In operating systems, the applications depend on the events in order to execute functions. Hence, the OS comes under the event-driven module paradigm. An event can occur in an OS when it executes some programs or applications. An event triggers the execution of a program and changes the state of an OS from a user-mode to a kernel mode.

In this tutorial, **we’ll discuss two popular types of events in the OS: trap and interrupt.** We’ll explore the fundamentals of these two events and **highlight the core differences between them.**

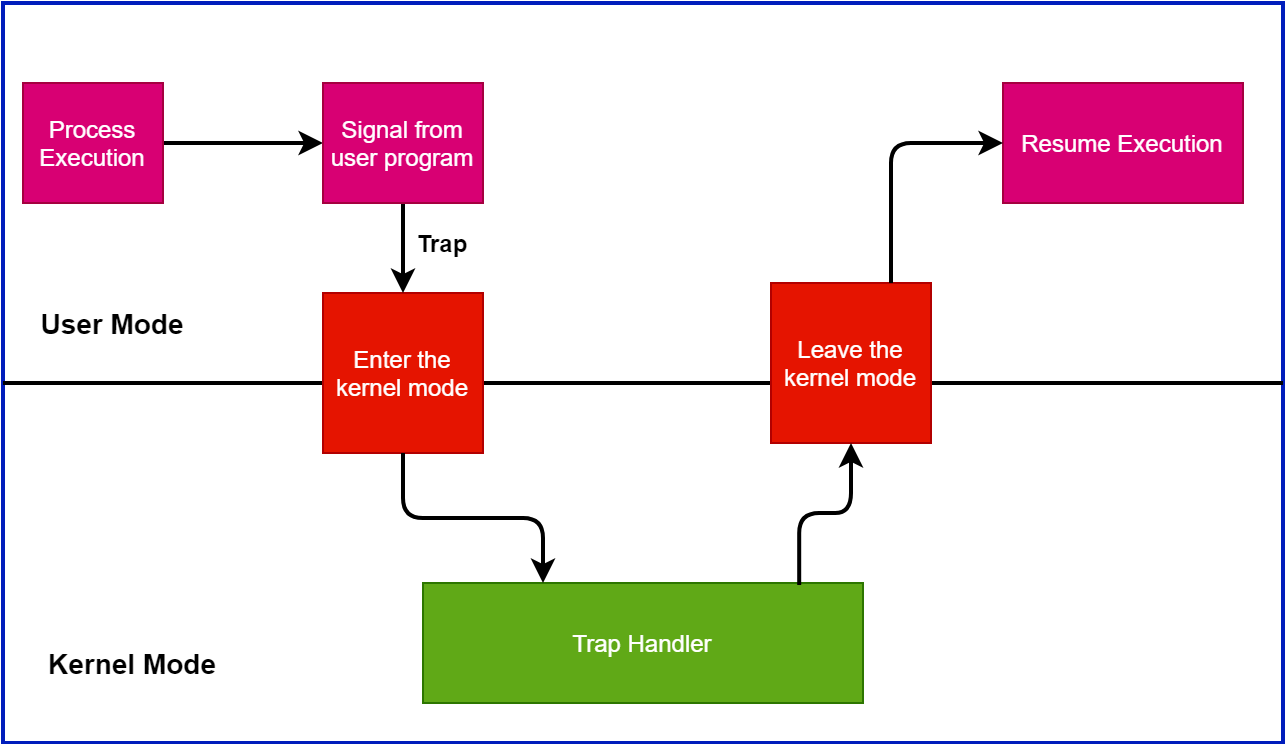
**2. Introduction to Trap in OS**

**A trap is a**[**synchronous**](https://en.wikipedia.org/wiki/Synchronization_(computer_science))**interrupt triggered by an exception in a user process to execute functionality.** Exception conditions like invalid memory access, division by zero, or a breakpoint can trigger a trap in an OS.

A trap changes the mode of an OS to a [kernel routine](https://en.wikipedia.org/wiki/Kernel_(operating_system)). Here, the OS executes some actions and returns the control to the process it was executing before. During a trap, the execution of a process is set as a high priority compared to user code.

Moreover, the trap handler is synchronous. When the OS detects a trap, it pauses the user process. As soon as the[system call](https://en.wikipedia.org/wiki/System_call) is completed, the OS resumes and continues the execution of the user processes:

**AD**



Additionally, a trap can be viewed as a [CPU interrupt handler](https://en.wikipedia.org/wiki/Interrupt_handler). The OS saves the stack pointers, registers memory, and can resume its previous execution again.

Let’s take an example. Suppose a user running a program on an OS. There is a print statement in the program:

printf(“%s\n”, str);

The OS executes this statement and triggers the print functionality. Furthermore, the print function displays the string returned by the code in the monitor. Hence, this is an example of a trap.

The OS executes the print statement with high priority.  As soon as the OS encounters this statement, it switches to kernel mode. After the OS finishes the execution, the mode is switched back to the user mode.

