1) Basics of Operating Systems: Definition

"An operating system (OS) is system software that manages computer hardware and software resources and provides common services for computer programs".

2 Basic Functions of OS:

a) The Operating System as an Extended Machine

The architecture (instruction set, memory organization, I/O, and bus structure) of most computers at the machine-language level is awkward to program, especially for input/output.

One of the major tasks of the operating system is to hide the hardware and present programs (and their programmers) with nice, clean, elegant, consistent, abstractions to work with instead.

According to Andrew Tanenbaum 'Operating systems turn ugly hardware into beautiful abstractions'.

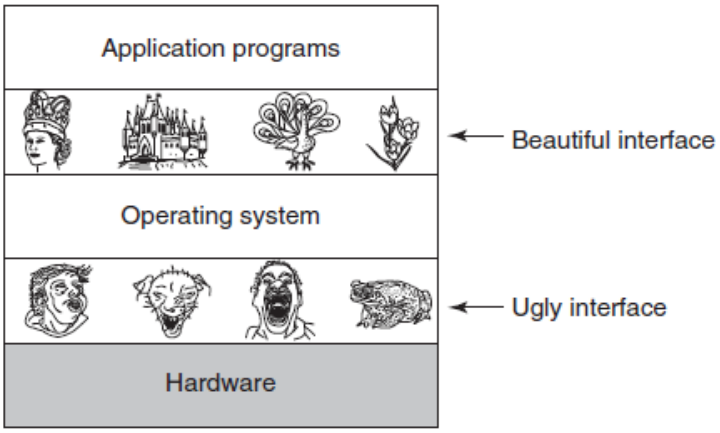


Figure: Operating systems turn ugly hardware into beautiful abstractions

b) The Operating System as a Resource Manager

Modern operating systems allow multiple programs to be in memory and run at the same time. Imagine what would happen if three programs running on some computer all tried to print their output simultaneously on the same printer. The first few lines of printout might be from program 1, the next few from program 2, then some from program 3, and so forth. The result would be a chaos.

But OS doesn't let this occur. OS buffers all the output destined for the printer on the disk. When one program is finished, the operating system can then copy its output from the disk file where it has been stored for the printer, while at the same time the other program can continue generating more output,

oblivious to the fact that the output is not really going to the printer (yet).

A modern computer consists of one or more processors, some main memory, disks, printers, a keyboard, a mouse, a display, network interfaces, and various other input/output devices. Managing all these components and using them optimally is a challenging job. For this reason, computers are equipped with a layer of software called the operating system, whose job is to provide user programs with a better and simpler model of the computer and manage all the resources.

2) Services provided by OS.

Operating System services can be provided into two different points of view.

a) User Point of View:

Where primary goal is to make programming task easier and provide convenient environment to the user.

b) System Point of View:

Where primary goal is efficiency i.e. to perform all the functions efficiently.

a) User Point of View

1) Program Execution

- The main purpose of OS is to provide an efficient and convenient environment for program execution.
- So, OS must provide various functions for loading a program in main memory, execute it, and after execution terminate it.
- The user doesn't have to worry about the memory allocation or multitasking or anything. OS takes care about all these.

2) I/O Operations

- A running program needs I/O operations for reading input data and outputting the results.
- As users cannot control I/O devices directly (for security reasons), OS provides services for I/O operations.

3) Communication

- There are situations where processes need to communicate with each other to exchange information.

- It may be between the processes running on the same computer or running on different computers.

4) Error Detection

- Errors may be there in user programs, CPU, memory or I/O devices.
- OS detects such errors and makes user aware about it.
- It also provides some error recovery mechanism.

5) File System Manipulations

- OS provides file manipulation functionalities like create, write, read and delete a file.

b) System Point of View

1) Resource Allocation

- When multiple users are sharing a same machine or when multiple jobs are running at a same time, there is a need for proper allocation of resources among them. OS does this.

2) Accounting

- It is a process of keeping info about which user uses which resources for what amount of time.
- Such information can be used to bill the users in multiuser environment or to get usage statistics to make future planning.

3) Protection

- OS ensures that all access to system resources is controlled. Also security from outsiders is important.

3) Explain System Call.

