29/9/23 Embedded Systems

Assignment - I

1) Difference between General purpose system & Embedded system?

General purpose s/m	Embedded s/m
1) It is designed for a wide range of applications & tasks. Such as personal computers, laptops & servers.	1) It is specialized systems design to perform specific functions or tasks within a larger s/m. Such as control s/m in a car.
2) Flexibility: They are highly flexible & can run a wide range of software applications.	2) They are inflexible & typically run a fixed set of instructions for a specific purpose.
3) Hardware Resources: These s/ms have more abundant hardware sources like CPU power, memory & storage.	3) These are resource-constrained & designed with minimal hardware needed for their specific func^{n}.
4) Software complexity: They can run complex o.s and a wide variety of software applications.	4) They often use RTOS and have specialized software tailored to their specific task.
5) User Interaction: They are typically designed for U.I & have I/P/O/P interfaces like keyboards & mouse.	5) It is minimal or non-existent; they have limited interface like buttons or simple displays.
6) Ex: Personal computers, smart-phones, and servers. high power consumption based on	6) Ex: Automative control units, home appliances & industrial automation s/m, Webcam, calculator
	7) less power consumption. Response time is high.

2) What are device drivers? applications

Device drivers are software programs that enable communication b/w an o.s (such as windows, macos, linux) & hardware devices (such as printers, graphic cards or keyboards) they act as intermediaries, translating high-level o.s commands into low-level instructions that the hardware can understand.

* It serve crucial functions:

1. Hardware Interaction: They allow the o.s to interact with hardware components, ensuring that the right signals & data are sent to & received from devices.
2. Stability & Compatibility: Drivers help ensure the stability & compatibility of hardware with the o.s. without the appropriate driver, a device may not function correctly or at all.
3. Performance Optimization: It can optimize the performances of

hardware by implementing efficient communication protocols & utilizing hardware features effectively.

4. Error Handling: they manage errors & exceptions that may occur during device operation, helping to prevent system crashes or data loss.

5. Updates & maintenance: Manufacturers often release driver updates to improve compatibility, fix bugs, or enhance performance. Users may need to update drivers periodically.

* Device drivers can be categorized into various types, including
  - Built-in Drivers: Some basic drivers for common hardware components are included with the O.S.
  - Vendor-supplied Drivers: Manufacturers provide specific drivers for their hardware products, which can be downloaded & installed separately.
  - Generic Drivers: In some cases, generic drivers are available for a category of devices, and they may work with various hardware brands.
  - Open Source Drivers: Some hardware especially in the Linux community, has OSD developed & maintained by the community.

* Installing the right drivers is essential to ensure that hardware functions correctly & efficiently on your computer.

Why device drivers are important?

It is important because they serve as a bridge b/w hardware devices & O.S on a computer.

1. Hardware Communication: It enables the OS to communicate with & control hardware components like graphics cards, printers & more.
2. Compatibility: It ensure that hardware devices are compatible with a wide range of O.S.
3. Bug fixes & updates: It can be updated to fix bugs, improve compatibility, & add new features to hardware devices.
4. Security: It plays a role in system security by providing a layer of isolation b/w the hardware & the OS. They can help prevent unauthorized access or interference with critical hardware components.

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

In embedded systems, hardware understands the codes written by developers through a combination of hardware components & the software (code) that runs on them.

How it generally works:

1. Processor (CPU): At the heart of most embedded systems is a microcontroller or microprocessor, which is essentially a specialized computer. The CPU executes instructions from memory.
2. Memory: Embedded systems have memory, which is used to store both program code & data. This memory is often divided into two main types.
   - Flash/ROM: This is where the program code is stored. It's non-volatile, meaning it retains its data even when the power is turned off.
   - RAM: This is used for temporary data storage during program execution.
3. Input/output (I/O) Interfaces: Embedded systems are connected to the outside world through I/O interfaces, including GPIO (General-purpose Input/output) pins, serial ports, analog to-digital converters, etc. These interfaces allow the hardware to interact with external devices & sensors.
4. 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.
5. Loading the code: The compiled machine code is loaded onto the microcontroller's 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.
6. Execution: When the embedded s/m is powered on or reset, the CPU fetches instructions from memory & executes them sequentially.
7. Peripherals & Sensors: During execution, the CPU interacts

with various hardware peripherals & sensors through the defined I/O interfaces. This includes reading sensor data, controlling actuators, & responding to external events.

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

In summary, the hardware understands the code in E.S because the code is compiled into machine code that matches the CPU's instruction set. The CPU fetches & executes these instructions, and the code interacts with various hardware components & peripherals to achieve the desired functionality in the E.S.

11) Difference between RTOS and general purpose OS?

RTOS	GPOS
1) purpose: It is designed for applications that require precise and predictable timing, making it suitable for real time & E.S	1) It is designed for general purpose computing tasks and offers more flexibility to run a wide range of applications
2. Determinism: It provides deterministic behaviour, ensuring that tasks meet specific timing deadlines. It guarantees a response within a fixed time frame.	2) It doesn't 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 scheduling algorithm, focusing on efficient resource utilization.
4. Resource Management: It efficiently manages s/m resources with minimal overhead.	4) It offers resource sharing among multiple applications & users, but this may introduce more overhead
5. Complexity: It is generally lightweight 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. Examples of RTOS: FreeRTOS, VxWorks & QNX.	6) Examples of GPOS. Windows, Linux, macOS & Android.