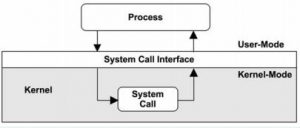
**System Call:**

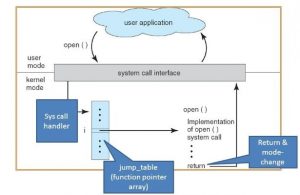
**System Calls are used to show all services offered by the OS. It serves as an interface b/w user program and**[**Operating System**](http://sciencerack.com/different-types-of-operating-system/)**. The system call interface layer includes entrance point in kernel code.**

**The interface between a process and an operating system is provided by system calls. In general, system calls are available as assembly language instructions. They are also included in the manuals used by the assembly level programmers.**

**System calls are usually made when a process in user mode requires access to a resource. Then it requests the kernel to provide the resource via a system call.**

****

**System calls provide an interface between a user program and**[**Operating System**](http://sciencerack.com/different-types-of-operating-system/)**. The system calls expose the services offered by the operating system to user programs. These are set of functions methods. Most system calls are written in assembly language and are machine dependent. Numerous higher level languages such as ‘C’ also permit to make system calls directly.**

****

**The system call interface layer includes entrance point in kernel code. All system resources are managed and controlled by the kernel. Any request from user or application that occupies access to any system resource must be tackled by kernel code. The user process must not be given untie access to kernel code for security cause. Many opening into kernel code called system calls are provided to the user so that the user processes can raise the execution of kernel code.**

**System calls permit processes and users to**[**Operating System**](http://sciencerack.com/different-types-of-operating-system/)**resources.**

**There are three universal methods that are used to pass information between a running program and the**[**Operating System**](http://sciencerack.com/different-types-of-operating-system/)**.**

* **One technique is to store parameters in registers.**
* **Another is to store parameters in a table in memory and get ahead of the address of table.**
* **The third way is to push parameters on the stack and let operating system to pop the parameters off the stack.**

**In computing, a system call is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. A system call is a way for programs to interact with the operating system. A computer program makes a system call when it makes a request to the operating system’s kernel. System call provides the services of the operating system to the user programs via Application Program Interface(API). It provides an interface between a process and operating system to allow user-level processes to request services of the operating system. System calls are the only entry points into the kernel system. All programs needing resources must use system calls.**

**Services Provided by System Calls :**

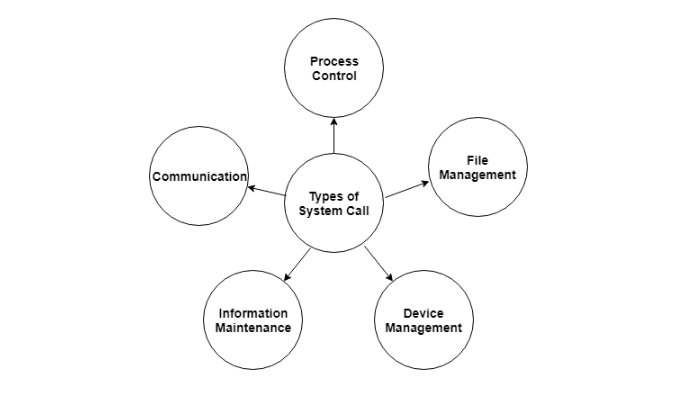
1. **Process creation and management**
2. **Main memory management**
3. **File Access, Directory and File system management**
4. **Device handling(I/O)**
5. **Protection**
6. **Networking, etc.**

**Types of System Calls : There are 5 different categories of system calls –**

* 1. **Process control: end, abort, create, terminate, allocate and free memory.**
  2. **File management: create, open, close, delete, read file etc.**
  3. **Device management**
  4. **Information maintenance**
  5. **Communication**

**Types of System Calls**

**There are mainly five types of system calls. These are explained in detail as follows:**

****

**Here are the types of system calls:**

**Process Control**

**These system calls deal with processes such as process creation, process termination etc.**

**These kinds of system calls are used to direct the processes. Some examples are end, abort, load, execute, create, process, terminate process etc.**

**File Management**

**These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.**

**Device Management**

**These system calls are responsible for device manipulation such as reading from device buffers, writing into device buffers etc. Some examples are request device, release device, read, write, get device attributes etc.**

**Information Maintenance**

**These system calls handle information and its transfer between the operating system and the user program. These types of system calls are used to maintain information. Some examples are get time or date, set time or date, get system data, set system data etc.**

