2...

System Structure

Objectives...

- To learn about different user Operating System Interfaces.
- To know about Concept of System Calls.
- To study about the System Program.

2.1 USER OPERATING SYSTEM INTERFACE

- Almost all operating systems have User Interface (UI). This interface can take several Forms one is Command-Line User Interface, Batch Interface, Menu Driven User Interface and Graphical User Interface (GUI).
- The importance of user interfaces are:
  - To assist users interacting with software.
  - To control how a user enters data and instructions.
  - To control how information is displayed.

Types of User Interface:

1. Command-Line User Interface(CLI)

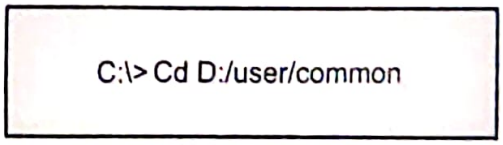

Fig. 2.1 (a): Example of CLI

- The Command-line user interface requires a user to type commands or press special keys on the keyboard to enter data and instructions that instruct the Operating system what to do. It has to be typed one line at a time.
- The Command-line user interface also requires memorization. The advantage of Command-line interface is that it helps the user to operate the computer quickly after memorizing the keywords and syntax.
- Command line interfaces are also called command-line user interfaces, console user interfaces and character user interfaces.

2. Menu Driven User Interface:

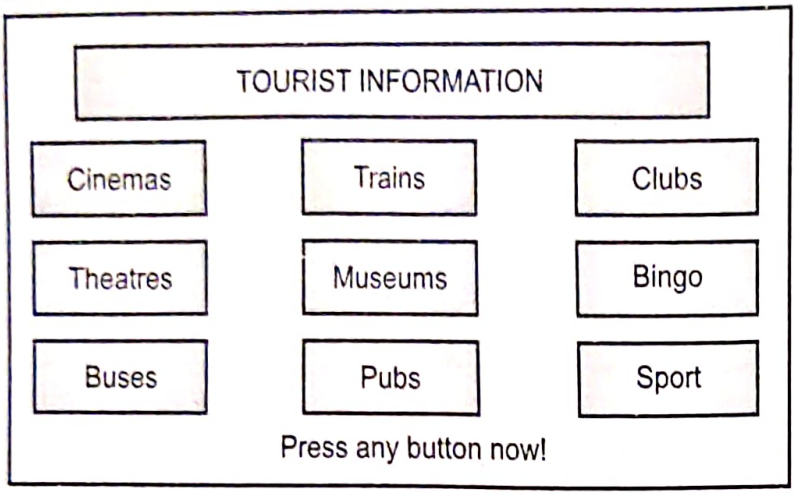

Fig. 2.1 (b): Example of Menu driven user interface

- It avoids memorizing the commands. Also it is much easier than Command Line Interface. Here user is displayed with menus and submenus by clicking the appropriate option specific tasks carried out.

3. Graphical User Interface:

- It is a software interface that user interact with a pointing device, such as mouse. These are very user friendly because they are easy to interact with system and to execute instructions. For example, when you are using Internet you are looking at the GUI of web browser.

