Unit I
System Structures

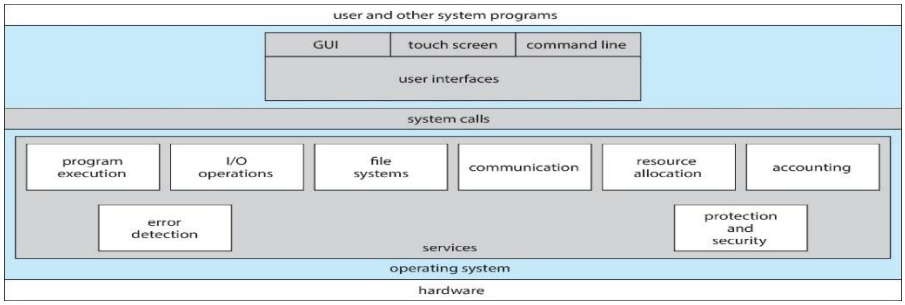
1. **Operating-System Services**
2. **User and Operating-System Interfaces**
   i. **Command Interpreters**
   ii. **Graphic User Interfaces**
   iii. **Choice of Interface**
3. **System Calls**
4. **Types of System Calls**
   i. **Process Control**
   ii. **File Management**
   iii. **Device Management**
   iv. **Information Maintenance**
   v. **Communication**
   vi. **Protection**
5. **System Programs**
6. **Operating-System Design and Implementation**
   i. **Design Goals**
   ii. **Mechanisms and Policies**
   iii. **Implementation**

System Structures

1. Operating-System Services

Operating systems provide an environment for execution of programs and services to programs and users.

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).
    - ✓ Varies between Command-Line (CLI), Graphics User Interface (GUI), touch-screen
  - **Program execution** - The system must be able to load a program into memory and to run that program, end 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** - The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.
  - **Communications** – Processes may exchange information, on the same computer or between computers over a network
    - ✓ Communications may be via shared memory or through message passing (packets moved by the OS)
  - **Error detection** – OS needs to be constantly aware of possible errors
    - ✓ May occur in the CPU and memory hardware, in I/O devices, in user program
    - ✓ For each type of error, OS should take the appropriate action to ensure correct and consistent computing
    - ✓ Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system
- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
  - **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - ✓ Many types of resources - CPU cycles, main memory, file storage, I/O devices.
  - **Logging** - To keep track of which users use how much and what kinds of computer resources
  - **Protection and security** - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
    - ✓ Protection involves ensuring that all access to system resources is controlled
    - ✓ Security of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts

2. User and Operating-System Interfaces

There are several ways for users to interface with the operating system. One allows users to directly enter commands to be performed by the operating system. The other allows users to interface with the operating system via a graphical user interface, or GUI.

i. Command Interpreters

- CLI allows direct command entry
- Some operating system include the command interpreter in the kernel.
- Others, such as Windows and UNIX, treat the command interpreter as a special program that is running when a job is initiated or when a user first logs on.
- The main function of the command interpreter is to get and execute the next user-specified command.

ii. Graphic User Interfaces

- User-friendly desktop metaphor interface
  - ✓ Usually mouse, keyboard, and monitor
  - ✓ Icons represent files, programs, actions, etc
  - ✓ Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a folder)
  - ✓ Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
  - ✓ Microsoft Windows is GUI with CLI "command" shell
  - ✓ Apple Mac OS X is "Aqua" GUI interface with UNIX kernel underneath and shells available
  - ✓ Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)
- Touchscreen devices require new interfaces
  - ✓ Mouse not possible or not desired
  - ✓ Actions and selection based on gestures
  - ✓ Virtual keyboard for text entry

iii. Choice of Interface

- The choice of whether to use a command-line or GUI interface is mostly one of personal preference.
- System administrators who manage computers and power users who have deep knowledge of a system frequently use the command-line Interface. For them, it is more efficient, giving them faster access to the activities they need to perform.
- In contrast, most Windows users are happy to use the Windows GUI environment and almost never use the MS-DOS shell interface.
- The user interface can vary from system to system and even from user to user within a system.

