Many decades ago, when computers were the size of a room, users were running their programs with huge struggles, and sometimes their programs even crashed the computer. To protect against programs that continuously crashed computers, we developed newer operating systems with two distinct operative modes.

Kernel mode, also known as system mode, is one of the central processing unit (CPU) operating modes. **While processes run in kernel mode, they have unrestricted access to the hardware.**

The other mode is user mode, which is a non-privileged mode for user programs. **Therefore, when a process runs in user mode, it has limited access to the CPU and the memory.**

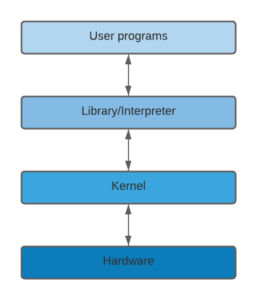
**3. Necessity for User Mode and Kernel Mode**

[OS kernel](https://www.baeldung.com/cs/os-kernel) is the most important program in the set. When the system boots, including several essential procedures that are needed for the system, it is loaded into the memory. The other programs are actually less important. Even though they can offer a wide range of interactive experiences to the user, the kernel determines the system’s basic shape and capability.

The OS must accomplish two major objectives:

* Interact with the hardware that services all of the low-level programmable elements
* Maintain an environment in which the computer system’s applications can run.  These are also called user programs.

Some operating systems, such as MS-DOS, allow user programs to directly interact with the hardware. On the other hand, Unix-like operating systems protect all low-level information about the physical organization of the machine from programs run by the user. **The below figure shows the general interaction between user space programs and kernel space programs in modern operating systems:**



**A program must request the OS in order to access a hardware resource.** The kernel reviews the request, and if it decides to give permission, interacts with the relevant hardware components on behalf of the user program.

**In order to impose this mechanism, modern operating systems depend on the availability of specialized hardware features.** These features should prevent user programs from directly interacting with low-level hardware components or accessing arbitrary memory spaces.

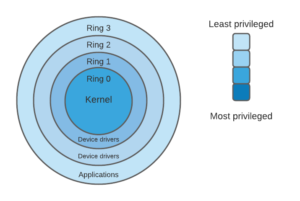
In fact, the hardware implements at least two different CPU execution modes: a non-privileged mode for user programs, and a privileged mode for kernel programs. Unix calls these user mode and kernel mode.

**Operating system protection rings are the most common way to implement a user mode apart from kernel mode.**

**3.1. Protection Rings**

Protection rings are mechanisms to protect data and functionality from faults and malicious behavior. They have two main jobs which are:

* Improving fault tolerance
* Providing computer security



Different operating systems provide different types of resource control. Within the architecture of a computer system, a protection ring is one of two or three hierarchical tiers of layers of privilege. Some CPU architectures, such as *x86*which has separate CPU modes at the hardware or microcode level, commonly implement this mechanism.

**As the above figure shows, protection rings have a hierarchy that arranges the most trusted ring, usually numbered zero, to the least trusted.** On most operating systems, Ring 0 is the level with the most privileges. It represents the kernel mode and interacts with physical hardware, such as the CPU and the memory.

**3.2. Examples of Protection Rings in Different Operating Systems**

Linux, Windows, and macOS are three operating systems that use kernel/user mode. Userspace programs must use a system call into kernel mode in order to perform specialized functions. It may even be the kernel space where the trusted code of the OS will perform the needed task and then return the execution back to the userspace.

Conversely, DOS and other similar simple operating systems and embedded devices run in kernel mode constantly. It means that drivers can be written as user programs. This situation may cause security problems for Internet-of-Things (IoT) devices.

**4. Switching From User to Kernel Mode**

Above we explained why we don’t want userspace programs to easily interact with kernel-mode or switch back to kernel mode. This access may cause the two privilege modes to become useless.

On the other hand, there are times when a user program does need to switch back to kernel mode, like when a program cannot read from disk, or get a line from the keyboard in user mode. OS is the only software that can do such operations.

At that point, the program must ask the OS to access the hardware on its behalf and get some input for the program. So we need a mechanism where a user-mode program can switch into kernel mode.

This mechanism is the system call. It is implemented in the CPU as the trap instruction. In order to add a new system call to the OS, we need to modify and add some files into kernel and userspace. After that, we must compile the kernel and boot it to use this new system call.

**4.1. System Calls**

**System calls are special functions that manage OS routines that occur in kernel mode.**Let’s look at how system calls work:

* To specify which particular service it needs from the OS, the user program places some values in registers or generates a stack frame with arguments.
* Then the user program executes the trap instruction.
* Trap instructions are part of the OS and they have memory protection; therefore, they cannot be modified by user programs. Also, they are not readable by user programs.
* The trap or system call handler instructions read the information of the requested service and then execute the request in kernel mode.
* When the system call is completed, the OS returns to user mode and exits the system call.

The trap instruction executes extraordinary behaviour in a single instruction, with the effects available at the next instruction from the perspective of the user program. **In fact, the CPU enters kernel mode to the system call handler, which performs the work and then returns to the program in user mode.**

**A system call can be used when we:**

* need to access privileged information like process id (pid), or changing scheduler policy
* access an I/O device or a file
* need to change the execution context

System calls are almost the only way for user mode to interact with low-level services. For example, if we want to change the scheduler policy of the OS, we need to do this operation with a system call.

**4.2. Why Do We Trust the OS?**

Any program operating in kernel mode has the complete privilege to the system. We have to trust it because we have no other choice. Consequently, we must trust the system software, which is the OS itself, and more specifically, the part of the OS that runs in kernel mode known as the [kernel](https://www.baeldung.com/cs/os-kernel).