جامعة ألأنبار

كلية علوم الحاسوب وتكنولوجيا المعلومات

قسم أنظمة شبكات الحاسوب

المرحلة الرابعه

Operating System

التدريسي: أ.م.د عمر منذر حسين

Overview

We can view an operating system from several points:

1. One view focuses on the **services that the system provides**;
2. on the interface that it makes available to users and programmers;
3. a third, on its components and their interconnections.

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.

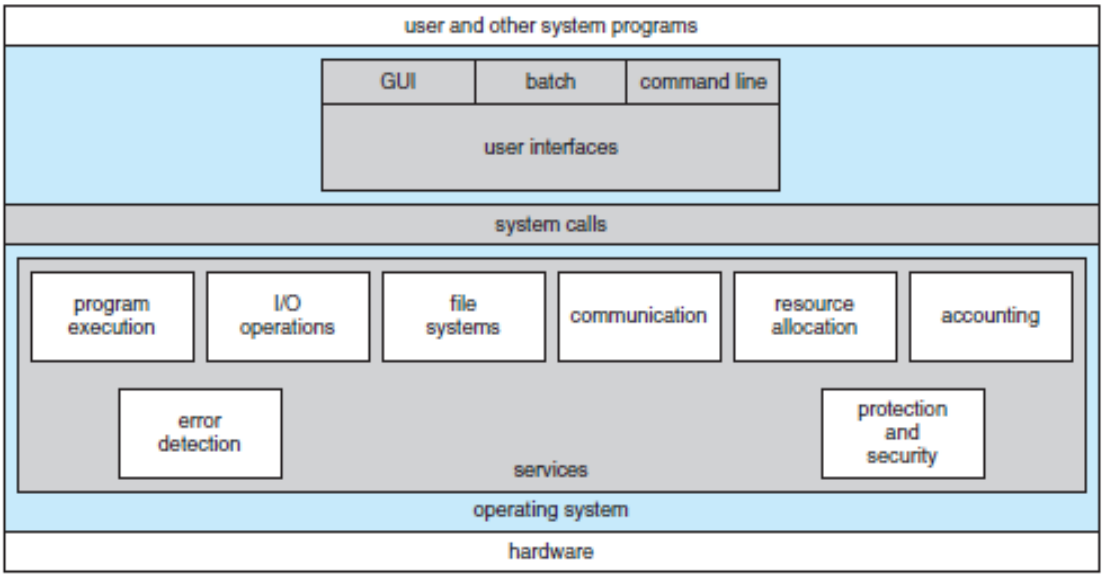

Figure 2.1 A view of operating system services.

- **One set of operating system services provides functions that are helpful to the user:**
  - **User interface.** Almost all operating systems have a user interface (UI). This interface can take several forms:
  - **Command-Line Interface (CLI)**, which uses text commands and a method for entering them.
  - **Batch interface**, in which commands are entered into files, and those files are executed.

- Most commonly, a **Graphical User Interface (GUI)**. Here, 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. For efficiency and protection, users usually cannot control I/O devices directly. Therefore, the operating system must provide a means to do I/O.
- **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.
- **Communications.** There are many conditions in which one process needs to exchange information with another process. Such 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 system.
- **Error detection.** The operating system needs to be detecting and correcting errors constantly. Errors may occur in the CPU and memory, in I/O devices, on disk, a connection failure on a network, or lack of paper in the printer, and in the user program. For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing. Sometimes, it has no choice but to halt the system. At other times, it might terminate an error-causing process or return an error code to a process for the process to detect and possibly correct.

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

multiple users can gain efficiency by sharing the computer resources among the users.

1. **Resource allocation.** [CPU cycles, main memory, and file storage, I/O devices, printers, USB storage drives, and other peripheral devices]
2. **Accounting.** keep track of which users use how much and what kind of computer resources.
3. **Protection and security.**

System Calls

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

- As you can see, even simple programs may make heavy use of the operating system. Frequently, systems execute thousands of system calls per second.
- Typically, application developers design programs according to an application programming interface (API). The 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.
- A programmer accesses an API via a library of code provided by the operating system.

Example

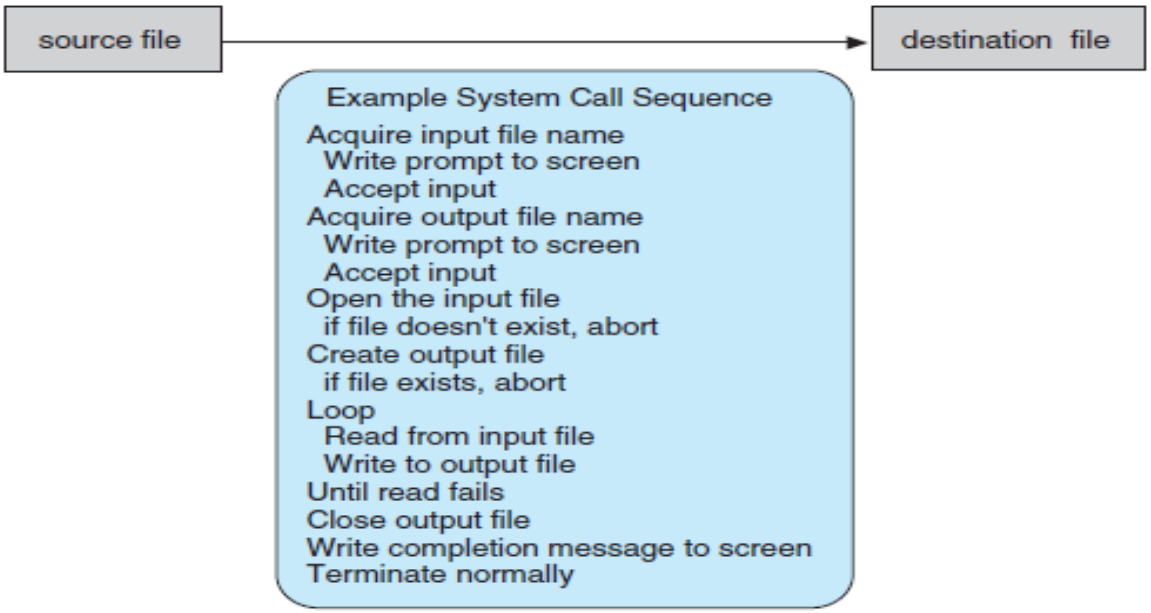

Figure 2.5 Example of how system calls are used.