**Communication**

**These system calls are useful for interprocess communication. They also deal with creating and deleting a communication connection. Some examples are create, delete communication connection, send, receive messages etc.**

**Some of the examples of all the above types of system calls in Windows and Unix are given as follows:**

|  |  |  |
| --- | --- | --- |
| **Types of System Calls** | **Windows** | **Linux** |
| **Process Control** | **CreateProcess()**  **ExitProcess()**  **WaitForSingleObject()** | **fork()**  **exit()**  **wait()** |
| **File Management** | **CreateFile()**  **ReadFile()**  **WriteFile()**  **CloseHandle()** | **open()**  **read()**  **write()**  **close()** |
| **Device Management** | **SetConsoleMode()**  **ReadConsole()**  **WriteConsole()** | **ioctl()**  **read()**  **write()** |
| **Information Maintenance** | **GetCurrentProcessID()**  **SetTimer()**  **Sleep()** | **getpid()**  **alarm()**  **sleep()** |
| **Communication** | **CreatePipe()**  **CreateFileMapping()**  **MapViewOfFile()** | **pipe()**  **shmget()**  **mmap()** |

**There are many different system calls as shown above. Details of some of those system calls are as follows:**

**wait()**

**In some systems, a process may wait for another process to complete its execution. This happens when a parent process creates a child process and the execution of the parent process is suspended until the child process executes. The suspending of the parent process occurs with a wait() system call. When the child process completes execution, the control is returned back to the parent process.**

**exec()**

**This system call runs an executable file in the context of an already running process. It replaces the previous executable file. This is known as an overlay. The original process identifier remains since a new process is not created but data, heap, stack etc. of the process are replaced by the new process.**

**fork()**

**Processes use the fork() system call to create processes that are a copy of themselves. This is one of the major methods of process creation in operating systems. When a parent process creates a child process and the execution of the parent process is suspended until the child process executes. When the child process completes execution, the control is returned back to the parent process.**

**exit()**

**The exit() system call is used by a program to terminate its execution. In a multithreaded environment, this means that the thread execution is complete. The operating system reclaims resources that were used by the process after the exit() system call.**

**kill()**

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

**What is a Thread?  
A thread is a path of execution within a process. A process can contain multiple threads.**

**Why Multithreading?  
A thread is also known as lightweight process. The idea is to achieve parallelism by dividing a process into multiple threads. For example, in a browser, multiple tabs can be different threads. MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc. More advantages of multithreading are discussed below**

**Process vs Thread?  
The primary difference is that threads within the same process run in a shared memory space, while processes run in separate memory spaces.  
Threads are not independent of one another like processes are, and as a result threads share with other threads their code section, data section, and OS resources (like open files and signals). But, like process, a thread has its own program counter (PC), register set, and stack space.**

***Advantages of Thread over Process***

***1. Responsiveness:*If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.**

***2. Faster context switch:*Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.**

***3. Effective utilization of multiprocessor system:*If we have multiple threads in a single process, then we can schedule multiple threads on multiple processor. This will make process execution faster.**

***4. Resource sharing:*Resources like code, data, and files can be shared among all threads within a process.  
Note: stack and registers can’t be shared among the threads. Each thread has its own stack and registers.**

***5. Communication:*Communication between multiple threads is easier, as the threads shares common address space. while in process we have to follow some specific communication technique for communication between two process.**

***Enhanced throughput of the system:*If a process is divided into multiple threads, and each thread function is considered as one job, then the number of jobs completed per unit of time is increased, thus increasing the throughput of the system.**

**Types of Threads  
There are two types of threads.  
User Level Thread  
Kernel Level Thread**

# Operating System | User Level thread Vs Kernel Level thread

|  |  |
| --- | --- |
| **USER LEVEL THREAD** | **KERNEL LEVEL THREAD** |
| **User thread are implemented by users.** | **kernel threads are implemented by OS.** |
| **OS doesn’t recognized user level threads.** | **Kernel threads are recognized by OS.** |
| **Implementation of User threads is easy.** | **Implementation of Kernel thread is complicated.** |
| **Context switch time is less.** | **Context switch time is more.** |
| **Context switch requires no hardware support.** | **Hardware support is needed.** |
| **If one user level thread perform blocking operation then entire process will be blocked.** | **If one kernel thread perform blocking operation then another thread can continue execution.** |
| **Example : Java thread, POSIX threads.** | **Example : Window Solaris.** |