3. System Calls

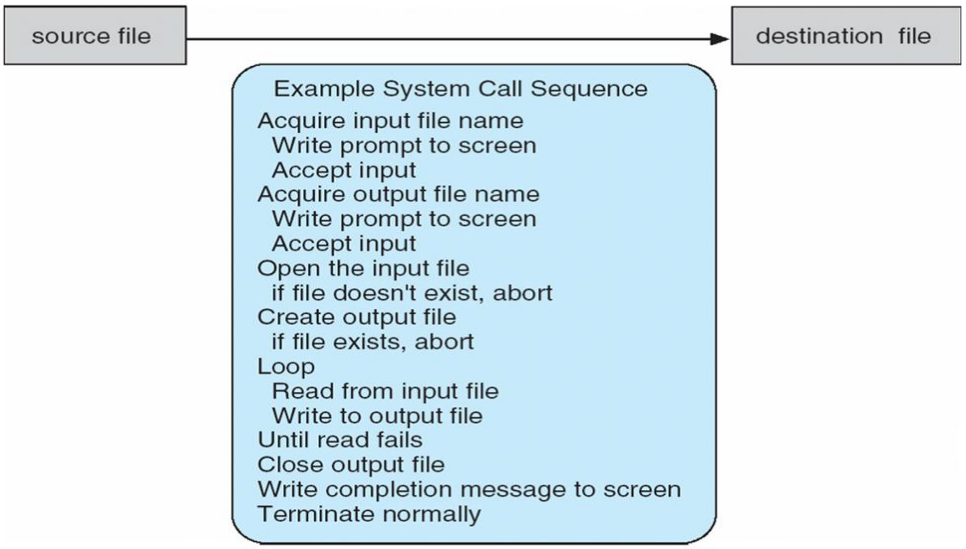
- System calls provide an interface to the services made available by an operating system.
- These calls are generally available as routines written in C and C++, although certain low-level tasks may have to be written using assembly-language instructions.

3.1 Example

Writing a simple program to read data from one file to another file.

1. The first input that the program need is the names of the two files: the input file and the output file. The names can be specified in many ways depending on the operating system design.
   a. One approach is to pass the names of the two files as part of the command.
      cp file1.txt file2.txt
      This command copies the input file **file1.txt** to the output file **file2.txt**.
   b. A Second approach is for the program to ask the user for the names. In a interactive system, this approach will require a sequence of system calls
      i. First to write a prompting message on the screen.
      ii. and then read the keyboard the characters that defines the two files.
   c. 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.
2. Once the two file names have been obtained, the program must open the input file and create and open the output file. Each of these operations require another system call.
3. Possible error conditions for each system call must be handled. For example, when the program tries to open the input file, 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 output an error message and then terminate abnormally



3.2 Application Programming Interface

- Frequently, system execute thousands of system calls per second. Most programmers never see this level of detail, however.
- 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.

- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM).
- Example of Standard API