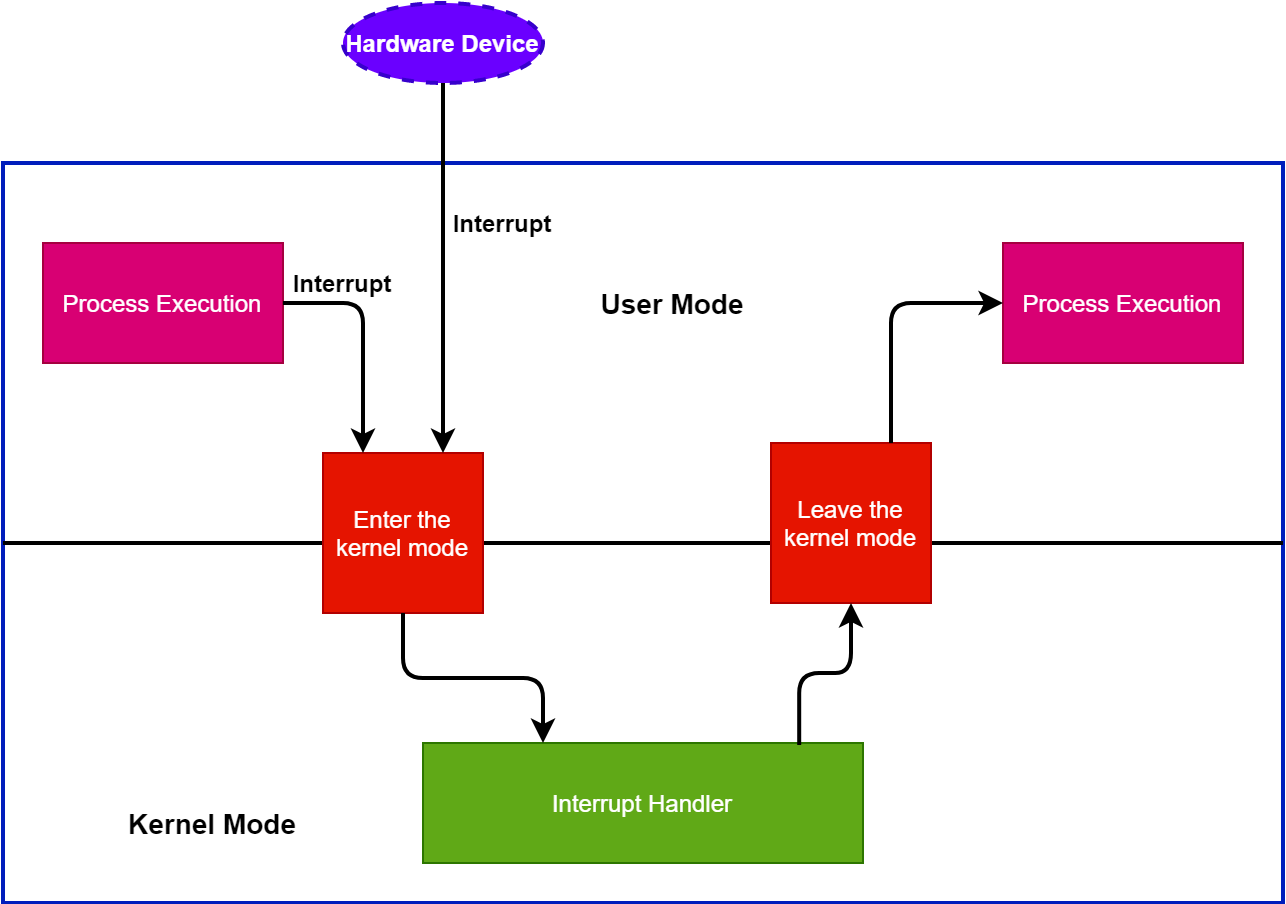
**3. Fundamentals of Interrupt in OS**

**An interrupt is a hardware or software signal that demands instant attention by an OS.** It notifies the processor that a critical process needs urgent execution. In such a case, the present working process is interrupted.

**AD**

All modern computers are interrupt-driven. The OS starts a sequence of instructions in a single application. Furthermore, the execution of the instruction continues until it’s completed or receives an interrupt signal. Additionally, a specific bus control line called the [Interrupt Service Routine (ISR)](https://en.wikipedia.org/wiki/Interrupt_handler) handles interrupts in I/O devices.

A CPU contains a specific interrupt pin known as an INT pin for the interruption. The INT pin connects hardware devices such as keyboards, NIC cards with the CPU. When we press a key on the keyboard, an interrupt is generated. Subsequently, the OS changes the context and invokes the keyboard interrupt handler routine. After completing the execution, it switches back to the previous process:



In general, **multiple hardware devices share a single INT pin using an interrupt controller.** To determine which device produced the interrupt, the processor contacts the interrupt controller. Furthermore, the processor performs the relevant interrupt handler procedure like keyboard or USB interrupt handler routine as per the communication with the interrupt controller.

In general, **an interrupt can occur from a hardware or a software device.** After an I/O operation is done, we can observe a hardware interruption. On the other hand, when an application ends or seeks specific operating system services, a software interrupt occurs.

**4. Difference Between Trap and Interrupt**

Now we know the fundamentals of a trap and interrupt in the OS. **Let’s discuss some core differences between these two concepts:**

