The probe function in a Linux kernel driver is called when a device matching the driver is found. Internally, it plays a crucial role in binding the driver to the detected hardware device. How the Probe Function Works Internally

1. Device Detection:
   * When a new device (e.g., PCIe, USB, etc.) is connected or detected at boot time, the kernel's bus subsystem (PCI, USB, etc.) scans for devices.
2. Driver Matching:
   * The bus driver (e.g., PCI, USB, I2C, etc.) matches the detected device against registered drivers.
   * This is done using identifiers like Vendor ID, Device ID, Class Codes, etc. (for PCIe) or Product/Vendor IDs (for USB).
   * The matching is defined in structures like pci\_device\_id for PCIe or usb\_device\_id for USB.
3. Probe Function Invocation:
   * If a matching driver is found, the kernel invokes its probe() function.
   * The function signature typically looks like:

static int my\_pci\_probe(struct pci\_dev \*pdev, const struct pci\_device\_id \*id);

* + The pdev structure contains information about the device.

1. Resource Allocation:
   * The probe function typically:
     + Enables the device (e.g., pci\_enable\_device())
     + Requests memory regions (pci\_request\_regions())
     + Maps MMIO/IO regions (ioremap() for MMIO)
     + Allocates and initializes driver structures
     + Registers interrupt handlers if needed (request\_irq())
2. Device Registration:
   * The driver registers itself with the kernel to expose functionality via sysfs, character devices, network interfaces, etc.

When a PCIe device is hot-plugged, the following sequence happens:

1. PCIe Hotplug Mechanism (ACPI or Native PCIe Hotplug):
   * The PCIe root complex detects the new device through link training.
   * The PCIe bus driver receives an interrupt (e.g., PCI\_EXP\_SLTSTA\_PDC for hotplug slots).
   * The kernel PCI subsystem starts scanning for newly connected devices.
2. Enumeration Process:
   * The kernel rescans the PCIe bus using pci\_scan\_slot(), pci\_scan\_child\_bus(), or similar functions.
   * It reads the Vendor ID and Device ID from the PCI configuration space.
   * It checks against registered pci\_driver structures.
3. Matching and Probe Execution:
   * If a driver matches the new device, its probe() function is called.
   * This initializes the device and makes it available to the system.

Does the PCIe Bus Driver Itself Enumerate Devices?

Not directly. The PCIe root complex and the kernel PCI subsystem handle enumeration. The PCI bus driver only detects and reports new devices. The actual enumeration is done by the kernel, and once a device is found, the matching driver’s probe function is executed.

The handoff from the probe function to the kernel PCIe subsystem for enumeration follows a structured process involving several kernel components. Let’s break it down step by step.

1. PCIe Device Detection (Hotplug or Boot Time)

* The PCIe root complex detects a device connection.
* The kernel PCI subsystem scans the PCIe bus to check for new devices.
* The PCIe bus driver (part of the kernel PCI subsystem) detects the new device.

2. PCIe Enumeration by Kernel (Before Probe is Called)

Before the probe function is even involved, the kernel PCI subsystem enumerates the PCIe bus and configures devices.

1. Bus Scanning:
   * The kernel scans the PCIe bus (recursively for bridges) using functions like:

pci\_scan\_child\_bus(struct pci\_bus \*bus)

* + The kernel reads the Vendor ID, Device ID, Class Code, and BARs from PCI config space.
  + It assigns resources like MMIO, I/O, and IRQs.

1. Device Registration:
   * Each discovered device is added to the kernel's device model using:

pci\_device\_add(struct pci\_dev \*dev, struct pci\_bus \*bus)

* + This registers the PCI device with the kernel.

3. Driver Matching & Probe Function Execution

Once the PCI device is registered, the kernel tries to match it with a driver.

1. Driver Matching (Using Vendor & Device ID)
   * Each driver registers a table of supported devices, e.g.:

static const struct pci\_device\_id my\_pci\_table[] = {

{ PCI\_DEVICE(0x1234, 0x5678) }, // Vendor ID, Device ID

{ 0 }

};

MODULE\_DEVICE\_TABLE(pci, my\_pci\_table);

* + The PCI subsystem looks at this table and matches the driver with the newly detected device.

1. Driver Probe Function is Called
   * Once matched, the kernel calls the driver's probe function:

static int my\_pci\_probe(struct pci\_dev \*pdev, const struct pci\_device\_id \*id)

* + The probe function:
    - Enables the device (pci\_enable\_device())
    - Allocates memory (pci\_request\_regions())
    - Maps MMIO/IO regions (ioremap())
    - Sets up interrupts (request\_irq())
    - Initializes the driver and registers interfaces

4. Handoff: Probe Function Completes & PCIe Device is Ready

* Once the probe function successfully executes, the driver is fully bound to the PCI device.
* The device is now operational and can be accessed through the driver.
* The PCI subsystem maintains device information in /sys/bus/pci/devices/.

Flow Summary (End-to-End Handoff)

1. PCIe root complex detects device (hotplug or boot).
2. Kernel PCI subsystem scans PCIe bus, reads Vendor ID, Device ID.
3. Kernel enumerates device (assigns MMIO, I/O, IRQs).
4. Kernel registers device and adds it to /sys/bus/pci/devices/.
5. Kernel matches device with driver using pci\_device\_id.
6. Kernel calls the driver’s probe function.
7. Driver initializes device (MMIO mapping, IRQ setup).
8. Device is ready for use.