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t     read(int fd, void *buf, size_t count)
```

return value — function name — parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

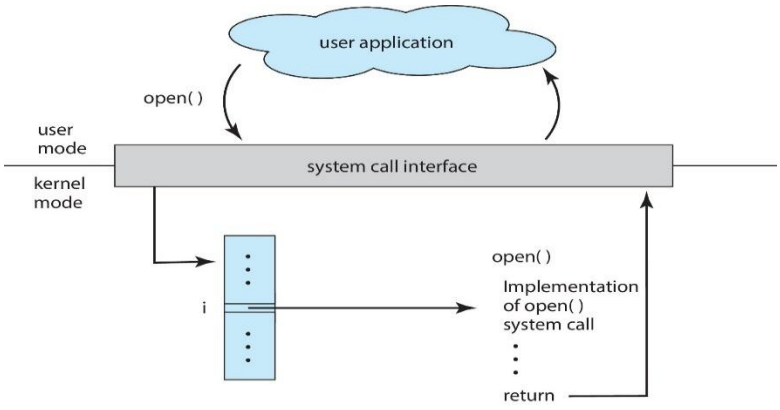
- `int fd`—the file descriptor to be read
- `void *buf`—a buffer into which the data will be read
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns –1.

System Call Implementation

- Typically, a number is associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API
  - Managed by run-time support library (set of functions built into libraries included with compiler)

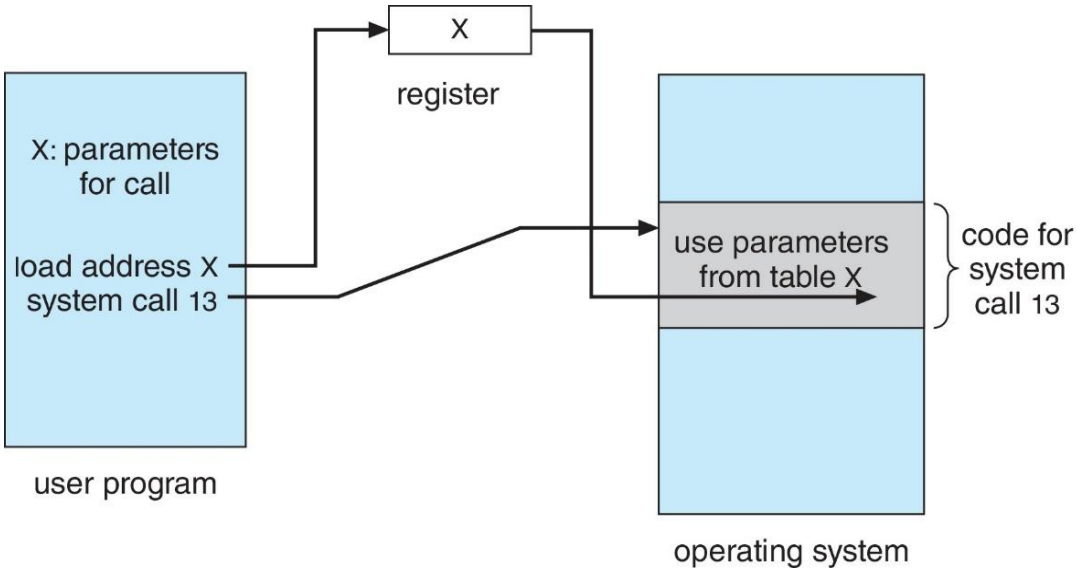
API System Call OS Relationship



System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - **Simplest**: pass the parameters in registers. In some cases, may be more parameters than registers
  - Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register. This approach taken by Linux and Solaris
- Parameters placed, or pushed, onto the stack by the program and popped off the stack by the operating system
- Block and stack methods do not limit the number or length of parameters being passed

Parameter Passing via Table



4. Types of System Calls

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

i. Process Control

- If a system call is made to terminate the currently running program abnormally, or if the program runs into a problem and causes an error trap, a dump of memory is sometimes taken and an error message generated.
- The dump is written to disk and may be examined by a debugger – errors, or bugs – to determine the cause of the problem.
- Process control do the following:
  - create process, terminate process
  - end (halt program execution normally) , abort (halt program execution annormally)
  - load and execute programs
  - get process attributes, set process attributes
  - wait for time to finish execution of currently executing jobs
  - wait event, signal event
  - allocate and free memory
  - Locks for managing access to shared data between processes

ii. File Management

- create() and delete() files. Either system call requires the name of the file or some of the file's attributes.
- Once the file is created, we need to open() it and to use it.
- We may also read(), write(), or reposition(). Finally, we need to close() the file close file, indicating that we are no longer using it.
- To determine the values of various attributes and perhaps to reset them if necessary. Two system calls get_file_attributes() and set_file_attributes() are required for this function.

iii. 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. Otherwise, the process will have to wait until sufficient resources are available. The various resources controlled by the operating system can be thought of as devices.
- A system with multiple users may require us to first request() a device, to ensure exclusive use of it.
- Once the device has been requested, we can read(), write(), and reposition() the device.
- After we are finished with the device, we release() it.
- get device attributes, set device attributes
- logically attach or detach devices

iv. Information Maintenance

- Many system calls exist simply for the purpose of transferring information between user program and the operating system.
- Most systems have a system call to return the current time() and date().
- Other system calls may return information about the system, such as the number of current users, the version number of the operating system, the amount of free memory or disk space, and so on.

Example: get system data and set system data

- Many operating systems provide calls to dump() memory.
- The operating system keeps information about all its processes and system calls are used to access this information

Example: get_process_attributes() and set_process_attributes()

v. 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.
- Messages can be exchanged between the processes either directly or indirectly through a common mailbox.
- In the shared-memory model, processes use shared_memory_create() and shared_memory_attach() system calls to create and gain access to regions of memory owned by other processes.

vi. Protection

- Protection provides a mechanism for controlling access to the resources provided by a computer system.
