Introduction of System Call

Last Updated : 01 Jul, 2024

A system call is a 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 requests the operating system's kernel.

System call **provides** the services of the operating system to the user programs via the Application Program Interface(API). It provides an interface between a process and an 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.

What is a System Call?

A system call is a mechanism used by programs to request services from the operating system (OS). In simpler terms, it is a way for a program to interact with the underlying system, such as accessing hardware resources or performing privileged operations.

A user program can interact with the operating system using a system call. A number of services are requested by the program, and the OS responds by launching a number of systems calls to fulfill the request. A system call can be written in high-level languages like C or Pascal or in assembly language. If a high-level language is used, the operating system may directly invoke system calls, which are predefined functions.

A system call is initiated by the program executing a specific instruction, which triggers a switch to kernel mode, allowing the program to request a service from the OS. The OS then handles the request, performs the necessary operations, and returns the result back to the program.

System calls are essential for the proper functioning of an operating system, as they provide a standardized way for programs to access system resources. Without system calls, each program would need to implement its methods for accessing hardware and system services, leading to inconsistent and error-prone behavior.

Services Provided by System Calls

- Process Creation and Management
- Main Memory Management
- File Access, Directory, and File System Management
- Device Handling(I/O)
- Protection
- Networking, etc.
    - **Process Control:** end, abort, create, terminate, allocate, and free memory.
    - **File Management:** create, open, close, delete, read files,s, etc.
    - **Device Management**
    - **Information Maintenance**
    - **Communication**

Features of System Calls

- **Interface:** System calls provide a well-defined interface between user programs and the operating system. Programs make requests by calling specific functions, and the operating system responds by executing the requested service and returning a result.
- **Protection:** System calls are used to access privileged operations that are not available to normal user programs. The operating system uses this privilege to protect the system from malicious or unauthorized access.
- **Kernel Mode:** When a system call is made, the program is temporarily switched from user mode to kernel mode. In kernel mode, the program has access to all system resources, including hardware, memory, and other processes.
- **Context Switching:** A system call requires a context switch, which involves saving the state of the current process and switching to the kernel mode to execute the requested service. This can introduce overhead, which can impact system performance.
- **Error Handling:** System calls can return error codes to indicate problems with the requested service. Programs must check for these errors and handle them appropriately.
- **Synchronization:** System calls can be used to synchronize access to shared resources, such as files or network connections. The operating system provides synchronization mechanisms, such as locks or semaphores, to ensure that multiple programs can access these resources safely.

How does System Call Work?

Here is a detailed explanation step by step how system calls work:

- **Users need special resources:** Sometimes programs need to do some special things that can’t be done without the permission of the OS like reading from a file, writing to a file, getting any information from the hardware, or requesting a space in memory.

- **The program makes a system call request:** There are special predefined instructions to make a request to the operating system. These instructions are nothing but just a “system call”. The program uses these system calls in its code when needed.
- **Operating system sees the system call:** When the OS sees the system call then it recognizes that the program needs help at this time so it temporarily stops the program execution and gives all the control to a special part of itself called ‘Kernel’. Now ‘Kernel’ solves the need of the program.
- **The operating system performs the operations:** Now the operating system performs the operation that is requested by the program. Example: reading content from a file etc.
- **Operating system give control back to the program :** After performing the special operation, OS give control back to the program for further execution of program .

Examples of a System Call in Windows and Unix

System calls for Windows and Unix come in many different forms. These are listed in the table below as follows:

Process	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	Fork() Exit() Wait()
File manipulation	CreateFile() ReadFile() WriteFile()	Open() Read() Write() Close()

Process	Windows	Unix
Device Management	SetConsoleMode() ReadConsole() WriteConsole()	Ioctl() Read() Write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	Getpid() Alarm() Sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	Pipe() Shmget() Mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorgroup()	Chmod() Umask() Chown()

Open(): Accessing a file on a file system is possible with the open() system call. It gives the file resources it needs and a handle the process can use. A file can be opened by multiple processes simultaneously or just one process. Everything is based on the structure and file system.