- Behind the scenes, the functions that make up an API typically invoke the actual system calls on behalf of the application programmer. For example, the Windows function CreateProcess( ) actually invokes the NTCreateProcess( ) system call in the Windows kernel.
- Why would an application programmer prefer programming according to an API rather than invoking actual system calls?
- There are several reasons for doing so:
- **Program portability**, an application programmer designing a program using an API can expect her program to compile and run on any system that supports the same API.
- Furthermore, actual system calls can often be more detailed and difficult to work with than the API available to an application programmer.
- For most programming languages, the run-time support system (a set of functions built into libraries included with a compiler) provides 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.

- Typically, a number is associated with each system call, and the system-call interface maintains a table indexed according to these numbers. The system call interface then invokes the intended system call in the operating-system kernel and returns the status of the system call and any return values.
- Thus, most of the details of the operating-system interface are hidden from the programmer by the API and are managed by the run-time support library. The relationship between an API, the system-call interface, and the operating system is shown in Figure 2.6,

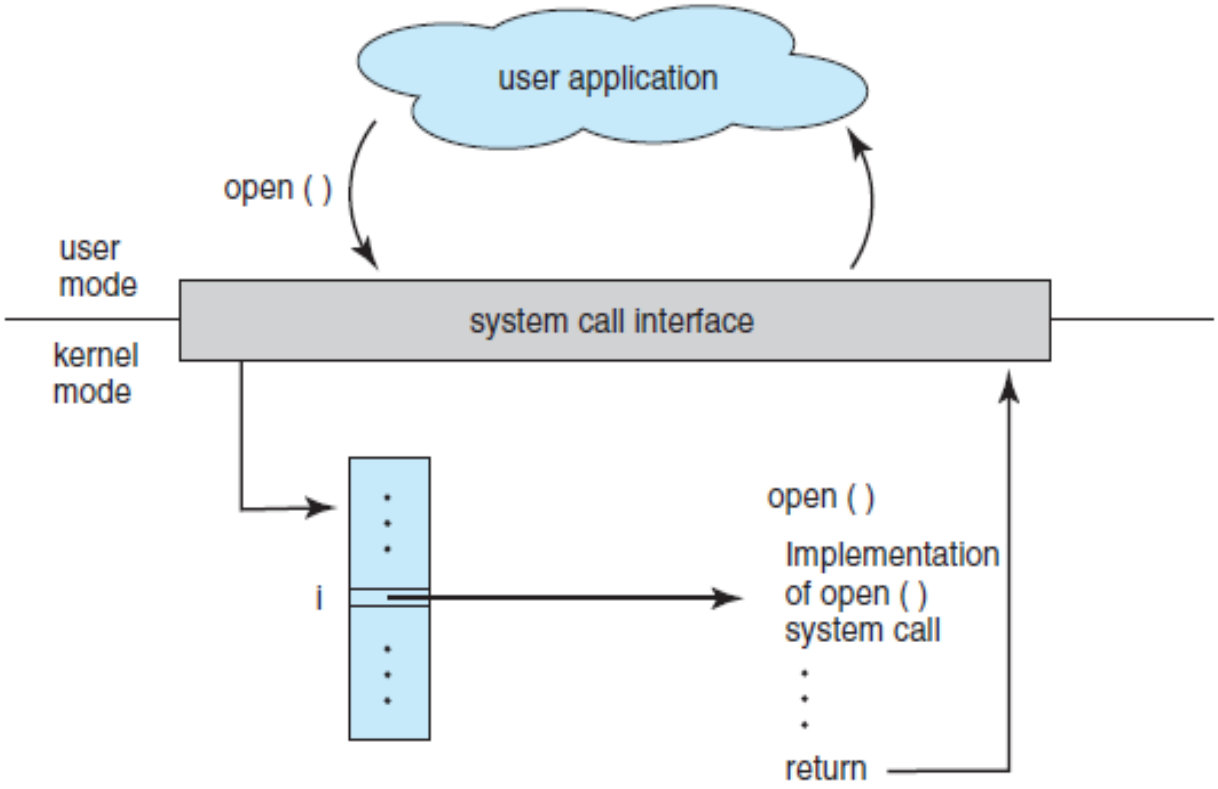

Figure 2.6 The handling of a user application invoking the `open()` system call.

Types of System Calls

- System calls can be grouped roughly into six major categories:
- **Process control.**
- **File manipulation.**
- **Device manipulation.**

- **Information maintenance.**
- **Communications.**
- **Protection.**

- Process control
  - 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
- Device management
  - 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

Figure 2.8 Types of system calls.

EXAMPLE OF STANDARD C LIBRARY

The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. As an example, let's assume a C program invokes the `printf()` statement. The C library intercepts this call and invokes the necessary system call (or calls) in the operating system—in this instance, the `write()` system call. The C library takes the value returned by `write()` and passes it back to the user program. This is shown below: