Difference between system call API and system calls:

The system calls are a programmatic way to access the services provided by the kernel. They provide a mechanism through which user space program can interact with kernel. This is done to ease the programmer from writing low level code and increases the security. The system calls are used to switch into kernel mode and use the resources that cannot be accessed in user mode.[i]

API or Application Programmer Interface is function definition to provide user some set of functionalities. API are used to provide a higher level of abstraction than the system calls to the developer. These interfaces are provided in a standardized way, one such standardization is POSIX. The API may or may not use the system call to provide the functionality. While the system call is an explicit request to the kernel through an interrupt.

There are various libraries in UNIX and UNIX-like system to provide API to the programmers. There are also API whose sole purpose is to issue the system call these are known as wrap around routine. An API may also use multiple system calls to provide the desired functionality to the user. It is not compulsory for the API to go in the kernel mode and hence some API also work in user mode.[ii]

Using system calls directly in the application is complicated and requires information about assembly code and low-level interface which vary from architecture to architecture. Due to this reason API are used for ease of use and abstraction

In Linux, man pages are divided into sections where section 3 library functions (API) and section 2 are for system calls.[iii]

Different classes of system calls:

The system call is divided into 6 broad categories according to the service provided by them

1. Process control

   This category contains system calls to provide the service related to process like creation, termination, loading process into main memory and executing it, allocation and deallocating memory, waiting for event etc. Ex: fork, exec

2. File manipulation

   This category contains system calls to provide the service related to file management like creation of file, read, write, getting file attributes, setting file attributes etc. Ex open, read

3. Device manipulation

   This category contains system calls to provide the service related to device management like reading from device buffer, writing to device buffer, requesting a device, releasing a device, logically attaching and detaching device, getting and setting device attributes etc. Ex ioctl,read, write.

4. Information maintenance

   This category contains system calls to provide the service related to transfer of information between kernel and the user space program like getting and setting process attributes, getting process attributes. Ex getpid, sleep.

5. Communication

This category contains system calls to provide the service related to inter-process communication like creation and deletion of communication connections, sending and receive messages. Eg pipe, mmap

6. Protection

   This category contains system calls to provide the service related to security and permissions like changing file permission, changing group and user owner, setting umask. Eg chown, chmod, umask

[iv]

Implementation of system call:

Legacy method:

An interrupt is called in the program using the INT 0x80 which in turn calls a system call interrupt handler code in ring 0 mode or in other words kernel mode. The parameters are passed on the registers and if the no of parameter is greater than the no of registers, they are loaded as struct on the stack. The struct address is then passed in the register.

Every system call is provided with a unique no to identify it. The EAX register is set with unique system call no which we want to call so that interrupt handler can get the information about it. The system call handler has an array of function pointer pointing to the actual code of the system call in the memory. These system call can be directly accessed with this table and system call no.

The result (exit status) of the called system call is stored in the EAX register. In UNIX-like systems positive and 0 values of the exit status corresponds to successful execution of the system call and negative value means some error in the execution. The negative exit status is then passed to the calling application program in the errno variable.

After the termination of the system call be it either successful or erroneous the control is again sent to the calling function and switches from kernel mode to user mode (ring no 3).[v]

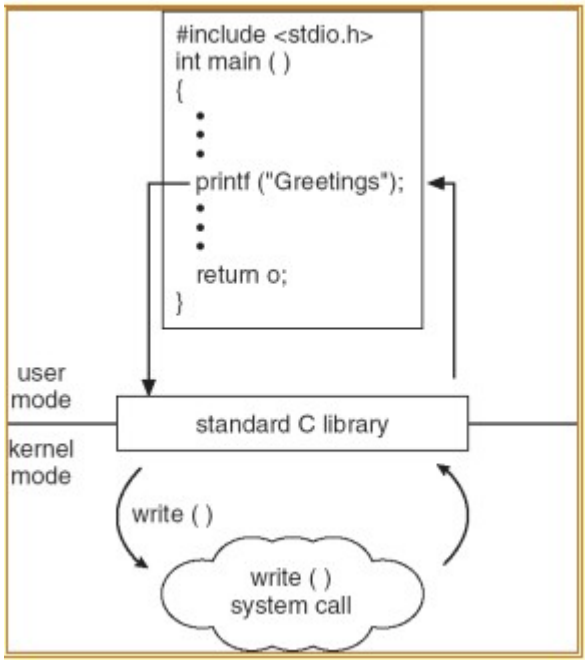
Modern Implementation

After Linux kernel 2.5 version new mechanism was created for system call entry and exit. In the x86 architecture after the release of Pentium II+ it was noted that that performance of software interrupts is not up to the mark because of which an alternative system call entry and exit was implemented using SYSENTER/SYSEXIT instructions.

