What is an Interrupt?

An interrupt is like a phone call that makes the CPU pause whatever it's doing to handle something urgent.

- Example:

Imagine you are writing an assignment, and suddenly, your phone rings. You pause writing, answer the call, and once done, you resume writing from where you left off.

Similarly, when an interrupt occurs, the CPU pauses what it was doing, handles the interrupt, and then continues from where it stopped.

Where Does the CPU Store Information When Interrupted?

When an interrupt occurs, the CPU saves its current state so that it can return to the interrupted task later. The saved information includes:

1. Program Counter (PC) → This keeps track of the instruction that was being executed before the interrupt happened.
2. Registers → These store temporary values (like numbers being used in calculations).
3. Other CPU States (depending on the system).

This saved information is stored in a special area called the Stack (a memory region in RAM).

📌 Think of the stack like a bookmark: When you pause reading a book, you place a bookmark so you can return to the same page later. The CPU does the same thing when an interrupt happens!

How the OS Handles an Interrupt (Step by Step Example: Keyboard Press)

Let’s say you are **typing in Notepad**, and you press the letter "A."

Step 1: Interrupt Occurs

- The keyboard sends a **signal (interrupt request)** to the CPU saying, **"Hey! The user just pressed 'A'!"**

Step 2: CPU Saves Its Work

- The CPU **pauses the current task** (Notepad running).
- It **saves the PC (current instruction address) and registers** in the **stack (RAM)**.

Step 3: CPU Checks the Interrupt Vector Table (IVT)

- The CPU looks up a special table called the **Interrupt Vector Table (IVT)**.
- This table **tells the CPU where to find the correct "Interrupt Service Routine" (ISR) to handle the keyboard input.**

📌 **Think of the IVT like a contact list**: If someone calls you, your phone checks the contact list to find their name.

Step 4: CPU Runs the Interrupt Service Routine (ISR)

- The CPU **jumps to the ISR for keyboard input.**
- The ISR reads **"A"** from the keyboard buffer (temporary storage for key presses).
- The OS processes it and sends it to Notepad.

Step 5: CPU Restores Saved Work and Continues

- The CPU **loads back the saved registers and PC** from the **stack**.
- It **resumes** running Notepad, now displaying the letter **"A"** on the screen.

What is an Interrupt?

An **interrupt** is like a phone call that makes the CPU pause whatever it's doing to handle something urgent.

🔹 Example:
Imagine you are **writing an assignment**, and suddenly, your phone rings. You **pause writing**, answer the call, and once done, you **resume writing** from where you left off.

Similarly, when an **interrupt** occurs, the CPU **pauses** what it was doing, handles the interrupt, and then continues from where it stopped.

Where Does the CPU Store Information When Interrupted?

When an interrupt occurs, the CPU **saves its current state** so that it can return to the interrupted task later. The saved information includes:

1. **Program Counter (PC)** → This keeps track of the instruction that was being executed before the interrupt happened.
2. **Registers** → These store temporary values (like numbers being used in calculations).
3. **Other CPU States** (depending on the system).

This saved information is stored in a special area called the **Stack (a memory region in RAM).**

📌 **Think of the stack like a bookmark**: When you pause reading a book, you place a bookmark so you can return to the same page later. The CPU does the same thing when an interrupt happens!

How the OS Handles an Interrupt (Step by Step Example: Keyboard Press)

Let’s say you are **typing in Notepad**, and you press the letter "A."

Step 1: Interrupt Occurs

- The keyboard sends a **signal (interrupt request)** to the CPU saying, **"Hey! The user just pressed 'A'!"**

Step 2: CPU Saves Its Work

- The CPU **pauses the current task** (Notepad running).
- It **saves the PC (current instruction address) and registers** in the **stack (RAM)**.

Step 3: CPU Checks the Interrupt Vector Table (IVT)

- The CPU looks up a special table called the **Interrupt Vector Table (IVT)**.
- This table **tells the CPU where to find the correct "Interrupt Service Routine" (ISR) to handle the keyboard input.**

📌 **Think of the IVT like a contact list**: If someone calls you, your phone checks the contact list to find their name.

Step 4: CPU Runs the Interrupt Service Routine (ISR)

- The CPU **jumps to the ISR for keyboard input.**
- The ISR reads **"A"** from the keyboard buffer (temporary storage for key presses).
- The OS processes it and sends it to Notepad.

Step 5: CPU Restores Saved Work and Continues

- The CPU **loads back the saved registers and PC** from the **stack**.
- It **resumes** running Notepad, now displaying the letter **"A"** on the screen.

1. What is the Interrupt Vector Table (IVT)?

The **Interrupt Vector Table (IVT)** is a **lookup table** stored in memory that maps each interrupt to its corresponding **Interrupt Service Routine (ISR).**

📌 **Think of IVT like a Contact List:**

- When you get a phone call, you check your contact list to find **who's calling** and **how to respond**.
- Similarly, when an interrupt occurs, the CPU checks the **IVT to find the ISR address** that should handle the interrupt.

---

2. Where is the IVT Stored?

- The IVT is stored **at a fixed location in memory (RAM or ROM, depending on the system).**
- Each entry in the IVT contains the **memory address of the ISR** for a specific interrupt.