- "The interface between **OS** and **USER Program** is defined by the set of calls that OS provide".
- "System call is a programming interface to the services provided by OS."
- System calls are typically written in high level language (C and C++).
- The caller needs to know nothing about how the system call is implemented.
- Most details of OS interface are hidden from the programmer. (by API - Application Programming Interface).
- The system calls available in the OS varies from OS to OS.
- System Calls Example:

  1) read()
  2) write()
  3) close()
  4) open()
  5) fork()

Ex 1) write() System call

- "C program invoking **printf()** library call, which calls **write()** system call."
- The C library handles this call and invokes the necessary system call(s) in the operating system—in this example, the write () system call.
- The C library takes the value returned by write () and passes it back to the user program. This is shown in the following figure.

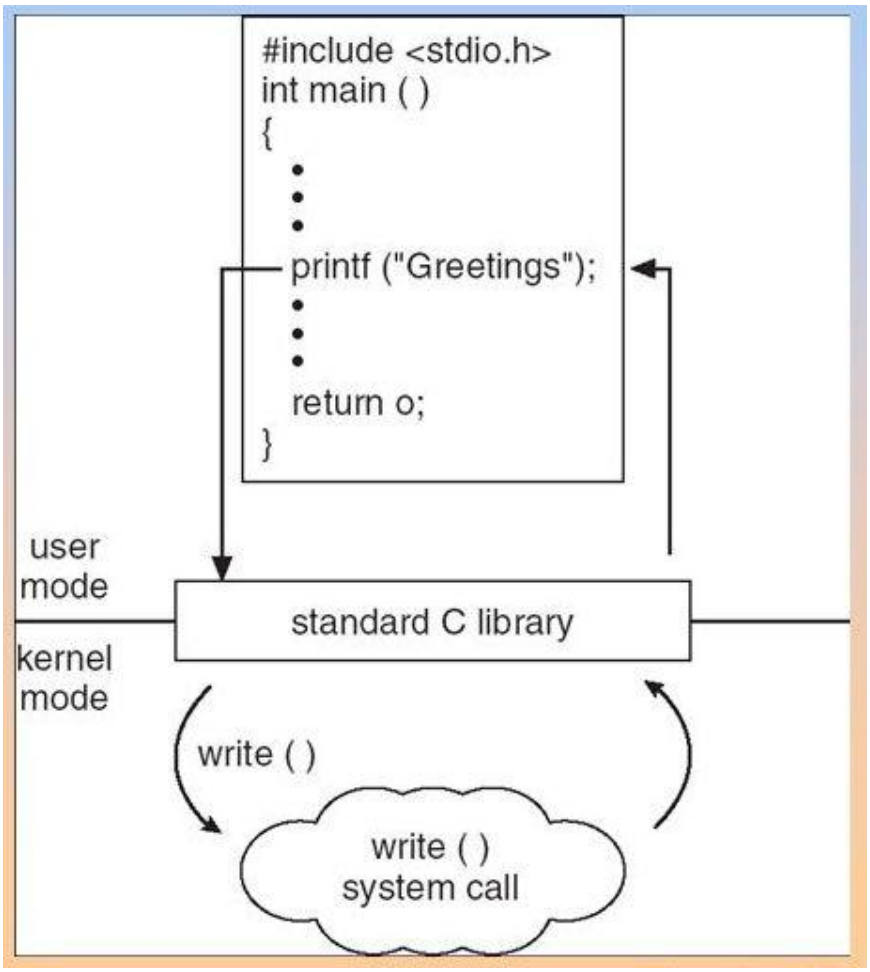


Figure : C library handling of write( )

Ex 2) read() System call

- Let's consider an example of read system call.

Count = read(fd, buffer, nbytes);