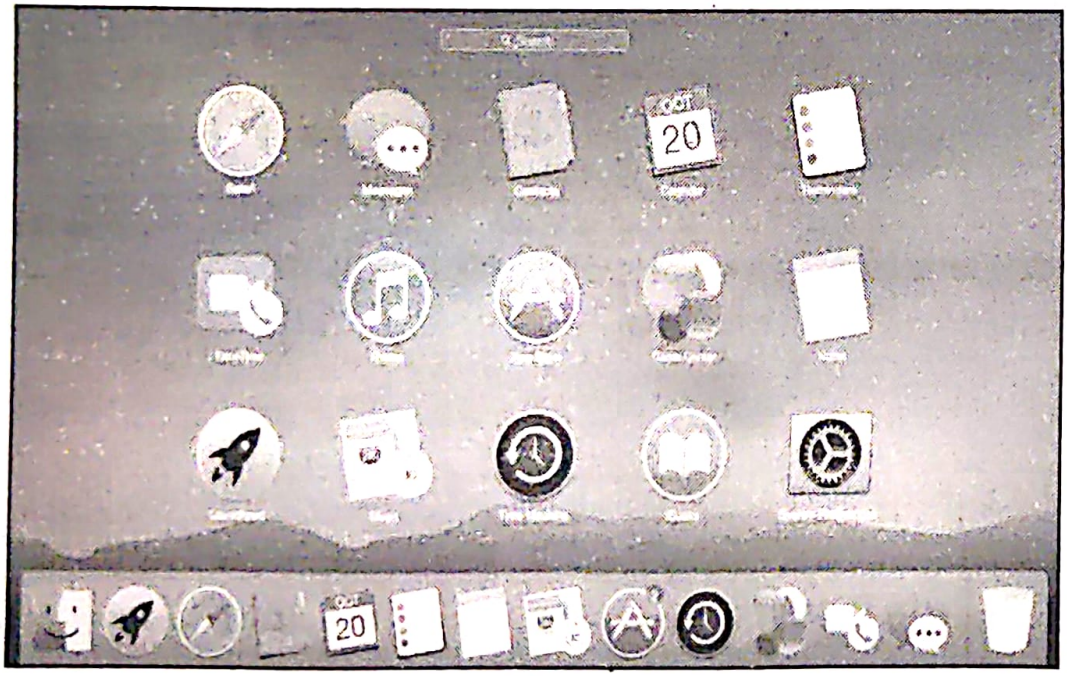

Fig. 2.1 (c): Example of GUI

Basic Components of a GUI:

- Graphical user interfaces, such as Microsoft Windows and the one used by the Apple Macintosh, feature the following basic components:
  - **Pointer:** A symbol that appears on the display screen and that you move to select objects and commands. Usually, the pointer appears as a small angled arrow. Text processing applications, however, use an I-beam pointer that is shaped like a Capital I.

- **Pointing device:** A device, such as a mouse or trackball that enables you to select objects on the display screen.
- **Icons:** Small pictures that represent commands, files or windows. By moving the pointer to the icon and pressing a mouse button, you can execute a command or convert the icon into a window. You can also move the icons around the display screen as if they were real objects on your desk.
- **Desktop:** The area on the display screen where icons are grouped is often referred to as the desktop because the icons are intended to represent real objects on a real desktop.
- **Windows:** You can device the screen into different areas. In each window, you can run a different program or display a different file. You can move windows around the display screen and change their shape and size at will.
- **Menus:** Most graphical user interfaces let you execute commands by selecting a choice from a menu.

4. Form-based interfaces:

- Uses text-boxes, text-areas, check-boxes etc. to create electronic form. User completes in order to enter data into the system.

