# Microkernel in Operating Systems

A microkernel is a type of operating system kernel that is designed to provide only the most basic services required for an operating system to function, such as memory management and process scheduling.

One major disadvantage is that message passing between user-level processes can be slower than direct system calls in a monolithic kernel.

**Kernel** is the core part of an operating system that manages system resources. It also acts as a bridge between the application and hardware of the computer. It is one of the first programs loaded on start-up (after the Bootloader).

**Kernel mode and User mode of CPU operation**

The CPU can execute certain instructions only when it is in kernel mode. These instructions are called privilege instruction.

- The operating system puts the CPU in kernel mode when it is executing in the kernel so, that kernel can execute some special operation.
- The operating system puts the CPU in user mode when a user program is in execution so, that the user program cannot interface with the operating system program.

**What is Microkernel?**

- A microkernel is one of the classifications of the kernel. Being a kernel it manages all system resources. But in a microkernel, the **user services** and **kernel services** are implemented in different address spaces. The user services are kept in **user address space**, and kernel services are kept under **kernel address space**, thus also reduces the size of kernel and size of an operating system as well.

And the microkernel is solely responsible for the most important services of the operating system they are named as follows:

- Inter process-Communication
- Memory Management
- CPU-Scheduling

A buffer is a memory area that stores data being transferred between two devices or between a device and an application.

**Caching –**
A *cache* is a region of fast memory that holds a copy of data. Access to the cached copy is much easier than the original file.

The main difference between a buffer and a cache is that a buffer may hold only the existing copy of a data item, while a cache, by definition, holds a copy on faster storage of an item that resides elsewhere.

A *spool* is a buffer that holds the output of a device,

Kernel I/O Subsystem in Operating System

Kernel I/O Subsystem, which provides an interface between the operating system and input/output (I/O) devices. The Kernel I/O Subsystem manages the I/O requests made by the user applications and translates them into hardware commands that the devices can understand.

**Monolithic Kernel** is another classification of Kernel. Like microkernel, this one also manages system resources between application and hardware, but **user services** and **kernel services** are implemented under the same address space.

A monolithic kernel is a type of operating system kernel in which all the operating system services run in kernel space, meaning they all share the same memory space. This type of kernel is characterized by its tight integration of system services and its high performance.

On the other hand, a microkernel is a type of operating system kernel in which only the most basic services run in kernel space, with other services running in user space. This type of kernel is characterized by its modularity, simplicity, and ability to run multiple operating systems on the same hardware.

| Basics | Micro Kernel | Monolithic Kernel |
|---|---|---|
| Size | Smaller in | Larger as OS and user both lie in the same address space. |
| Execution | Slower | Faster |
| Extendible | Easily extendible | Complex to extend |
| Security | If the service crashes then there is no effect on | If the process/service crashes, the whole system crashes as both user |

| Basics | Micro Kernel | Monolithic Kernel |
|---|---|---|
| | working on the microkernel. | and OS were in the same address space. |
| Code | More code is required to write a microkernel. | Less code is required to write a monolithic kernel. |
| Examples | L4Linux, macOS | Windows, Linux BSD |
| Security | More secure because only essential services run in kernel mode | Susceptible to security vulnerabilities due to the amount of code running in kernel mode |
| Platform independence | More portable because most drivers and services run in user space | Less portable due to direct hardware access |
| Communication | Message passing between user-space servers | Direct function calls within kernel |
| Performance | Lower due to message passing and more overhead | High due to direct function calls and less overhead |

# Introduction of System Call

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**.

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 | Windows | Unix |
|---|---|---|
| Process Control | CreateProcess()<br>ExitProcess()<br>WaitForSingleObject() | Fork()<br>Exit()<br>Wait() |
| File manipulation | CreateFile()<br>ReadFile()<br>WriteFile() | Open()<br>Read()<br>Write()<br>Close() |
| Device Management | SetConsoleMode()<br>ReadConsole()<br>WriteConsole() | Ioctl()<br>Read()<br>Write() |
| Information Maintenance | GetCurrentProcessID()<br>SetTimer()<br>Sleep() | Getpid()<br>Alarm()<br>Sleep() |
| Communication | CreatePipe()<br>CreateFileMapping()<br>MapViewOfFile() | Pipe()<br>Shmget()<br>Mmap() |
| Protection | SetFileSecurity()<br>InitializeSecurityDescriptor()<br>SetSecurityDescriptorgroup() | Chmod()<br>Umask()<br>Chown() |

**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.

**We can done it through,**

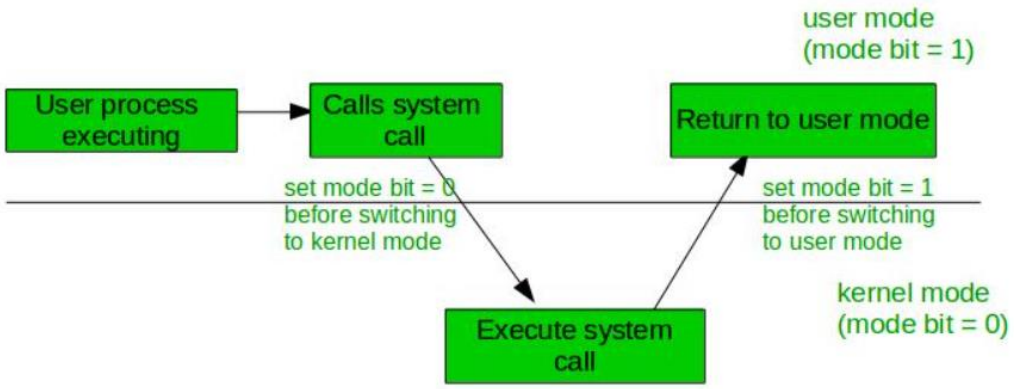
1. **Passing parameters in registers**
2. **Address of the block is passed as a parameter in a register.**
3. **Parameters are pushed into a stack.**

The **getrlimit()** and **setrlimit()** system calls can be used to get and set the resource limits such as files, CPU, memory etc. associated with a process. *Each resource has an associated **soft and hard limit.***

- ***soft limit:*** *The soft limit is the actual limit enforced by the kernel for the corresponding resource.*
- ***hard limit:*** *The hard limit acts as a ceiling for the soft limit.*

***The soft limit ranges in between 0 and hard limit.***

**User mode –**
When the computer system is run by user applications like creating a text document or using any application program, then the system is in user mode. When the user application requests for a service from the operating system or an interrupt occurs or system call, then there will be a transition from user to kernel mode to fulfill the requests.
***Note:*** *To switch from kernel mode to user mode, the mode bit should be 1.*




**Kernel Mode:** When the system boots, the hardware starts in kernel mode and when the operating system is loaded, it starts user application in user mode. To provide protection to the hardware, we have privileged instructions which execute only in kernel mode. If the user attempts to run privileged instruction in user mode then it will treat instruction as illegal and traps to OS. Some of the privileged instructions are:

1. Handling Interrupts
2. To switch from user mode to kernel mode.
3. Input-Output management.