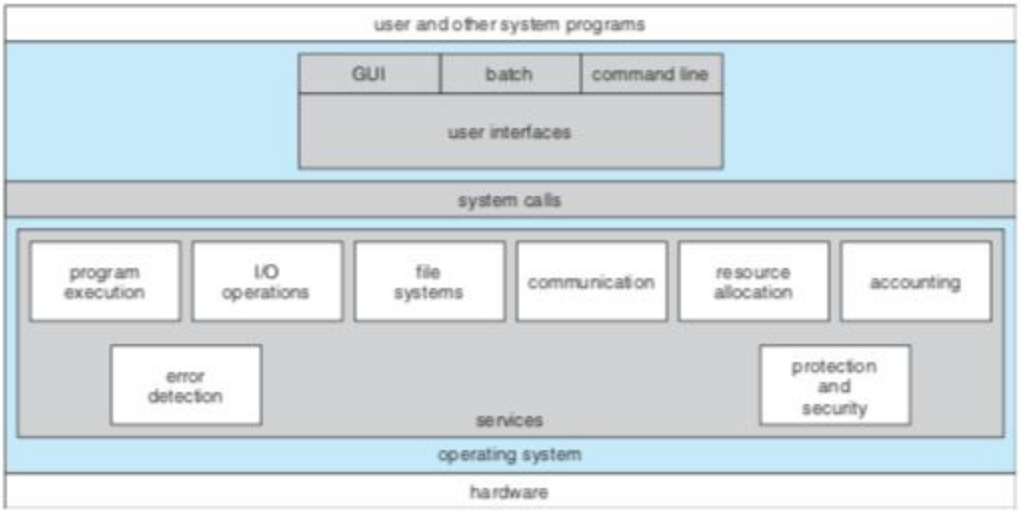

Figure 2.1 A view of operating system services.

- User interface
  - The interface can take several forms. A command-line interface uses text commands and a method for entering them.
  - Batch interface, in which commands and directives to control those commands are entered into files, and those files are executed.
  - Graphical user interface where the interface is a window system with a pointing device to direct I/O, choose from menus, and make selections and a keyboard to enter text.
- Program execution
  - The system must be able to load a program into memory and to run that program. The program must be able to end its execution, either normally or abnormally (indicating error)

- I/O operations
    - A running program may require I/O which may involve a file or an I/O device
- File-system manipulation
    - Programs need to read and write files and directories. They also need to create and delete them by name, search for a given file, and list file information.
    - Finally, some operating systems include permissions management to allow or deny access to files or directories based on file ownership.
- Communications
    - Communication may occur between processes that are executing on the same computer or between processes that are executing on different computer systems tied together by a computer network.
    - Communications may be implemented via shared memory, in which two or more processes read and write to a shared section of memory, or message passing, in which packets of information in predefined formats are moved between processes by the operating systems.
- Error detection
    - The operating system needs to be detecting and correcting errors constantly.

Another set of operating system functions exists not for helping the user but rather for ensuring the efficient operation of the system itself.

- Resource allocation
    - When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them.
- Accounting
    - We want to keep track of which users use how much and what kinds of computer resources.
- Protection and security
    - When several separate processes execute concurrently, it should not be possible for one process to interfere with the others or with the operating system itself.
    - Protection involves ensuring that all access to system resources is controlled.

2.2 User and Operating-System Interface

Command interpreter that allows users to directly enter commands to be performed but the operating system.

2.2.1 Command Interpreters

The main function of the command interpreter is to get and execute the next user-specified command.

2.2.2 Graphical User Interface

Rather than entering commands directly via a command-line interface, users employ a mouse-based window and menu system characterized by a desktop metaphor.

2.2.3 Choice of Interface

System administrators who manage computers and power users who have deep knowledge of a system frequently use the command-line interface.

2.3 System Calls

System calls provide an interface to the services made available by an operating system.

Let's use an example to illustrate how system calls are used: writing a simple program to read data from one file and copy them to another file. The first input that the program will need is the names of the two files: the input file and the output file. These names can be specified in many ways, depending on the operating-system design. One approach is for the program to ask the user for the names. In an interactive system, this approach will require a sequence of system calls, first to write a prompting message on the screen and then to read from the keyboard the characters that define the two files. On mouse-based and icon-based systems, a menu of file names is usually displayed in a window. The user can then use the mouse to select the source name, and a window can be opened for the destination name to be specified. This sequence requires many I/O system calls.