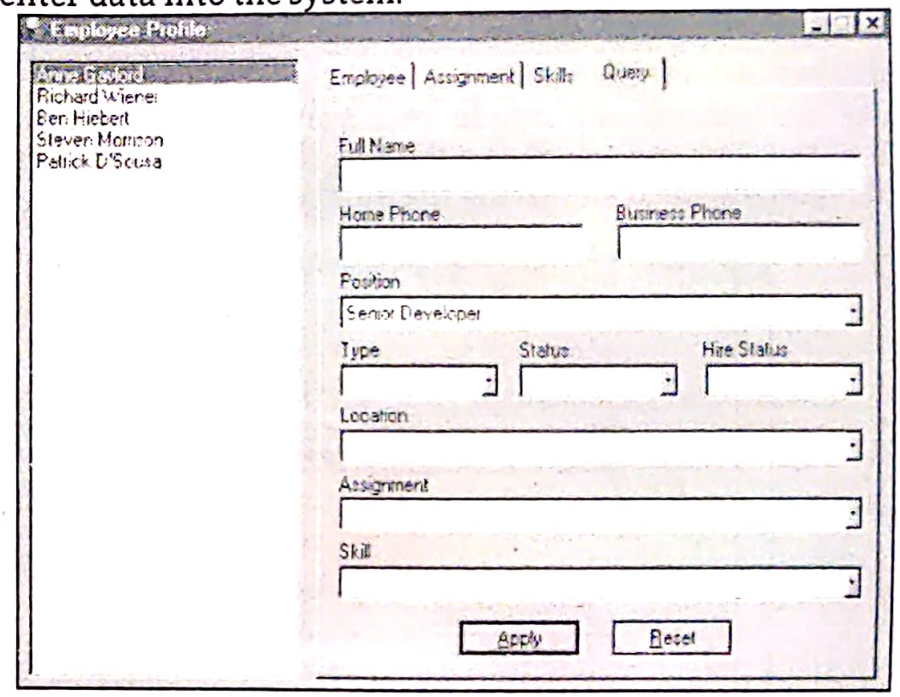

Fig. 2.1 (d): Example of a Form-based Interface

2.2 SYSTEM CALLS

- System calls are programming interface to the services provided by the operating system.
- A system call is a request by the user to the operating system to do something on user's behalf. System call provides an interface between a running program and an operating system.
- System calls are instructions that generate an interrupt that causes the operating system to gain control of the processor.
- When a program makes a system call, the mode is switched from user mode to kernel mode. This is called a **Context Switch**.

- Typically, system calls written in a high level languages like C or C++.
- The operating system then determines what kind of system call it is and performs the appropriate services for the system caller.
- For example, for I/O a process involves a system call telling the operating system to read or write particular area and this request is satisfied by the operating system.

2.2.1 How to make a System Call?

- A system call is made using the system call machine language instruction. These calls are generally available as assembly language instructions and are usually listed in the manuals used by assembly - language programmers.
- Certain systems allow system calls to be made directly from a higher language program, in which case the calls normally resemble predefined function or subroutine calls. They may generate a call to a special run-time routine that makes the system call.
- Several languages such as C, C++ have been defined to replace assembly language for systems programming. These languages allow system calls to be made directly.
- For example, UNIX system calls may be invoked directly from a C or C++ program. System calls for Microsoft Windows Platforms are part of the Win32 API, which is available for use by all the compilers written for Microsoft Windows.
- Steps involved in making a system call:
  - We will take the example of a simple 'read system call.' It has three parameters along with it, the first one specifying the file, the second specifying the pointing to the buffer, and the third one giving the number of bytes to read.

