A computer *process* is a program in execution. Its *state* includes the executable code and data in the main memory (the memory image of the process), the value of the program counter (giving the address of the next machine instruction to execute), and the values of all the general-purpose registers.

When an event or process of higher priority needs to get prompt attention from the CPU, it can *interrupt* the execution of the current process holding the CPU, save its key state information in memory or hard disk, do whatever interrupt processing with the CPU, then resume the original process’s execution by restoring its state in memory and registers.

Interrupt processing is at the core of modern operating system implementation. You must make sure that you thoroughly understand it.

### 1. What is an interrupt?

An *interrupt* is a dynamic event that needs prompt attention by the CPU. Usually an interrupt only needs a short period of CPU time to serve it. After that the original process can resume its execution.

There are two types interrupting events: *hardware interrupts* that are those issued by I/O device controllers when they need CPU to process I/O data, and software interrupts, also called *traps*, that are raised when the current process executes a special trap instruction to indicate that something wrong has happened or the process needs special service from the operating system (like performing some I/O operation).

Only a limited number of types of interrupts are defined. For each type of I/O devices, there is a special program called an *interrupt handler* to serve the interrupt requests from these devices. For all software traps, there is also a special trap handler defined.

Each type of interrupt has an associated *priority level*. A running process would only be interrupted by an interrupt source or trap of higher priority. When the CPU is executing an interrupt handler, the interrupt handler may be further interrupted by an interrupt source of even higher priority.
2. How to issue an interrupt request?

The CPU, main memory, and I/O devices are all connected through a system bus. The system bus consists of many parallel metal wires, some carrying data, some carrying addresses for the main memory or I/O device IDs (as well as addresses for I/O device ports; each I/O device has a few registers called ports, and each port has a unique address), and some carrying control signals.

There are two groups of wires of the system bus that are critical to the working of interrupt processing:

- The **Interrupt Request Line**: It has two states. When it is high (carrying high voltage), at least one I/O device or a trap is requesting an interrupt.
- The **Address Lines**: During interrupt processing, they will carry the device ID of the interrupt source with the highest priority.

For an I/O device to make an interrupt request, it only needs to raise the Interrupt Request Line. Several I/O devices can raise the Interrupt Request Line at the same time, but only the ID of the device with the highest priority will be broadcast on the Address lines.

3. How is an interrupt request detected?

There is a special **Interrupt Enabled Bit** in a system control register. If this bit carries 1, the system is ready to process interrupt requests. If this bit carries 0, the system will ignore any interrupt requests. The latter is important when the current process is doing some very sensitive operation that an interrupt may destabilize the system.

For the discussion of interrupt processing, the execution of an instruction has three phases:

a. Read the instruction from the main memory, and save it in the Instruction Register
b. Execute the instruction
c. If the Interrupt Enabled Bit is carrying 1, then check the Interrupt Request Line for any interrupt requests. If the Interrupt Request Line is raised, enter the **Interrupt Processing Phase**

A natural observation from the above description is that the system only checks for potential interrupt requests at the end of execution of each instruction.

4. How to find the interrupt’s handler?

As we mentioned earlier, each type of interrupts will be processed by its own special interrupt handler. These interrupt handlers are an important component of an operating system. The starting address of each of these interrupt handlers are saved in a table called
an *Interrupt Vector*. The Interrupt Vector is usually stored at the lower end of the main memory, starting from address 0. Suppose an I/O device has ID 3, then the starting address of its interrupt handler is in memory address 3.

Upon entering the interrupt processing phase, the following events will happen:

a. Reset (put 0 in) the *Interrupt Enabled Bit* to disable further nested interrupting
b. The hardware will automatically push all the values of the program counter and general purpose-registers into a system stack
c. Suppose the Address Lines of the system bus is now carrying $k$, where $k$ is a small integer, the $k$th memory word’s value (the starting address of the interrupt handler for the current interrupt source) will be loaded into the CPU program counter, and the CPU starts to run the interrupt handler

5. What will the interrupt handler do?

The interrupt handler will do the following steps:

a. If necessary, save more state information for the interrupted process. For example, maybe the *carry bit* of the ALU is needed by this handler thus needs to be saved.
b. Set (put 1 in) the *Interrupt Enabled Bit* to enable further nested interrupting
c. Do the necessary processing designed for this type of interrupt or trap. Typically, some data will be transferred between a CPU register and one of the registers of an I/O device controller. Since the interrupt is now enabled, it is possible that the execution of this interrupt handler be further interrupted by sources of even higher priority.
d. Restore any information saved in step a.
e. Upon quitting the interrupt handler, pop the values of the program counter and general-purpose registers of the interrupted process back into these registers.
f. Now the CPU can resume the execution of the interrupted process.