Privileged Instructions

Privileged Instructions are those instructions that can only be executed by the Operating System Kernel or privileged processes such as a device driver. These instructions typically perform operations that require direct access to hardware or other privileged resources, such as setting up memory mappings or accessing I/O devices. The instructions that can only run in Kernel Mode are called Privileged Instructions.

Regular user programs (in user mode) are not allowed to execute these instructions as they can directly affect the hardware or the whole system.

Real-Life Example: Imagine a Car

1. **Driver Seat vs. Passenger Seat**
   - The **driver** (Operating System) has full control of the car (hardware). They can operate the steering, brakes, ignition, etc.
   - The **passenger** (User Program) can only do certain things (like roll the window up/down, adjust their seat). They cannot directly steer or brake.
2. **Privileged Operations**
   - Steering, braking, or shifting gears (Privileged Instructions) can only be done by the driver (OS). If a passenger (user program) tries to grab the wheel unexpectedly, it's illegal or unsafe.
   - Similarly, **privileged instructions** can only be carried out by the OS (the driver). If a normal program (passenger) tries, the system stops it (like a trap or exception).

Characteristics of Privileged Instructions (With Analogies)

1. **Illegal if Attempted by a Passenger**
   - If a passenger tries to grab the steering wheel (i.e., user mode tries to execute a privileged instruction), the driver (hardware) immediately stops it and flags it as unsafe.
   - In a computer, the CPU "traps" this illegal instruction and hands control back to the OS.
2. **Timer Setting**
   - Before letting someone else drive (or while letting the passenger use the car in a controlled environment), the driver sets a timer to reclaim control after a certain period.
   - In a computer, the OS sets a **timer interrupt**. When it goes off, the OS automatically regains control, preventing any one program from running forever.
3. **Modifying the Timer Is Privileged**
   - The driver (OS) is the only one who can change the timer for the car. If anyone else tries to turn it off or reset it, that's privileged.
   - In the OS world, changing the timer (which decides how long a program runs) is **privileged**.
4. **Used by the OS for Correct Operation**
   - Just like a driver needs to do certain tasks (like shifting gears, stepping on the brake) to keep the ride safe and smooth, the OS needs **privileged instructions** to manage resources properly.

Examples of Privileged Instructions

- **I/O Instructions & Halt**
  - *I/O Instructions:* Telling the hardware to read/write from disk, printer, etc.
  - *Halt:* Stopping the CPU from doing any work. These are critical operations only the OS should do.
- **Turning Off All Interrupts**
  - Like turning off all signals or warnings in a car—very dangerous if done at the wrong time. The OS alone should decide when it's safe to do so.
- **Setting the Timer**
  - As mentioned, only the OS can change the "time limit" given to each user program.
- **Context Switching**

    - Switching from one user (process) to another is like the driver deciding who gets to sit in the passenger seat next. Only the OS can do it safely without losing track of who's who.
- **Clearing Memory or Removing a Process from Memory**
    - Imagine removing luggage (process data) from the trunk. Only the driver (OS) should decide which luggage belongs in or out to avoid chaos.
- **Modifying the Device-Status Table**
    - A table that keeps track of which devices (like the car's radio, AC, etc.) are in what state. Only the driver (OS) can change it.

Role of the Operating System in Managing Privileged Instructions

1. **Access Control**
    - The OS decides who can do certain things (like letting only the driver steer the car). This prevents untrusted or malicious programs (passengers) from controlling critical hardware.
2. **Memory Protection**
    - The OS makes sure each process only touches its own "stuff" (memory). In a car analogy, each passenger has their own seatbelt, and they can't randomly unbuckle someone else's seatbelt or move their seat without permission.
3. **Interrupt Handling**
    - When something urgent happens (like a car sensor alert), the driver stops to handle it properly, then resumes. Similarly, the OS saves the state of the running program, handles the interrupt, and then continues.
4. **Virtualization**
    - This is like having a car simulator or "dummy" wheel for the passenger. The OS creates a *virtual environment* where a user can "pretend" they're controlling certain hardware, but they're not actually steering the real car. This keeps things safe and isolated.