Once the two file names have been obtained, the program must open the input file and create the output file. Each of these operations requires another system call. Possible error conditions for each operation can require additional system calls. When the program tries to open the input file, for example, it may find that there is no file of that name or that the file is protected against access. In these cases, the program should print a message on the console (another sequence of system calls) and then terminate abnormally (another system call). If the input file exists, then we must create a new output file. We may find that there is already an output file with the same name. This situation may cause the program to abort (a system call), or we may delete the existing file (another system call) and create a new one (yet another system call). Another option, in an interactive system, is to ask the user (via a sequence of system calls to output the prompting message and to read the response from the terminal) whether to replace the existing file or to abort the program.

When both files are set up, we enter a loop that reads from the input file (a system call) and writes to the output file (another system call). Each read and write must return status information regarding various possible error conditions. On input, the program may find that the end of the file has been reached or that there was a hardware failure in the read (such as a parity error). The write operation may encounter various errors, depending on the output device (for example, no more disk space).
Finally, after the entire file is copied, the program may close both files (another system call), write a message to the console or window (more system calls), and finally terminate normally (the final system call).

An Application Programming Interface (API) specifies a set of functions that are available to an application programmer, including the parameters that are passed to each function and the return values the programmer can expect.

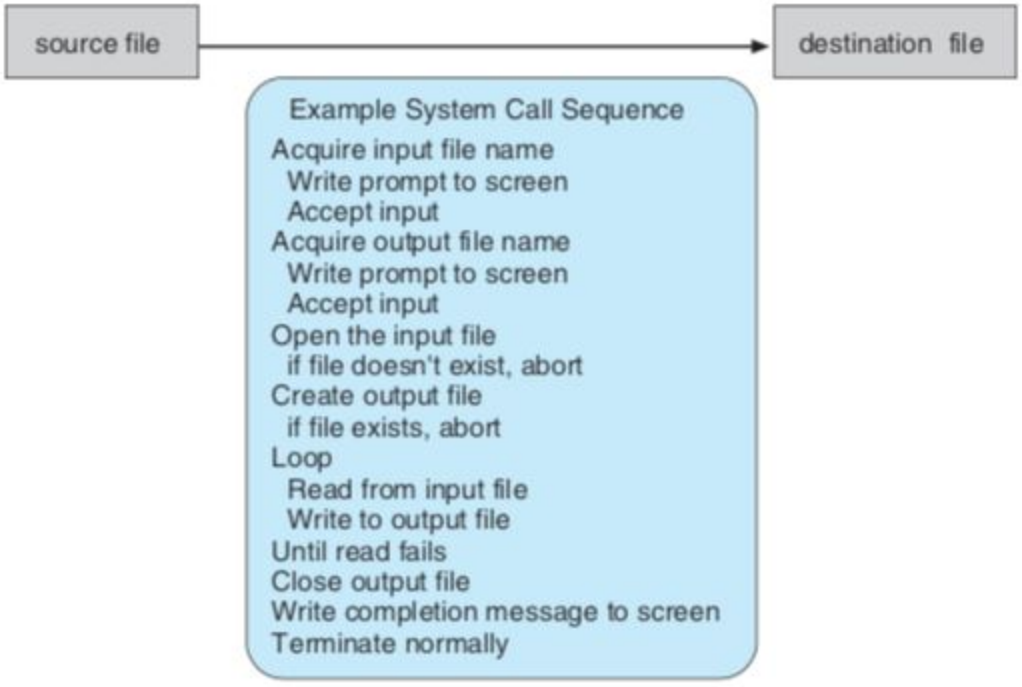

Figure 2.5 Example of how system calls are used.

A system- call interface that serves as the link to system calls made available by the operating system. The system-call interface intercepts function calls in the API and invokes the necessary system calls within the operating system.

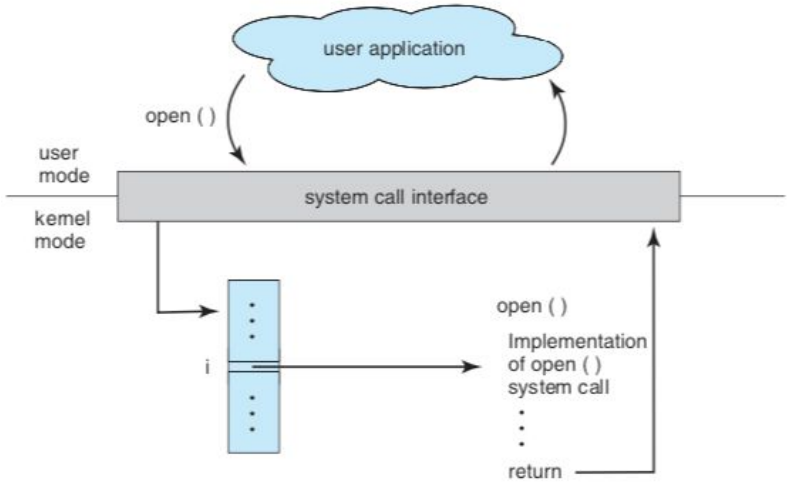

Figure 2.6 The handling of a user application invoking the `open()` system call.

