IV USER AND OPERATING SYSTEM INTERFACE

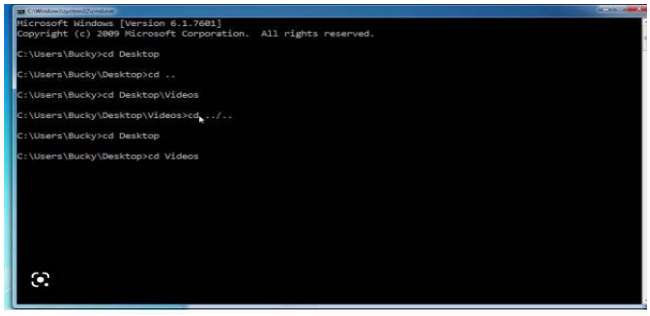
1. User Interfaces

There are different types of user interfaces each of which provides a different functionality:

- Command Based Interface
- Graphical User Interface
- Touch Based Interface
- Voice Based Interface
- Gesture Based Interface

❖ Command Based Interface

Command based interface requires a user to enter the commands to perform different tasks like creating, opening, editing or deleting a file, etc. The user has to remember the names of all such programs or specific commands which the operating system supports. The primary input device used by the user for command based interface is the keyboard. Command-based interface is often less interactive and usually allows a user to run a single program at a time. Examples of operating systems with command-based interfaces include MS-DOS and Unix.



Command Based Interface

Graphical User Interface

❖ Graphical User Interface

Users run programs or give instructions to the computer in the form of icons, menus and other visual options. Icons usually represent files and programs stored on the computer and windows represent running programs that the user has launched through the operating system. The input devices used to interact with the GUI commonly include the mouse and the keyboard. Examples of operating systems with GUI interfaces include Microsoft Windows, Ubuntu, Fedora and Macintosh, among others.

❖ **Touch Based Interface**

Today smartphones, tablets, and PCs allow users to interact with the system simply using the touch input. Using the touchscreen, a user provides inputs to the operating system, which are interpreted by the OS as commands like opening an app, closing an app, dialing a number, scrolling across apps, etc. Examples of popular operating systems with touch-based interfaces are Android and iOS. Windows 8.1 and 10 also support touch-based interfaces on touchscreen devices.

Touch Based Interface

Voice Based Interface

❖ **Voice Based Interface**

Modern computers have been designed to address the needs of all types of users including people with special needs and people who want to interact with computers or smartphones while doing some other task. For users who cannot use input devices like the mouse, keyboard, and touchscreens, modern operating systems provide other means of human-computer interaction. Users today can use voice-based commands to make a computer work in the desired way. Some operating systems which provide voice-based control to users include iOS (Siri), Android (Google Now or "OK Google"), Microsoft Windows 10 (Cortana), and so on.

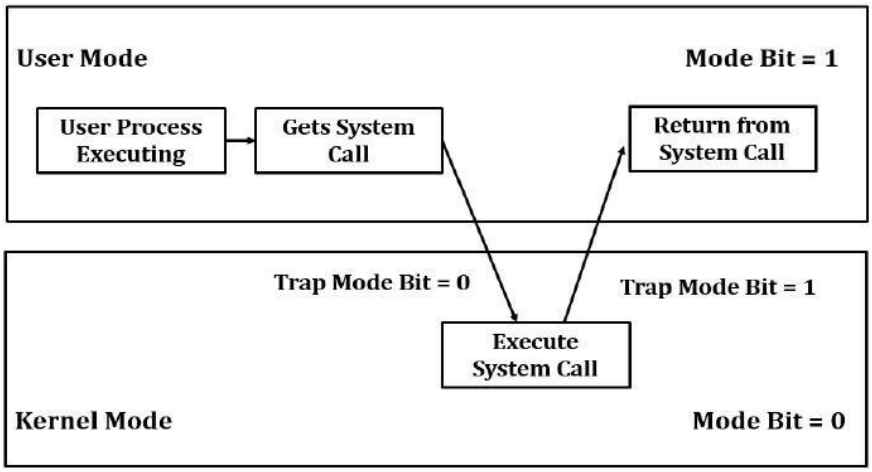
❖ **Gesture Based Interface**

Some smartphones based on Android and iOS as well as laptops let users interact with the devices using gestures like waving, tilting, eye motion, and shaking. This technology is evolving faster and it has promising potential for application in gaming, medicine, and other areas.

a. SYSTEM CALLS

A **system call** is a mechanism that provides the interface between a process and the operating system. It is a programmatic method in which a computer program requests a service from the kernel of the OS. System call offers the services of the operating system to the user programs via API (Application Programming Interface). System calls are the only entry points for the kernel system.

Working of System Call



Step 1) The processes executed in the user mode till the time a system call interrupts it.

Step 2) After that, the system call is executed in the kernel-mode on a priority basis.

Step 3) Once system call execution is over, control returns to the user mode.,

Step 4) The execution of user processes resumed in Kernel mode.

Need of System Call

- Reading and writing from files demand system calls.
- If a file system wants to create or delete files, system calls are required.
- System calls are used for the creation and management of new processes.
- Network connections need system calls for sending and receiving packets.
- Access to hardware devices like scanner, printer, need a system call.

Example of how system calls are used.