Summary

- **Privileged instructions** are like critical car controls—only the driver (OS) can use them.
- **Non-privileged instructions** are like passenger controls (rolling down a window)—safe enough for user applications.
- The **operating system** ensures that these privileged operations remain secure and only accessible to trusted processes, prevents malicious programs from hijacking system resources, and keeps everything running smoothly.

Non-Privileged Instructions

Non-Privileged Access in a Nutshell

- **Non-privileged access** means a program can only execute certain "safe" operations that **do not** directly impact the overall system's security or stability.
- It's like a **passenger** in a car who can do certain things (like roll down the window or change the radio station), but **cannot** take control of the steering wheel or brakes.

Real-Life Car Analogy

1. **Driver Seat vs. Passenger Seat**
    - The **passenger** (user-mode program) has limited access—e.g., adjusting AC temperature, changing music, or rolling down the window.
    - The **driver** (operating system in kernel mode) has privileged access to the steering, pedals, and ignition.
2. **Safe and Localized Actions**

    - In a car, if you adjust your seat or window, you're not affecting how the car drives or endangering others.
    - In a computer system, **non-privileged instructions** (like basic arithmetic, reading/writing your own memory space) don't endanger the entire OS.

Examples of Non-Privileged (User-Mode) Instructions

1. **Arithmetic and Logic**
    - Adding, subtracting, comparing numbers.
    - In a car analogy, adjusting the volume of the radio—safe and localized.
2. **Data Manipulation**
    - Moving data around in your own memory space (like adjusting your seat position—only affects your seat, not the car's engine).
3. **Branching and Jumping**
    - Typical code decisions and loops (like choosing which song to play next).
    - They don't directly control the hardware, just affect how the program (passenger) operates internally.
4. **Reading Files (with Permission)**
    - If the OS permits you, you can read your own files or directories. It's like browsing your personal backpack on the passenger seat.

Non-Privileged Mode Characteristics

1. **Limited Access**
    - No direct hardware control (e.g., user-mode cannot disable interrupts or rewrite hardware registers).
    - Can only request the OS to perform privileged tasks via "system calls," similar to asking the driver for help.
2. **Isolation and Safety**
    - Each program is "sandboxed" in its own memory space, so it doesn't crash or alter the entire system.
    - This is like each passenger in their own seat, with seatbelts to prevent them from moving around dangerously.
3. **Performance vs. Security**
    - Non-privileged instructions can run quickly and freely without needing to switch to kernel mode.
    - However, if you need something more "powerful," you have to ask the OS (like asking the driver to speed up or change route).

Why Do We Have Non-Privileged Mode?

1. **User Convenience and Flexibility**
    - Users (passengers) can do a lot of things on their own without constantly asking the driver to do every minor operation.
    - Programs can perform typical tasks—calculations, user interfaces, etc.—without the overhead of kernel involvement.
2. **Security**
    - By restricting direct hardware access, the OS ensures that malicious or buggy programs can't mess up the system.
    - The passenger can't suddenly slam the brakes, preserving safety for everyone in the car.
3. **System Stability**
    - If a non-privileged program misbehaves or crashes, it only affects that one program. The rest of the system (and other passengers) keep functioning.

    - Like if a passenger spills a drink on themselves, it typically won’t stop the entire car from running.

The Role of the Operating System

- **Gatekeeper**: The OS decides when to grant access to more sensitive resources (e.g., files that belong to someone else, network sockets, or special devices).
- **Resource Manager**: It tracks who is using what (like a driver keeping track of who’s got the seatbelts on, windows up/down).
- **Error Handling**: If a user-mode program tries something illegal, the OS stops it or alerts the user.

Summary

- **Non-privileged access** (user mode) is where typical applications run safely, with a limited set of instructions that do not risk the entire system.
- Just like a car passenger, a user-mode application can do many things, but it cannot control the vehicle’s critical functions.
- This separation (privileged driver vs. non-privileged passenger) helps ensure **security, stability, and efficiency** in modern computing systems.