Three general methods are used to pass parameters to the operating system. The simplest approach is to pass the parameters in registers. In some cases, however, there may be more parameters than registers. In these cases, the parameters are generally stored in a block, or table, in memory, and the address of the block is passed as a parameter in a register (Figure 2.7). This is the approach taken by Linux and Solaris. Parameters also can be placed, or pushed, onto the stack by the program and popped off the stack by the operating system. Some operating systems prefer the block or stack method because those approaches do not limit the number or length of parameters being passed.

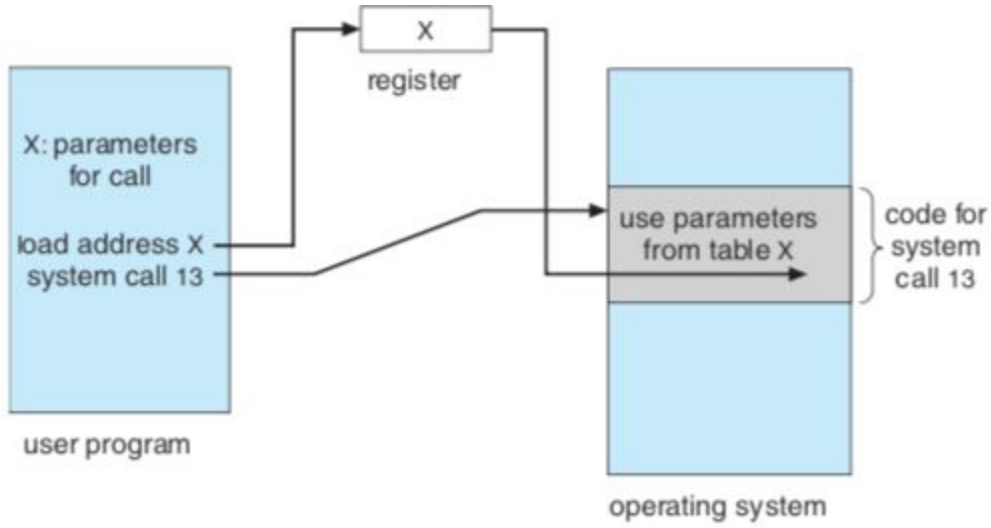

Figure 2.7 Passing of parameters as a table.

2.4 Types of System Calls

System calls can be grouped roughly into six major categories: process control, file manipulation, device manipulation, information maintenance, communications and protection.

2.4.1 Process Control

- Processcontrol
  - end, abort
  - load, execute
  - create process, terminate process
  - get process attributes, set process attributes
  - wait for time

    - wait event, signal event
    - allocate and free memory
- File management
    - create file, delete file
    - open, close
    - read, write, reposition
    - get file attributes, set file attributes
- Devicemanagement
    - request device, release device
    - read, write, reposition
    - get device attributes, set device attributes
    - logically attach or detach devices
- Information maintenance
    - get time or date, set time or date
    - get system data, set system data
    - get process, file, or device attributes ◦ set process, file, or device attributes
- Communications
    - create, delete communication connection
    - send, receive messages
    - transfer status information
    - attach or detach remote devices

2.4.3 Device Management

A process may need several resources to execute - main memory, disk drives, access to files, and so on. If the resources are available, they can be granted, and control can be returned to the user process. Otherwise, the process will have to wait until sufficient resources are available.

2.4.4. Information Maintenance

Many system calls exist simply for the purpose of transferring information between the user program and the operating system.

2.4.5 Communication

There are two common models of interprocess communication: the message-passing model and the shared-memory model. In the message-passing model, the communicating processes exchange messages with one another to transfer information.
In the shred-memory model, processes use shared_memory_create() and shared_memory_attach() sustem calls to create and gain acccess to regions of memory owned by other processes. Recall that, normally, the operating sustem tries to prevent one process from accessing another process's memory. Shared memory requires that two or more

processes agree to remove this restriction. Message passing is useful for exchanging smaller amounts of data, because no conflicts need to be avoided.