Key Takeaways

* Probe is not responsible for enumeration. Enumeration happens first.
* The kernel PCI subsystem does the scanning, registering, and resource assignment.
* The probe function is just the driver's initialization routine after a match.
* The device is available to userspace only after the probe function succeeds.

The journey from hot-plug hardware detection to the probe function execution in the Linux kernel involves multiple hardware and software layers. Below is an exhaustive step-by-step breakdown of the entire flow.

Step 1: PCIe Device Hot Plug Detection (Hardware Level)

1. Physical Connection & Link Training:
   * A PCIe device is inserted into a hot-pluggable PCIe slot.
   * The PCIe root complex (RC) detects a change in the electrical state of the link.
   * Link training is initiated to establish communication between the root complex and the device.
2. PCIe Hotplug Event Signaling:
   * The PCIe switch or root port detects the hotplug event.
   * It generates an interrupt (PCIe Hotplug event) via:
     + PCIe Native Hotplug (Handled by the PCIe root complex)
     + ACPI Hotplug (Handled via ACPI BIOS notifications)
   * The interrupt is forwarded to the CPU.

Step 2: Kernel Interrupt Handling

1. Interrupt Handler Execution:
   * The kernel receives the PCIe hotplug interrupt and processes it.
   * If PCIe Native Hotplug is enabled, the kernel’s PCIe hotplug driver handles the event.
   * If ACPI Hotplug is enabled, the ACPI subsystem notifies the PCI subsystem.
2. Hotplug Driver Notifies the PCI Subsystem:
   * The PCI hotplug driver calls:

pci\_bus\_add\_devices(bus);

* + This triggers a PCI bus rescan.

Step 3: PCIe Bus Rescan & Device Enumeration

1. Kernel Scans the PCI Bus for New Devices:
   * The kernel iterates over all PCIe buses using:

pci\_scan\_child\_bus(struct pci\_bus \*bus);

* + It scans for newly connected devices.

1. PCI Configuration Space Read (Detect New Device):
   * The kernel reads the PCI Configuration Space of each device at Bus:Device:Function (BDF) address.
   * It retrieves:
     + Vendor ID, Device ID (Registers 0x00 - 0x02)
     + Class Code, Subsystem ID
     + BAR (Base Address Registers)
   * If a valid Vendor ID & Device ID are found, the device is considered present.
2. Resource Allocation (MMIO, I/O, IRQs):
   * The kernel assigns MMIO and I/O addresses based on the BARs.
   * IRQs are assigned to the device.
3. PCI Device Registration in Kernel:
   * The kernel adds the new device to the internal PCI device list using:

pci\_device\_add(struct pci\_dev \*dev, struct pci\_bus \*bus);

* + The device appears in /sys/bus/pci/devices/.

Step 4: PCI Driver Matching & Probe Function Execution

1. Match Device with Registered Drivers:
   * The kernel compares the newly detected Vendor ID and Device ID with registered drivers.
   * Drivers register supported devices using:

static const struct pci\_device\_id my\_pci\_table[] = {

{ PCI\_DEVICE(0x1234, 0x5678) }, // Vendor ID, Device ID

{ 0 }

};

MODULE\_DEVICE\_TABLE(pci, my\_pci\_table);

1. Call the Driver’s Probe Function:
   * If a matching driver is found, the kernel invokes the driver’s probe function:

static int my\_pci\_probe(struct pci\_dev \*pdev, const struct pci\_device\_id \*id)

* + The pdev structure contains the device's information.

1. Device Initialization in Probe Function:
   * The driver initializes the device by:
     + Enabling the device:

pci\_enable\_device(pdev);

* + - Requesting MMIO/I/O regions:

pci\_request\_regions(pdev, "my\_pci\_driver");

* + - Mapping BARs (if MMIO is used):

bar0 = pci\_iomap(pdev, 0, pci\_resource\_len(pdev, 0));

* + - Setting up interrupts (if applicable):

request\_irq(pdev->irq, my\_irq\_handler, IRQF\_SHARED, "my\_pci\_driver", dev);

* + - Registering interfaces (e.g., character device, network interface).

1. Probe Function Returns Success:
   * If the probe function returns 0, the device is considered successfully initialized.
   * If an error occurs, the kernel unbinds the driver from the device.

Step 5: Device is Operational

1. Device is Ready for Use:
   * The device driver now fully manages the device.
   * The device appears in /sys/class/ (e.g., /sys/class/net/ for network devices).
   * User-space applications can now interact with the device.

Flow Summary: End-to-End Handoff

1. Hardware Detection (PCIe Hotplug)
   * PCIe link training & electrical detection.
   * PCIe root complex generates a hotplug event.
2. Interrupt Handling (Kernel)
   * Kernel receives hotplug interrupt.
   * PCI hotplug driver processes the event.
3. Enumeration (PCI Subsystem)
   * Kernel rescans PCI bus.
   * Reads PCI Configuration Space.
   * Allocates resources (MMIO, I/O, IRQs).
   * Registers device in /sys/bus/pci/devices/.
4. Driver Binding
   * Kernel matches device with a registered driver.
   * Calls the driver’s probe() function.
5. Device Initialization (Driver)
   * Probe function initializes the device.
   * Sets up MMIO, I/O, IRQs, and interfaces.
6. Device Ready for Use
   * Device is available for user-space applications.

Key Takeaways

* Enumeration happens before the probe function is executed.
* Kernel PCI subsystem handles device discovery, resource allocation, and driver matching.
* The probe function only initializes the device after successful matching.
* If the probe function fails, the kernel unbinds the driver and the device remains unusable.