EMBEDDED SYSYEM AND INTERNET OF THINGS

1. What is the boot process of a computer and what is a BIOS?

The boot process of a computer is the sequence of steps that occur when a computer starts up. This process begins when the power button is pressed, and it ends when the operating system is fully loaded and ready to use.

BIOS (Basic Input/Output System) is a firmware that is built into the computer's motherboard. It is responsible for initializing and testing the hardware components of the computer, including the CPU, memory, and storage devices. BIOS also provides a set of low-level routines that allow the operating system to communicate with the hardware. During the boot process, BIOS is the first program that is loaded into memory, and it is responsible for starting up the computer and loading the operating system.

2. How operating system works? List down 5 tasks of an OS.

An operating system (OS) is a software that manages computer hardware and software resources, and provides common services for computer programs. Here are five tasks of an OS:

1. Process Management: The OS manages and schedules processes (programs) that are running on the computer, allocating CPU time, memory, and other resources.

2. Memory Management: The OS allocates and manages memory resources for programs, ensuring that each program has enough memory to run properly.

3. Device Management: The OS manages input/output (I/O) devices such as keyboards, mice, printers, and network cards, providing a common interface for programs to access these devices.

4. File Management: The OS manages files and directories on storage devices such as hard drives, allowing users and programs to create, read, write, and delete files.

5. Security Management: The OS provides security features such as user authentication, access control, and encryption, to protect the computer and its data from unauthorized access or modification.

3. What is a Bootloader and how does it work?

A bootloader is a program or code that is responsible for loading and initializing the operating system (OS) or firmware of a device. It is typically located in the device's non-volatile memory, such as the read-only memory (ROM) or flash memory.

When a device is powered on or reset, the bootloader is the first code that runs. It performs a series of checks to verify the integrity of the OS or firmware, and then loads it into the device's random-access memory (RAM). The bootloader also initializes the hardware components of the device, such as the processor, memory, and input/output (I/O) interfaces.

The bootloader may also provide additional features, such as the ability to update the OS or firmware, configure system settings, or perform diagnostic tests. Once the bootloader has completed its tasks, it transfers control to the loaded OS or firmware, which takes over the operation of the device.

In summary, the bootloader is a critical component of a device's boot process that ensures the proper loading and initialization of the OS or firmware.

4. What is Real-time operating system?

A real-time operating system (RTOS) is an operating system designed to handle real-time applications that require predictable and deterministic response times. This means that the OS must be able to respond to events or inputs within a specified time frame, typically in microseconds or milliseconds. RTOS is commonly used in embedded systems, industrial automation, medical devices, and other applications where timing is critical. RTOS typically prioritizes tasks based on their importance and urgency, and uses preemptive scheduling to ensure that high-priority tasks are executed first.

5. List the differences between BareMetal vs RTOS programming?

Baremetal programming:

1. Baremetal programming is the process of writing software directly on hardware without any operating system.

2. It is typically used in low-level applications where performance and efficiency are critical.

3. Baremetal programming requires a deep understanding of hardware and low-level programming languages such as assembly language.

4. Baremetal programming provides complete control over the hardware, allowing for highly optimized and efficient code.

5. Baremetal programming is not scalable and can become complex as the system grows in size and complexity.

RTOS programming:

1. RTOS programming involves writing software for an operating system that manages system resources and provides services to applications.

2. It is typically used in complex applications where multiple tasks need to be executed simultaneously.

3. RTOS programming requires knowledge of high-level programming languages such as C or C++ and an understanding of operating system concepts.

4. RTOS programming provides a higher level of abstraction, allowing for easier development and maintenance of code.

5. RTOS programming is scalable and can handle large and complex systems with ease.

6. How to choose between BareMetal and RTOS for project?

If the project requires high performance and efficiency, with complete control over hardware, baremetal programming may be the better choice. This is often the case in embedded systems, where resources are limited and real-time responsiveness is critical.

the other hand, if the project involves multiple tasks that need to be executed simultaneously, with a need for abstraction and easier development and maintenance of code, RTOS programming may be the better choice. This is often the case in complex systems such as aerospace and defense, medical devices, and industrial automation.

Other factors to consider when choosing between baremetal and RTOS programming include the complexity of the system, the availability of hardware resources, the level of expertise of the development team, and the cost and time constraints of the project. Ultimately, the choice should be based on a careful analysis of these factors and a thorough understanding of the requirements and constraints of the project.