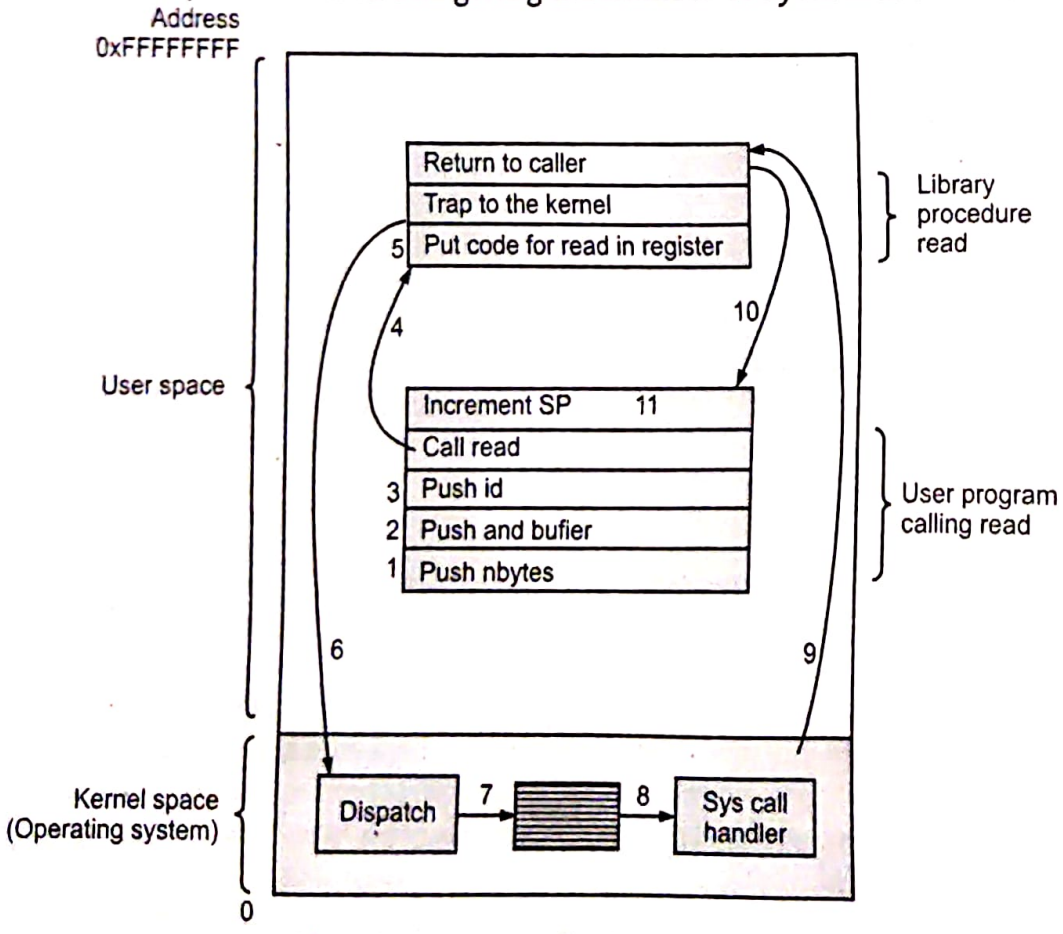

Fig. 2.2: Steps to make System Call

```
count= read(fd, buffer, nbytes)
```

- The count consists of total number of bytes to read and if some error occurs it is set to –1 which is also indicated by a global variable *errno*. The program should check this error variable timely to check if any error has occurred. The 1st and the 3rd parameters are passed by value and the 2nd parameter is passed by reference i.e. the buffer address is passed.

Steps 1-3 : The calling program pushes the parameters onto the stack.

Step 4 : The actual call to the library procedure is made. This instruction is the normal procedure call instruction used to call all procedures.

Step 5 : The library procedure places the system call number into the register.

Step 6 : The library procedure executes a TRAP instruction to switch from user mode to kernel mode and start execution at a fixed address within the kernel.

Step 7 : The kernel examines the system call number and then dispatches it to the correct system call handler. This correct number is given in the table of system call handlers by pointers referenced at the system call number.

Step 8 : The system call handler works.

Step 9 : The operation is completed, and the user is given back control once the TRAP instruction is set.

Step 10 : Then the control is transferred to the user program from the read procedure

Step 11 : Finally, SP is incremented to clean up the stack. In this way the job of read system call is completed.

Types of System Calls:

- System calls can be roughly grouped into following major categories:
  1. Process or Job Control.
  2. File Management.
  3. Device Management.
  4. Information Maintenance.
- In the following sections, we will see details of each category.

(i) File and I/O System Calls:

open	Get reading to read or write a file.
create	Create a new file and open it.
read	Read bytes from an open file.
write	Write bytes to an open file.
close	Indicate that you are done reading or writing a file.

(ii) Process Management System Calls:

create process	Create a new process
exit	Terminate the process making the system call
wait	Wait for another process to exit
fork	Create a duplicate of the process working the system call
execv	Run a new program in the process making the system call

(iii) Interprocess Communication System calls:

createMessageQueue	Create a queue to hold messages
SendMessage	Send a message to a message queue
ReceiveMessage	Receive a message from a message queue

2.3 PROCESS OR JOB CONTROL [S-18, 23]

- Process is nothing but program in execution. The execution process must be in sequential manner.
- A running program needs to be able to halt its execution either normally (end) or abnormally (abort).
- If the program discovers an error in its input and wants to terminate abnormally, it may also want to define an error level.
- A process or job executing one program may want to load and execute another program.
- This allows the control card interpreter to execute program as directly by the control cards of the user job.
- If we create a new job or process, we should be able to control its execution. We may also want to terminate a job or process that we created (terminate process).
- If we find that it is incorrect or no longer needed we may require waiting time to finish execution (wait time). Another set of system calls are helpful in debugging a program.