Read(): Data from a file on the file system is retrieved using it. In general, it accepts three arguments:

- A description of a file.
- A buffer for read data storage.
- How many bytes should be read from the file
  Before reading, the file to be read could be identified by its file descriptor and opened using the open() function.

Wait(): In some systems, a process might need to hold off until another process has finished running before continuing. When a parent process creates a child process, the execution of the parent process is halted until the child process is complete. The parent process is stopped using the wait() system call. The parent process regains control once the child process has finished running.

Write(): Data from a user buffer is written using it to a device like a file. A program can produce data in one way by using this system call. generally, there are three arguments:

- A description of a file.
- A reference to the buffer where data is stored.
- The amount of data that will be written from the buffer in bytes.

Fork(): The fork() system call is used by processes to create copies of themselves. It is one of the methods used the most frequently in operating systems to create processes. When a parent process creates a child process, the parent process's execution is suspended until the child process is finished. The parent process regains control once the child process has finished running.

Exit(): A system call called exit() is used to terminate a program. In environments with multiple threads, this call indicates that the thread execution is finished. After using the exit() system function, the operating system recovers the resources used by the process.

Methods to Pass Parameters to OS

If a system call occur, we have to pass parameter to the Kernal part of the Operating system.

For example look at the given **open()** system call:

C

```
//function call example
```

```
#include <fcntl.h>

int open(const char *pathname, int flags, mode_t mode);
```

Here **pathname**, **flags** and **mode_t** are the parameters.

So it is to be noted that :

- We can't pass the parameters directly like in an ordinary function call.
- In Kernal mode there is a different way to perform a function call.

So we can't run it in the normal address space that the process had already created and hence we cant place the parameters in the top of the stack because it is not available to the Kernal of the operating system for processing. so we have to adopt any other methods to pass the parameters to the Kernal of the OS.

We can done it through,

- **Passing parameters in registers**
- **Address of the block is passed as a parameter in a register.**
- **Parameters are pushed into a stack.**

Let us discuss about each points in detail:

1. Passing Parameters in Registers

- It is the simplest method among the three
- Here we directly pass the parameters to registers.
- But it will it is limited when, number of parameters are greater than the number of registers.
- Here is the C program code:

C

```
// Passing parameters in registers.

#include <fcntl.h>
```

```
#include <stdio.h>

int main()
{
    const char* pathname = "example.txt";
    int flags = O_RDONLY;
    mode_t mode = 0644;

    int fd = open(pathname, flags, mode);
  // in function call open(), we passed the parameters
pathanme,flags,mode to the kernal directly

    if (fd == -1) {
        perror("Error opening file");
        return 1;
    }

    // File operations here...

    close(fd);
    return 0;
}
```

2. Address of The Block is Passed as Parameters

- It can be applied when the number of parameters are greater than the number of registers.
- Parameters are stored in blocks or table.
- The address of the block is passed to a register as a parameter.
- Most commonly used in Linux and Solaris.
- Here is the C program code:

C

```
//Address of the block is passed as parameters

#include <stdio.h>
```

```
#include <fcntl.h>

int main() {
    const char *pathname = "example.txt";
    int flags = O_RDONLY;
    mode_t mode = 0644;

    int params[3];
          // Block of data(parameters) in array
    params[0] = (int)pathname;
    params[1] = flags;
    params[2] = mode;

    int fd = syscall(SYS_open, params);
          // system call

    if (fd == -1) {
        perror("Error opening file");
        return 1;
    }

    // File operations here...

    close(fd);
    return 0;
}
```

3. Parameters Are Pushed in a Stack

- In this method parameters can be pushed in using the program and popped out using the operating system
- So the Kernal can easily access the data by retrieving information from the top of the stack.
- Here is the C program code

C

```
//parameters are pushed into the stack

#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

int main() {
    const char *pathname = "example.txt";
    int flags = O_RDONLY;
    mode_t mode = 0644;

    int fd;
    asm volatile(
        "mov %1, %%rdi\n"
        "mov %2, %%rsi\n"
        "mov %3, %%rdx\n"
        "mov $2, %%rax\n"
        "syscall"
        : "=a" (fd)
        : "r" (pathname), "r" (flags), "r" (mode)
        : "%rdi", "%rsi", "%rdx"
    );

    if (fd == -1) {
        perror("Error opening file");
        return 1;
    }

    // File operations here...

    close(fd);
    return 0;
}
```