3. How Does the CPU Find the Correct ISR?

Step-by-Step Process

🔹 **Step 1: Interrupt Occurs**

- A device (e.g., keyboard) sends an **interrupt request (IRQ)** to the CPU.

🔹 **Step 2: CPU Pauses Current Execution**

- The CPU **saves its current state** (registers, PC) in the stack.

◆ **Step 3: CPU Checks the IVT**

- The CPU gets the **interrupt number** (e.g., `9` for a keyboard press).
- It **looks up** this number in the **Interrupt Vector Table (IVT)**.
- The IVT provides the **memory address of the corresponding ISR** (e.g., `0x0024`).

◆ **Step 4: CPU Jumps to ISR**

- The CPU **transfers control** to the ISR at the found memory address.
- The ISR **executes its code** to handle the interrupt (e.g., reading the pressed key).

◆ **Step 5: ISR Completes and CPU Resumes**

- Once the ISR finishes, the CPU **restores the saved registers and PC** from the stack.
- The CPU **resumes** the interrupted program.

Yes! ✅ **In most operating systems, the child process gets:**

1. **Its own memory space in RAM** → The child gets a separate memory area, though some OSes use **copy-on-write** to temporarily share memory until modifications happen.
2. **Its own Process Control Block (PCB)** → The OS creates a **new PCB** for the child, with a unique **Process ID (PID)** and separate process management information.

However, some resources (like open files) **may still be shared** between parent and child, depending on how the OS handles process creation.

What is a Page Table?

A **page table** is like a **map** that helps the operating system (OS) keep track of where a process's memory is stored in **RAM**.

Why Do We Need a Page Table?

1. **Memory is divided into small blocks** → Instead of giving each process a big chunk of memory, the OS divides memory into **pages** (small fixed-size blocks, like 4KB or 8KB).

2. **A process may not be stored in one place** → A program's data might be scattered across different parts of RAM instead of one continuous block.
3. **The CPU doesn't understand RAM locations directly** → The CPU uses **virtual addresses**, but RAM works with **physical addresses**. The page table **translates virtual addresses to physical addresses**.

How Does a Page Table Work?

4. When a program runs, it uses **virtual memory** (addresses from 0 to some max value).
5. The **page table** translates these **virtual addresses** to **physical addresses** in RAM.
6. The CPU looks at the **page table** every time it accesses memory.

Example:

- Suppose a process wants to access **virtual address 1000**.
- The **page table** checks where that part of memory is stored in **physical RAM** (e.g., at address **5000**).
- The OS retrieves the data from **physical address 5000**.

Key Components of a Page Table

Each entry in the page table contains:

- **Virtual page number** → The page number used by the process.
- **Physical frame number** → The actual location in RAM where the page is stored.
- **Flags** (extra info like read/write permissions, whether the page is in RAM or on disk).

What If the Page Is Not in RAM? (Page Fault)

- If a process tries to access a page that **isn't in RAM**, the OS loads it from the **hard disk (swap space)** into RAM.
- This process is called a **page fault** and can slow down the system

How is Memory Copied?

- **Copy-on-Write (COW):**
    - Modern operating systems use **Copy-on-Write** optimization.
    - This means:
        - **Initially**, the child and parent share the **same physical memory** marked as **read-only**.
        - If **either** tries to **modify** a variable, **only then** a **new copy** is made for that process.
        - This saves memory because copies are made **only when needed**.

1. What is the Process Table?

- It is a **table in the kernel space** that holds an **entry for each process**.
- Each entry **points to the PCB** of a process.
- It acts as an **index** to quickly **locate and manage processes**.

---

2. What's Stored in the Process Table?

Each entry in the process table typically contains:

1. **PID (Process ID)** — Unique identifier for each process.
2. **PPID (Parent Process ID)** — PID of the parent process.
3. **State of the Process** — Running, Ready, Waiting, Zombie, etc.
4. **Pointer to PCB** — To access the complete PCB for that process.

5. **Accounting Information** — CPU usage, execution time, etc.
6. **Scheduling Information** — Priority and scheduling queue pointers.

---

3. How is it Different from PCB?

- The **process table** is like an **index** or a **directory**.
  - It holds **basic information** and **points to the PCB**.
- The **PCB (Process Control Block)** is the **full record** of a process, containing:
  - **Registers, Program Counter, Memory pointers (Page Table), File descriptors**, etc.
  - It is **much larger** and **more detailed** compared to the process table entry.

---

4. Why Do We Need a Process Table?

- It **organizes and manages processes** efficiently.
- The **kernel** uses it to:
  - **Find any process** quickly using its PID.
  - **Schedule processes** by checking their state.
  - **Reap zombies** by looking at their exit status and PID.

• The child has exceeded its usage of some of the resources that it has been allocated. (To determine whether this has occurred, the parent must have a mechanism to inspect the state of its children.) • The task assigned to the child is no longer required. • The parent is exiting, and the operating system does not allow a child to continue if its parent terminates. Some systems do not allow a child to exist if its parent has terminated. In such systems, if a process terminates (either normally or abnormally), then all its children must also be terminated. This phenomenon, referred to as cascading termination, is normally initiated by the operating system