System calls related to Process Control are:

(i) End, Abort.
(ii) Load, Execute.
(iii) Create process, Terminate process.
(iv) Ready process, Dispatch process.
(v) Suspend process, Resume process.
(vi) Get process attributes, Set process attributes.
(vii) Wait for Time.
(viii) Wait event, Signal event.
(ix) Change the priority of a process.

2.4 DEVICE MANAGEMENT

- Files can be thought of as abstract or virtual devices. Thus, many of the system calls for files are also needed for devices.
- If there are multiple users of the system however, the users must first request the device to ensure that they have an exclusive use of it.
- After the users are finished with the device, they must release it. These functions are similar to the open/close system calls for files.
- Once, the device has been requested the users can read, write and reposition the device just as with files.

- In fact, the similarity between input/output devices and files is so great that many operating systems merge the two into a combined file/device structure. In this case input/output devices are identified by special file names.

System calls related to Device Management are:

(i) Request a device, Release a device.

(ii) Read, Write, Reposition.

(iii) Get device attributes, Set device attributes.

(iv) Logically attach or detach devices.

2.5 FILE MANAGEMENT

- Operating System provides several common system calls dealing with files.
- Users need to be able to create and delete files. Such system calls require the name of the file and perhaps some of its attributes. Once the file is created, the users need to open it and use it.
- They may also need to read, write and reposition a file. Finally, they need to close the file, indicating that they are no longer using it.
- The users may need the same sets of operations for directories if there is a directory structure in the file system.
- In addition, for either files or directories, users need to be able to determine the values of various attributes and perhaps reset them if necessary.
- File attributes include the file name, file type, protection codes, accounting information and so on. Two system calls get file attributes and set file attributes are required for this function.

System calls for File Manipulation are:

(i) Create a file, Delete a file.

(ii) Open a file, Close a file.

(iii) Create a directory.

(iv) Read, Write, Reposition.

(v) Get file attributes, Set file attributes.

(vi) Create a link.

(vii) Change the working directory.

2.6 INFORMATION MAINTENANCE

- Many system calls exist simply for the purpose of transferring information between the user program and the operating system.
- For example, 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.
- In addition the operating system keeps information about all of its jobs and processes and there are system calls to access this information.
- Generally, there are also calls to reset it, (get process attributes and set process attributes).

System calls related to Information Maintenance are:

(i) Get Time or Date, Set Time or Date.

(ii) Get system Data, Set system Data.

(iii) Get process, File or Device attributes.

(iv) Set process, File or Device attributes.

2.7 COMMUNICATION

- These system calls are useful for Inter Process Communication (IPC). They also deal with creating and deleting a communication connection.
- There are two common models of IPC, one is the Message-passing Model and another is the Shared Memory Model.
- Message-passing Model uses a common mailbox to pass messages between processes.
- Shared memory Model use certain system calls to create and gain access to create and gain access to regions of memory owned by other processes. The two processes exchange information by reading and writing in the shared data.
- The operating system and the various active application programs are only guaranteed to interact easily if the individual processes are well-coordinated with one another. For this reason, communication via relevant system calls is essential.

System calls related to Communication are:

(i) Create, delete communication connection.

(ii) Send, receive messages.

(iii) Transfer status information.

(iv) Attach and detach remote services.

2.8 SYSTEM CALL IMPLEMENTATION

- Basically, a number is associated with each system call. It is used to number the system calls.
- System call interface maintains a table indexed according to these numbers. The system call interface invokes intended system call is operating system kernel and return status of the system call and any return values.
- The caller needs to know nothing about how the system call is implemented. Just need to obey API and understand what operating system will do as a result call.
- Most details of operating system interface are hidden from programmer by API. It is managed by run-time support library.
- Most of the systems calls are available in assembly language. Some systems may allow system calls to be written in higher-level language program, in which the calls normally like predefined function or subroutine calls.
- They may be generated call to a special run-time routine that makes the system calls, or the system call may be generated directly in line.