For Example

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 to pass the names of the two files as part of the command For

example, the UNIX cp command:

cp in.txt out.txt

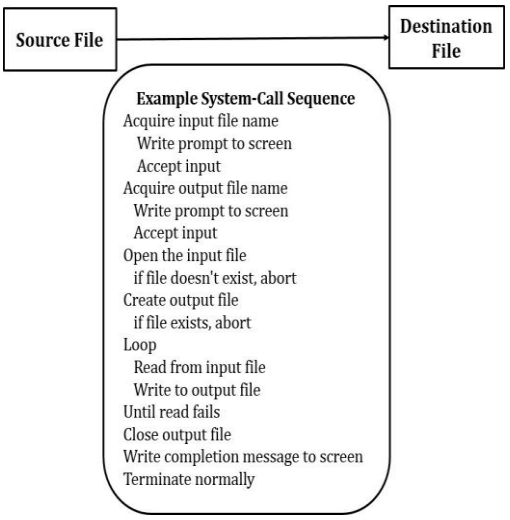
This command copies the input file in.txt to the output file out.txt.

A second 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 and open the output file. Each of these operations requires another system call. 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.

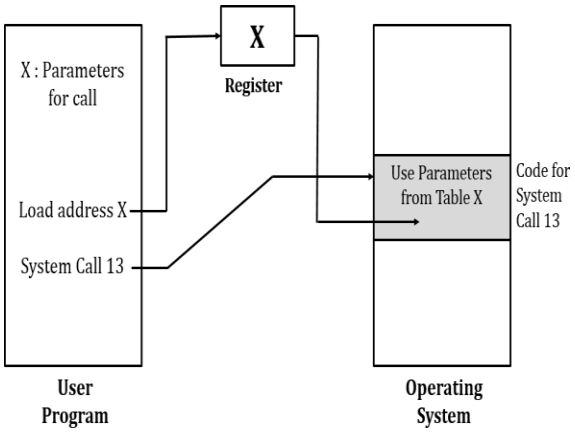
In these cases, the program should output an error message (another sequence of system calls) and then terminate abnormally (another system call). If the input file exists, then we must create a new output file. We may find that there is already an output file with the same name. This situation may cause 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 available disk space).

Finally, after the entire file is copied, the program may close both files (two system calls), write a message to the console or window (more system calls), and finally terminate normally (the final system call).

Passing of Parameters as a table



Rules for passing Parameters for System Call

Here are general common rules for passing parameters to the System Call:

- Parameters should be pushed on or popped off the stack by the operating system.
- Parameters can be passed in registers. For five or fewer parameters, registers are used.
- More than five parameters, the block method is used. 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

Types of System calls

Here are the five types of System Calls in OS:

- ❖ Process Control
- ❖ File Management
- ❖ Device Management
- ❖ Information Maintenance
- ❖ Communications

Process Control

This system calls perform the task of process creation, process termination, etc.

Functions:

- End and Abort
- Load and Execute
- Create Process and Terminate Process
- Wait and Signal Event
- Allocate and free memory

File Management

File management system calls handle file manipulation jobs like creating a file, reading, and writing, etc.

Functions:

- Create a file
- Delete file

- Open and close file
- Read, write, and reposition
- Get and set file attributes

Device Management

Device management does the job of device manipulation like reading from device buffers, writing into device buffers, etc.

Functions:

- Request and release device
- Logically attach/ detach devices
- Get and Set device attributes

Information Maintenance

It handles information and its transfer between the OS and the user program.

Functions:

- Get or set time and date
- Get process and device attributes

Communication:

These types of system calls are specially used for interprocess communications.

Functions:

- Create, delete communications connections
- Send, receive message
- Help OS to transfer status information
- Attach or detach remote devices

Important System Calls Used in OS wait()

A process needs to wait for another process to complete its execution. This occurs when a parent process creates a child process, and the execution of the parent process remains suspended until its child process executes. The suspension of the parent process automatically occurs with a wait() system call. When the child process ends execution, the control moves back to the parent process.

fork()

Processes use this system call to create processes that are a copy of themselves. With the help of this system Call parent process creates a child process, and the execution of the parent process will be suspended till the child process executes.

exec()

This system call runs when an executable file in the context of an already running process that

replaces the older executable file. However, the original process identifier remains as a new process is not built, but stack, data, head, data, etc. are replaced by the new process.

kill():

The kill() system call is used by OS to send a termination signal to a process that urges the process to exit. However, a kill system call does not necessarily mean killing the process and can have various meanings.

exit():

The exit() system call is used to terminate program execution. Specially in the multi- threaded environment, this call defines that the thread execution is complete. The OS reclaims resources that were used by the process after the use of exit() system call.

2. SYSTEM PROGRAMS

System programs provide a convenient environment for program development and execution. It can be divided into:

- File manipulation
- Status information
- File modification
- Programming language support
- Program loading and execution
- Communications
- Application programs

- **File management**.

These programs create, delete, copy, rename, print, list, and generally access and manipulate files and directories.

- **Status information**.

Some programs simply ask the system for the date, time, amount of available memory or disk space, number of users, or similar status information. Others are more complex, providing detailed performance, logging, and debugging information. Typically, these programs format and print the output to the terminal or other output devices or files or display it in a window of the GUI. Some systems also support a **registry**, which is used to store and retrieve configuration information.

- **File modification:** .

Several text editors may be available to create and modify the content of files stored on disk or other storage devices. There may also be special commands to search contents of files or perform transformations of the text.

- **Programming-language support**:

Compilers, assemblers, debuggers, and interpreters for common programming languages (such as C, C++, Java, and Python) are often provided with the operating system or available as a separate download.

- **Program loading and execution**:

Once a program is assembled or compiled, it must be loaded into memory to be executed. The system may provide absolute loaders, relocatable loaders, linkage editors, and overlay loaders. Debugging systems for either higher-level languages or machine language are needed as well.

- **Communications**:

These programs provide the mechanism for creating virtual connections among processes, users, and computer systems. They allow users to send messages to one another's screens, to browse web pages, to send e-mail messages, to log in remotely, or to 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. Constantly running system-program processes are known as services, subsystems, or daemons