Advantages of System Calls

- **Access to Hardware Resources:** System calls allow programs to access hardware resources such as disk drives, printers, and network devices.
- **Memory Management:** System calls provide a way for programs to allocate and deallocate memory, as well as access memory-mapped hardware devices.
- **Process Management:** System calls allow programs to create and terminate processes, as well as manage inter-process communication.
- **Security:** System calls provide a way for programs to access privileged resources, such as the ability to modify system settings or perform operations that require administrative permissions.
- **Standardization:** System calls provide a standardized interface for programs to interact with the operating system, ensuring consistency and compatibility across different hardware platforms and operating system versions.

Disadvantages of System Call

- **Performance Overhead**: System calls involve switching between user mode and kernel mode, which can slow down program execution.
- **Security Risks**: Improper use or vulnerabilities in system calls can lead to security breaches or unauthorized access to system resources.
- **Error Handling Complexity**: Handling errors in system calls, such as resource allocation failures or timeouts, can be complex and require careful programming.
- **Compatibility Challenges**: System calls may vary between different operating systems, requiring developers to write code that works across multiple platforms.
- **Resource Consumption**: System calls can consume significant system resources, especially in environments with many concurrent processes making frequent calls.

Conclusion

In conclusion, system calls are a important part of how computer programs interact with the operating system. They provide a way for applications to request services from the OS, such as accessing files, managing memory, or communicating over networks. System calls act as a bridge between user-level programs and the low-level operations handled by the operating system kernel. Understanding system calls is essential for developers to create efficient and functional software that can leverage the full capabilities of the underlying operating system.

Frequently Asked Questions on System Call in Operating Systems – FAQs

How does a system call work?

When a program executes a system call, it transitions from user mode to kernel mode, which is a higher privileged mode. The transition is typically initiated by invoking a specific function or interrupting instruction provided by the programming language or the operating system.

Once in kernel mode, the system call is handled by the operating system. The kernel performs the requested operation on behalf of the program and returns the result. Afterward, control is returned to the user-level program, which continues its execution.

Why do programs need to use system calls?

Programs use system calls to interact with the underlying operating system and perform essential tasks that require privileged access or system-level resources.

What are some examples of system calls?

Examples include opening and closing files, reading and writing data, creating processes, allocating memory, and performing network operations like sending and receiving data.

Why are system calls necessary?

System calls are necessary as they enable applications to request essential services from the operating system, such as file access, memory management, and hardware control, ensuring secure and efficient interaction with computer resources.

S Sami…

Previous Article

Operating System Services

Next Article

System Programs in Operating System

Similar Reads

Difference between system call and library call

1. System Call :There are two modes in the computer system one is user mode and another is kernel mode. In computer system there are different types of processes that are running on a computer system. When a user runs an application it is said to be in user mode or computer is in user mode. When there is a requirement of hardware resource, the proc

3 min read

Xv6 Operating System -adding a new system call

Prerequisite - Xv6 Operating System -add a user program In last post we got to know how to add user program in Xv6 Operating System. Now here you will see how to add new system call in Xv6 Operating

System. A dding new system call to xv6: A system call is way for programs to interact with operating system. A computer program makes system call when

6 min read

Fork System Call in Operating System

In many operating systems, the fork system call is an essential operation. The fork system call allows the creation of a new process. When a process calls the fork(), it duplicates itself, resulting in two processes running at the same time. The new process that is created is called a child process. It is a copy of the parent process. The fork syst

5 min read

Implementation of sleep (system call) in OS

In this article, we are going to learn about sleep (system call) in operating systems. In the computer science field, a system call is a mechanism that provides the interface between a process and the operating system. In simple terms, it is basically a method in which a computer program requests a service from the kernel of the operating system. W

4 min read

Linux system call in Detail

A system call is a procedure that provides the interface between a process and the operating system. It is the way by which a computer program requests a service from the kernel of the operating system. Different operating systems execute different system calls. In Linux, making a system call involves transferring control from unprivileged user mod

4 min read

View More Articles

Article Tags : Operating Systems