- **Example:** Consider two files: Input file and Output file. The program is to read data from input file and copy into output file. To execute this program, the sequence of system calls occurs as follows:
  1. To obtain two file names from keyboard, system calls are required.
  2. Then system call is open input file and creates an output file.
  3. While opening, if file is not exist then program will print a message or terminate abnormally. This requires again system calls.
  4. If file exists, then for reading from the input file (a system call) and writing to output file (another system call) calls are required.
  5. For closing both files, we required two system calls.
- Some operating system provides API which invokes the actual system calls on behalf of the application programmer. Most of the programming languages provides system call interface, which intercepts function calls in the API and invokes system calls within operating system.
- The relationship between an API, the system call interface and the operating system is shown in Fig. 2.3.

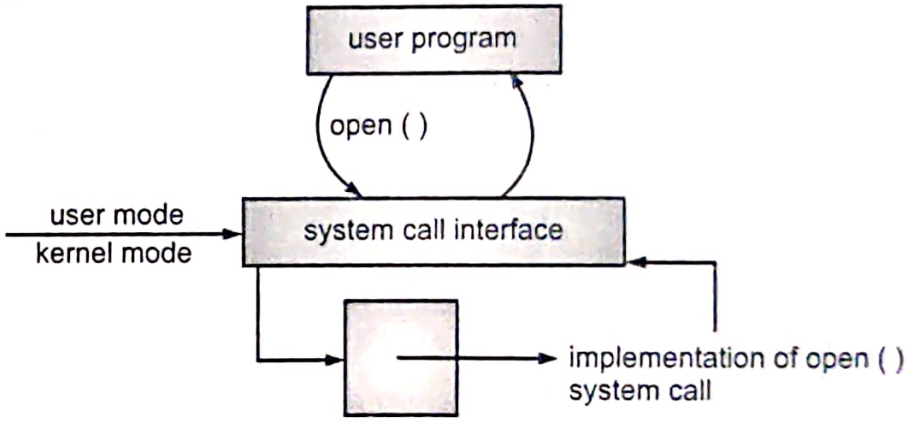

Fig. 2.3: Procedure to Invoke the system call

- The parameter can be passed by using registers to implement system call. Sometimes parameters are stored in a block in memory and the address of block or table is passed as a parameter in register. UNIX uses such a mechanism.
- Implementation of system call varies on different systems. Generally, a unique number identifies the type of system call and a system call is made by calling the particular number.
- Sometimes there is a need of sending some additional information along with, when the call is made. That is called as, parameter or parameter list of the system call [e.g. to open file, we need to specify file name and mode in which to open the file along with system call fopen( )].
- Special register is kept aside for the purpose of sending parameters. If parameters are too many in number and are beyond the capacity of the register to hold, parameters are stored in a block or table in the memory and the base address of it is passed as parameter through the register.
- Some programming languages like C permit system calls through program. To really understand what operating system does, we must examine these calls closely.

2.9 IMPORTANT SYSTEM CALLS

- **wait():**
  - In some systems, a process needs to wait for another process to complete its execution. This type of situation 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():**
  - The fork() system call is used to create processes. When a process (a program in execution) makes a fork() call, an exact copy of the process is created. Now there are two processes, one being the parent process and the other being the child process.
  - With the help of this system call, the parent process creates a child process and the execution of the parent process will be suspended till the child process executes.
- **exec():**
  - The exec() system call is also used to create processes. But there is one big difference between fork() and exec() calls. The fork() call creates a new process while preserving the parent process. But, an exec() call replaces the address space, text segment, data segment etc. of the current process with the new process.
- **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.