2.4.6 Protection

Protection provides a mechanism for controlling access to the resources provided by a computer system.

2.5 System Programs

System programs provide a convenient environment for program development and execution. Some of them are simply user interfaces to system calls.
They can be divided in these categories:

- File Management
- Status information
- File modification
- Programming-language support
- Program loading and execution
- Communications
- Background services

Along with system programs, most operating systems are supplied with programs that are useful in solving common problems or performing common operations. - Application programs

2.6 Operating-System Design and Implementation

2.6.1 Design Goals

The first problem in designing a system is to define goals and specifications. At the highest level, the design of the system will be affected by the choise of hardware and the type of system: batch, time sharing, single user, multiuser, distributed, real time, or general purpose. The requirements can, however, be divided into two basic groups: user goals and system goals.

2.6.2 Mechanisms and Policies

One important principle is the separation of policy from mechanism. Mechanisms determine how to do something; policies determine what will be done.

2.6.3 Implementation

The advantages of using a higher-level language, or at least a systems-implementation language, for implementing operating systems are the same as those gained when the language is used for application programs: the code can be written faster, is more compact, and is easier to understand and debug.

An operating system is far easier to port - to move to some other hardware- if it is written in a higher-level language.

2.7 Operating-System Structure

2.7.2 Layered Approach

A system can be made modular in many ways. One method is the layered approach. In which the operating system is broken into a number of layers (levels). The bottom layer is the hardware, the highest is the user interface.

The main advantage of the layered approach is simplicity of construction and debugging. The layers are selected so that each uses functions (operations) and services of only lower-level layers, This approach simplifies debugging and system verification. The first layer can be debugged without any concern for the rest of the system, because, by definition, it uses only the basic hardware (which is assumed correct) to implement its functions.

The mayor difficulty with the layered approach involves appropriately defining the various layers. Because a layer can use only lower-level layers, careful planning is necessary.

2.7.3 Microkernels

This method structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs. The result is a smaller kernel. There is little consensus regarding which services should remain in the kernel and which should be implemented in user space

2.7.4 Modules

Perhaps the best current methodology for operating-system design involves using loadable kernel modules. Here, the kernel has a set of core components and links in additional services via modules, either at boot time or during run time.

2.7.5 Hybrid Systems

Operating systems combine different structures, resulting in hybrid systems that address performance, security, and usability issues.

2.8 Operating-System Debugging

Debugging is the activity of finding and fixing errors in a system, both in hardware and in software. Performance problems are considered bugs, so debugging can also include performance tuning, which seeks to improve performance by removing processing bottlenecks.

2.8.1 Failure Analysis

If a process fails, most operating systems write the error information to a log file to alert system operators that the problem occurred. The operating system can also take a core dump - a capture of the memory of the process - and store it in a file for later analysis.
A failure in the kernel is called a crash, When a crash occurs, error information is saved to a log file, and the memory state is saved to a crash dump.

2.8.2 Performance Tuning

The operating system must have some means of computing and displaying measures of system behavior. The operating does this by producing trace listings of system behaviour.

2.8.3 DTrace

DTrace is a facility that dynamically adds probes to a running system, both in user processes and in the kernel. Debugging the interactions between user-level and kernel code is nearly impossible without a toolset that understands both sets of code and can instrument the interactions.
Profiling, which periodically samples the instruction pointer to determine which code is being executed, can show statistical trends but not individual activities.
Within the kernel, actions called enabling control blocks, or ECBs, are performed when probes fire. One probe can cause multiple ECBs to execute if more than one consumer is interested in that probe.

2.9 Operating-System Generation

The system must be configured or generated for each specific computer site, a process sometimes known as system generation SYSGEN. This SYSGEN program reads from a given file, or asks the operator of the system for information concerning the specific configuration of the hardware system, or probes the hardware directly to determine what components are there.

2.10 System Boot

The procedure of starting a computer by loading the kernel is known as booting the system. On most computer systems, a small piece of code known as the bootstrap program or boostrap loader locates the kernel, loads it into main memory, and starts its execution. At a location is the initial bootstrap program. This program is in the form of read-only memory (ROM), because the RAM is in an unknown state at system startup. ROM is convenient because it needs no initialization and cannot easily be infected by a computer virus. All forms of ROM are also known as firmware, since their characteristics fall somewhere between those of hardware and those of software.