1) What are device drivers in Computer ?

Device drivers are essential software Components that facilitate communication b/w Computer hardware and the Operating system (OS). They play a Critical role in enabling hardware devices to function properly and efficiently.

Device drivers often Simply known as drivers, Is a Set of files that tells a piece of hardware how to functions communicate with a Computer OS. All piece of hardware require a driver. from our internal Computer Components.

software programs that act as intermediaries b/w hardware devices and the Operating System. There are various types of device drivers. including; kernel mode Drivers., User Mode drivers, Virtual Device Drivers, Filter Drivers, etc.,

Device drivers are indispensable Components of Modern Computing System. They enable the seamless interaction b/w the OS and a wide range of hardware devices. and ensuring System stability, compatability, performance optimization, and understanding the role of drivers is Crucial for troubleshooting hardware issues.

3, How actually . c files are Converted into .exe files ?

c files ~~are~~ it first go through the pre processor, then Compiler compiles it into assembler and creates object file (main.o). Then linker link the main.o with required header objects and libraries and creates a executable file (program.exe)

2. Difference between General purpose Systems and Embedded System.

→ A General-purpose System, like a typical personal computer, is designed to perform a wide range of tasks and can run various software applications. it's flexible and adaptable. with a general operating system like windows, macOs, or Linux.

→ An Embedded System is designed for a Specific function or set of functions. It's often a part of a larger System or product. Such as a microwave oven, automotive control system, or a medical device. Embedded Systems have limited computing resources run Specialized Software, and are optimized for reliability and efficiency.

→ General-purpose Systems Serve a broad range of applications while embedded systems have a Specific, dedicated purpose.

→ General-purpose systems run a variety of Software applications, whereas Embedded systems use Specialized, often proprietary Software.

→ General-purpose systems use standard hardware Components, while Embedded Systems often have custom hardware tailored to their Specific task.

→ General-purpose systems are flexible and can adapt to different tasks, while embedded systems are fixed in their function.

→ G-P S typically run full-featured Operating systems. while E.S ~~have~~ ~~limited~~ ~~resources~~ ~~to~~ may use real-time Operating systems (RTOS) or Custom firmware.

→ General-purpose Systems have ample computing resources (CPU, RAM, Storage); embedded systems have limited resources to meet their Speific requirement.

→ General-purpose systems often have a graphical user Interface (GUI) for user interaction, while embedded Systems may have Simple interfaces or none at all.

→ General-purpose Systems are Versatile and capable of running Various applications. while Embedded systems are purpose-built for Speific tasks and operate with Constraints in term of resources of functionality.

3. How can hardware understand the codes that we write in Embedded systems ?

In Embedded Systems, hardware understands the codes you write through a combination of hardware components and a microcontroller or microprocessor.

Embedded Systems typically have a microcontroller or microprocessor at their core. These are specialized chips designed to execute instructions. They have a CPU (central processing unit) and various peripherals.

When you write code for an embedded system. It's usually in a high-level programming language like C or C++, you use a Compiler to convert this high-level code into machine code (binary code) that the microcontroller can understand.

The machine code is stored in the memory of the embedded system. This memory can be divided into various sections. including program memory (where the code resides) and data memory (for variables and data).

The CPU of the microcontroller fetches instructions from the program memory one by one. These instructions are in the form of binary code. which corresponds to specific operations.

The microcontroller has interface that allow it to interact with external hardware devices or sensors. These interfaces can be controlled by writing specific code that manipulates the hardware registers associated with them.

Compiler/Assembler: Developers write code in a high-level programming language or assembly language. They use a compiler or assembler to convert this human readable code into machine code, which Consists of binary instructions that the CPU can understand.

Loading the code: The Compiled machine code is loaded onto the microcontrollers flash memory or non-volatile storage. This is typically done during the programming which can be done using a special programmer tool or through interfaces like USB, UART.

Execution: When the embedded s/m is powered on or reset, the cpu fetches instructions from memory & executes them sequentially.

pheripherals & Sensors: During execution, the cpu interacts with various hardware pheripherals & sensors through the defined I/O interfaces. This includes reading data, controlling actuators, & responding to external events

Feedback & control: The code often includes logic for decision making and control.

4. Difference b/w RTOS & General purpose Operating Systems.

Real-Time Operating Systems (RTOS)	General-purpose operating Systems (GPOS)
1) RTOS is designed for application that require precise and deterministic timing, often in embedded systems, robotics, and control systems. It ensures that tasks are executed within Specified time Contraints.	1) GPOS, also known as Standard desktop or Server operating systems (Eg., windows, Linux, macOs), are designed for General-purpose Computing tasks. They are Versatile and cater to a wide range of applications.
2) RTos provides deterministic behavior, meaning it guarantees that tasks will Complete within predefined time frames. This is Critical for System where timing is Crucial, such as aviation, medical devices, and industrial automation.	2) It does not guarantee deterministic behaviour & can have variable response times, making it less Suitable for RT applications
3) Scheduling: It uses priority-based Scheduling algorithms to prioritize tasks with different levels of importance	3) It typically uses a time sharing or multilevel queue sheduling algorithm, focusing on efficient resource utilization.
4) Resource Management: It efficiently manages s/m resources with minimal over head.	4) It offers resource sharing among multiple applications & users but this may introduce more over head.
5) Complexity: It is generally light weight and designed to be Simple & fast, focusing on real time responsiveness	5) It tends to be more complex and feature-rich Supporting a wide range of applications & Services.
6) Example of RTOS: FreeRTO, QNX	6) Examples of GPOS: windows, Linux, macos, Android.