2.10 SUMMARY OF SYSTEM CALLS IN UNIX AND WINDOW

Table 2.1: Types of System calls in Unix and Windows

Types of System Calls	UNIX	Windows
Process control	fork() wait() exit()	CreateProcess() WaitForSingleObject() ExitProcess()
File manipulation	open() read() write() close()	CreateFile() ReadFile() WriteFile() CloseHandle()

contd. ...

Device manipulation	ioctl() read() write()	SetConsoleMode() ReadConsole() WriteConsole()
Information maintenance	getpid() alarm() sleep()	GetCurrentProcessId() SetTimer() Sleep()
Communication	pipe() shm_open() mmap()	CreatePipe() CreateFileMapping() MapViewOfFile()
Protection and Security	chmod() chown() umask()	SetFileSecurity() SetSecurityDescriptorGroup() IntializeSecurityDescriptor()

- **Example:** Consider 'C' fragment running on UNIX and invokes system call open() for a file. Initially the program is loaded into memory and starts the execution. The system uses fork() and exec() system calls. The configuration is shown in Fig. 2.4 (a).

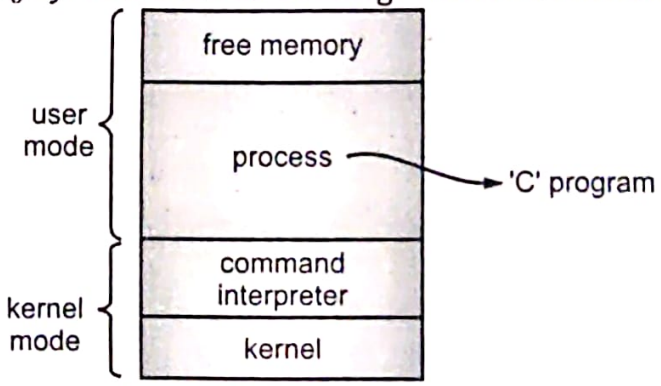

Fig. 2.4 (a): Running a program

- Let us assume 'C' program invoke a file opening statement. The Fig. 2.4 (b) shows how C library handles a open() calls.

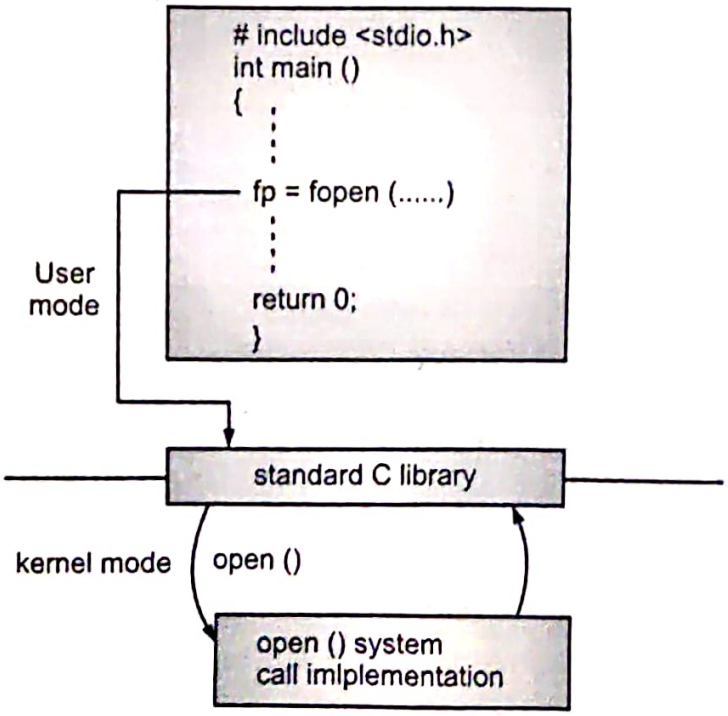

Fig. 2.4 (b): Handling a system call in 'C'