Apart from the entry/exit mechanism difference this implementation is quite like the legacy implementation. By which we mean running the instructions currently in the memory staring from the address given in the corresponding system call table entry[vi]

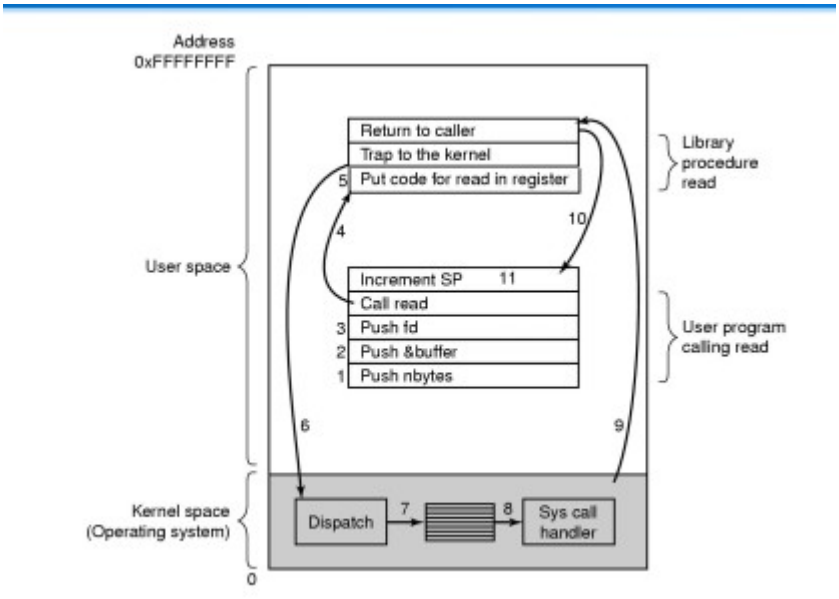
Images:

System call API working

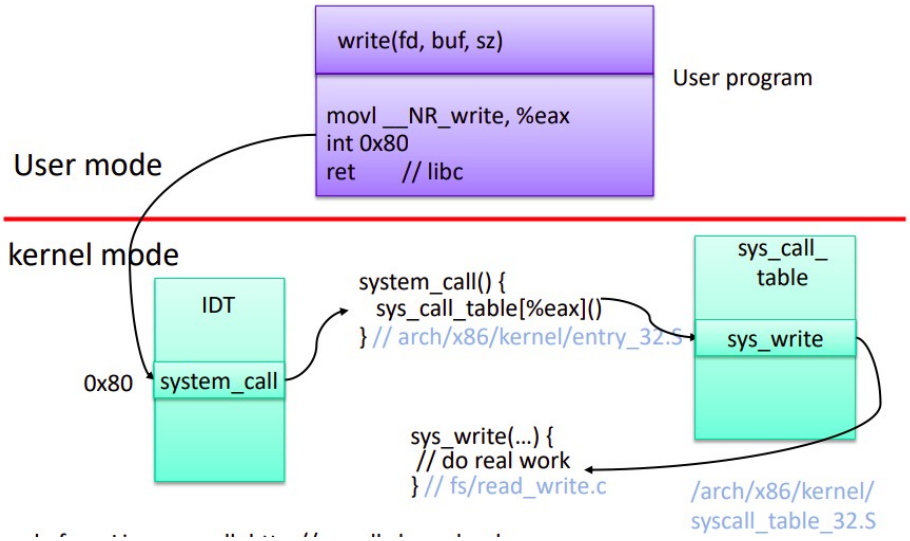


vii

System call implementation



viii


write(fd, buf, sz)
User program
movl __NR_write, %eax
int 0x80
ret // libc
User mode
kernel mode
IDT
0x80
system_call
system_call() {
sys_call_table[%eax]()
} // arch/x86/kernel/entry_32.S
sys_call_
table
sys_write
sys_write(...) {
// do real work
} // fs/read_write.c
/arch/x86/kernel/
syscall_table_32.S

[i] https://cw.fel.cvut.cz/old/_media/courses/ae3b33osd/osd-lec2-14.pdf
[ii] http://www.cs.columbia.edu/~jae/4118/L10-syscall.pdf
[iii] http://www2.latech.edu/~box/csc222/lecture05.pd
[iv] http://www.cs.iit.edu/~cs561/cs450/system_calls/style/8.html
[v] Understanding the linux kernel Chapter 10
[vi] https://articles.manugarg.com/systemcallinlinux2_6.html
[vii] https://cw.fel.cvut.cz/old/_media/courses/ae3b33osd/osd-lec2-14.pdf
[viii] Understanding the linux kernel
[ix] http://www.cs.columbia.edu/~jae/4118/L10-syscall.pdf