fd = specifies file

buffer = points to buffer

nbytes = no. of bytes to read

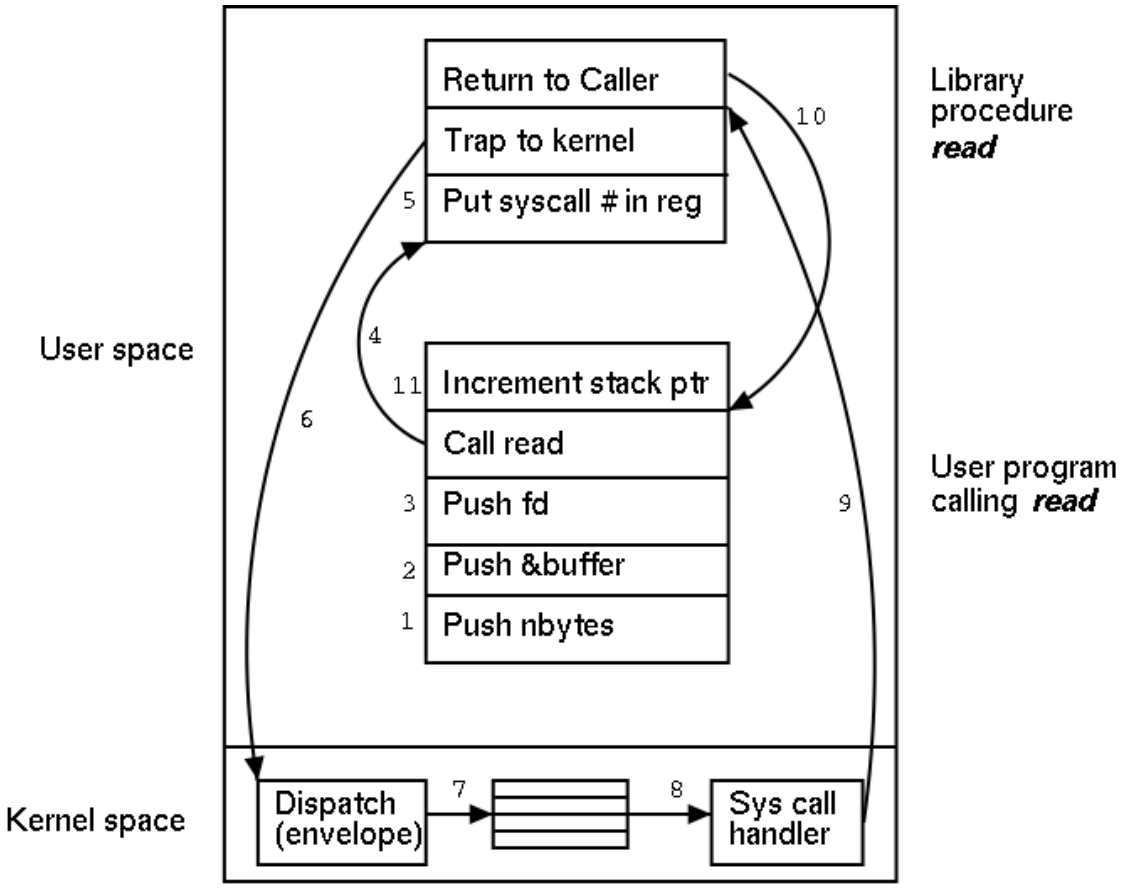
- No. of bytes actually read is placed in Count. This value is generally same as nbytes but it can be sometimes even smaller if end of file is encountered while reading.
- If the system call could not be carried out either due to
  - Invalid parameters or
  - Disk error

Count is set to -1.

Important points to keep in mind:

1) If a process is executing a user program in user mode and needs a system service, such as reading data from a file, it has to execute 'a **TRAP** instruction' to transfer control to OS.
2) The OS then figures out what the calling process wants by checking the parameters.
3) It then carries out the **system call** and returns the control to the instruction following system call.
4) Only system calls can enter in kernel mode procedure calls do not.
5) Like nearly all system calls, it is invoked from C programs, by calling a library procedure with the same name as the System Call : read.

6) System calls are performed in series of steps.
7) C and C++ compilers push the parameters onto the stack in reverse order.



Steps:

1-3) 1st and 3rd parameters are called by value, 2nd parameter is passed by reference, meaning that the address of buffer is passed, not the contents of the buffer.

4) Then actual call to the library procedure is made, which is a normal procedure call.

5) The library procedures (possibly written in assembly language), puts the **system call no.** in a Register.

6) Then it executes **TRAP** instruction to switch from user mode to kernel mode.
7) The kernel code examines the **system call no**. and then dispatches to correct system call handler.
8) At that point **system call handler** runs.
9) Once system call handler has completed its work the control may be **returned to the user space library procedure** following TRAP instruction.
10) This procedure then **returns to the user program** in the usual way procedure calls return.
11) Stack pointer is **incremented** exactly enough to remove the parameters pushed before the call to read. (Assuming that stack grows downwards).