2.11 SYSTEM PROGRAM [S-18, S-19, W-22]

- System programs provide basic functioning to users so that they do not need to write their own environment for program development (editors, compilers) and program execution (shells).
- These programs act as an interface between the operating system and the application programs. They provide an environment in which application programs can be developed and executed in a convenient manner.
- These system programs can be divided into several categories:
  1. **File Manipulation:** These programs create, delete, copy, rename, print, dump, list and generally manipulate files and directories.
  2. **Status Information:** Some programs simply ask the operating system for the date, time, and amount of available memory or disk space, number of users or similar status information.
  3. **File Modification:** Several text editors may be available to create and modify the contents of files stored on a disk or a tape.
  4. **Programming Language Support:** Compilers, assemblers and interpreters for common programming languages. (Such as Fortran, Cobol, Pascal, Basic, C and so on) are often provided with the operating system.
  5. **Program Loading and Execution:** Once a program is assembled or compiled, it must be loaded into the memory to be executed. The operating system may provide absolute loaders, linkage editors and Debugging systems in order to achieve this.
  6. **Applications Programs:** In addition, most operating systems come with programs which are useful to solve some particularly common problems, such as Compilers, Text Formatters, Plotting Packages, Database Systems and so on.

2.12 CONCEPT OF OPERATING SYSTEM STRUCTURE

- This point we have already covered in first chapter. Please refer to Unit 1 point 1.5.

Summary

- Operating system provides interface to users to interact with applications through command line interface, menu driven interface, Graphical user Interface.
- System calls provide interface between the process and operating system. System calls are programming interface to the service provided by operating system. A system call is made using the system call machine language instruction. There are file and I/O system calls, Process management system calls, Inter-process communication system calls.
- Process or job control is program in execution it must be in sequential manner. Operating system provides several common system calls with files.
- System calls related to device management are request a device, release a device, read, write, reposition, get device attributes, set device attributes, logically attach or detach devices.

- User need to do basic operations on files, system calls for file manipulations are : create file, open file, delete file, close a file, Read and write file, create a directory.
- System programs provide basic functioning to users so that they do not need to write their own environment for program development (editors, compilers) and program execution (shells). These are categorized into file manipulation, status information, file modification, programming language support, program loading and execution, application programs.

Check Your Understanding

1. In Layered operating system which is the highest level?
   (a) Hardware (b) User Interface
   (c) Kernel (d) None of Above
2. What is the another name used for command line interpreter?
   (a) Shell (b) Kernel
   (c) Command (d) Prompt
3. Which system call is used for creating a file?
   (a) Read (b) Write
   (c) Open (d) Close
4. Which system call does not return control to calling point, on termination?
   (a) fork (b) exec
   (c) wait (d) exit
5. A fork system call will fail if _____.
   (a) The previously executed statement is also fork call
   (b) The limit on maximum no of processes in system would be executed
   (c) The limit on minimum no of processes in system that can under execution
   (d) All of above

Answers

1. (b)	2. (c)	3. (c)	4. (b)	5. (b)

Practice Questions

Q.I Answer the following questions in short:

1. Which are major types of system call?
2. What is interface? State its types?
3. Which are Information Maintenance system calls?
4. What is Device Management?
5. What is File Management?
6. Define operating system structure?

Q.II Answer the following questions:

1. Explain controls of GUI?
2. Explain the following system calls:
   (i) wait
   (ii) exit

3. What is meant by system calls? How to make it?
4. With neat diagram describe structure of operating system.
5. What is system program? What are its types? Explain in detail.
6. Write any two system calls of device manipulation.
7. Write system calls related to file management.

Q.III Define terms:

1. Shared-memory model
2. Message passing model
3. Process or job control
4. System call
5. System program

Previous Exam Questions

Summer 2018

1. What is Process? **[2 M]**

Ans. Refer to Section 2.3.

2. Define System Program. **[2 M]**

Ans. Refer to Section 2.11.

3. List and explain system calls related to Process and Job control. **[4 M]**

Ans. Refer to Section 2.3.

Summer 2019

1. Explain various types of system program. **[4 M]**

Ans. Refer to Section 2.11.

■■■