- System calls providing protection include set_permission() and get_permission(), which manipulate the permission settings of resources such as files and disks.
- The allow_user() and deny_user() system calls specify whether particular users can or cannot be allowed access to certain resources.

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

The following illustrates various equivalent system calls for Windows and UNIX operating systems.

	Windows	Unix
Process control	`CreateProcess()`	`fork()`
	`ExitProcess()`	`exit()`
	`WaitForSingleObject()`	`wait()`
File management	`CreateFile()`	`open()`
	`ReadFile()`	`read()`
	`WriteFile()`	`write()`
	`CloseHandle()`	`close()`
Device management	`SetConsoleMode()`	`ioctl()`
	`ReadConsole()`	`read()`
	`WriteConsole()`	`write()`
Information maintenance	`GetCurrentProcessID()`	`getpid()`
	`SetTimer()`	`alarm()`
	`Sleep()`	`sleep()`
Communications	`CreatePipe()`	`pipe()`
	`CreateFileMapping()`	`shm_open()`
	`MapViewOfFile()`	`mmap()`
Protection	`SetFileSecurity()`	`chmod()`
	`InitlializeSecurityDescriptor()`	`umask()`
	`SetSecurityDescriptorGroup()`	`chown()`

5. System Programs/Services

- System programs, also known as system utilities provide a convenient environment for program development and execution.
- They can be divided into:
  - **File management**: These programs create, delete, copy, rename, print, dump, list and generally manipulate files and directories.
  - **Status information:** Some ask the system for info - date, time, amount of available memory, disk space, and number of users. Others provide detailed performance, logging, and debugging information.
  - **Programming language support:** Compilers, assemblers, debuggers, and interpreters for common programming languages are often provided with the operating system.

- **Program loading and execution:** Once a program is assembled or compiled, it must be loaded into memory to be executed.
- **Communications:** These programs provide the mechanism for creating virtual connections among processes, users, and computer systems. Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another
- **Background services:** All general-purpose systems have methods for launching certain system-program processes at boot time. Some of these processes terminate after completing their tasks, while others continue to run until the system is halted.
- **Application programs:** Most operating systems are supplied with programs that are useful in solving common problems or performing common operations. Such application programs include, Web browsers, word processors, spreadsheets and database systems

6. Operating-System Design and Implementation

i. Design Goals

- The design of the system will be affected by the choice of hardware and the type of system: batch, time sharing, single user, multiuser, distributed, real time, or general purpose.
- Requirements can be divided into two basic groups: user goals and system goals
  - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient
- Specifying and designing an OS is a highly creative task of software engineering.

ii. 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.
- For example, the timer construct is a mechanism for ensuring CPU protection, but deciding how long the timer is to be set for a particular user is a policy
- The separation of policy and mechanism is important for flexibility. Policies are likely to change across places over time.

iii. Implementation

- Once an operating system is designed, it must be implemented. Because operating systems are collections of many programs, written by many people over a long period of time, it is difficult to make general statements about how they are implemented.
- Early operating systems were written in assembly language. Now most are written in a higher-level language such as C or an even higher-level language such as C++.
- An operating system can be written in more than one language. The lowest levels of the kernel might be assembly language. Higher-level routines might be in C, and system programs might be in C or C++, in interpreted scripting languages like PERL or Python.
- The advantages of using a higher-level language for implementing operating systems are: the code can be written faster, is more compact, and is easier to understand and debug.
- The only possible disadvantages of implementing an operating system in a higher-level language are reduced speed and increased storage requirements.