CMR
INSTITUTE OF
TECHNOLOGY

USN

Internal Assesment Test - III

Sub:	Embedded Computing Systems							Code:	10CS72
Date:	15/11/2016	Duration:	90 mins	Max Marks:	50	Sem:	VII	Branch:	CSE
Answer Any FIVE FULL Questions									

		Marks	OBE	
			CO	RBT
1(a)	**What is Task control block (TCB)? Explain the structure of TCB.**	[2]	CO2	L2
(b)	**Three processes with process IDs P1, P2 and P3 with estimated completion time 8, 5, 4 milliseconds, respectively, enters the queue together in the order P2, P3, P1. Process P4 with estimated execution time 4 milliseconds entered the 'Ready' queue 3 milliseconds later the start of execution of P1. Calculate the waiting time and Turn Around Time (TAT) for each process and the average waiting time and Average Turn Around Time (Assuming there is no I/O waiting) for the processor using RR algorithm with time slice=2ms.**	[8]	CO3	L3
2(a)	**List different types of multitasking.**	[2]	CO1	L1

1(a) **What is Task control block (TCB)? Explain the structure of TCB.** [2] CO2 L2

A Task Control Block (TCB) is a data structure that is used for holding the information corresponding to a task.

Structure of TCB:

Task ID: Task identification number

Task State: Current state (e.g., running, idle, etc.)

Task type: Hard real-time/soft-real-time, or background task.

Task Priority: Priority of the task

Task Context Pointer: Pointer for saving context.

Task Memory Pointers: Pointer to code memory, data memory, and stack memory

Task System Resource Pointers: Pointers to system resources (semaphores, mutex, etc.)

Task Pointers: Pointers to other TCBs

Other Parameters: Other relevant task parameters.

(b) **Three processes with process IDs P1, P2 and P3 with estimated completion time 8, 5, 4 milliseconds, respectively, enters the queue together in the order P2, P3, P1. Process P4 with estimated execution time 4 milliseconds entered the 'Ready' queue 3 milliseconds later the start of execution of P1. Calculate the waiting time and Turn Around Time (TAT) for each process and the average waiting time and Average Turn Around Time (Assuming there is no I/O waiting) for the processor using RR algorithm with time slice=2ms.** [8] CO3 L3

P2	P3	P1	P2	P3	P1	P4	P2	P1	P4	P1
0 2	3-4	5-6	7-8	9-10	11-12	13-14	15	16-17	18-19	20-21

Process	Waiting Time (ms)	TAT (ms)
P1	13	21
P2	10	15
P3	6	10
P4	8	12
AVG	9.25	14.5

2(a) **List different types of multitasking.** [2] CO1 L1

1. Cooperative multitasking

2. Preemptive multitasking
3. Non-preemptive multitasking

(b) **What is advanced configuration and power interface? Explain the basic global power states supported by ACPI.** [8] CO2 L2

Advanced Configuration and Power Interface (ACPI) is an open industry standard for power management services. ACPI provides some basic power management facilities and abstracts the hardware layer, the OS has its own power management module that determines the policy, and the OS then uses ACPI to send the required controls to the hardware.

ACPI supports the following five basic global power states:

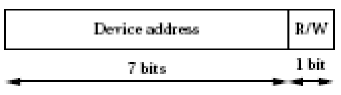
- G3, the mechanical off state, in which the system consumes no power.
- G2, the soft off state, which requires a full OS reboot to restore the machine to working condition. This state has four sub-states"
    - S1, a low wake up latency state with no loss of system context.
    - S2, a low wake up latency state with loss of CPU and system cache state.
    - S3, a low wake up latency state in which all system states except for main memory is lost, and
    - S4, the lowest power sleeping state in which all devices are turned off.
- G1, the sleeping state in which the system appears to be off and the time required to return to working condition is inversely proportional to power consumption.
- G0, the working state in which the system is fully usable.
- The legacy state in which the system does not comply with ACPI.

3(a) **Define distributed embedded system.** [2] CO1 L1

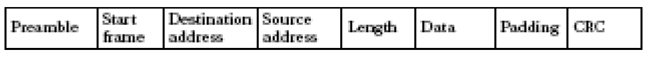
An embedded system built around a network or one in which communication between processing elements is explicit.

(b) **Illustrate with a neat diagram, the packet structure of the following: (i) I^{2}C (ii) Ethernet (iii) CAN (iv) IP** [8] CO1 L1

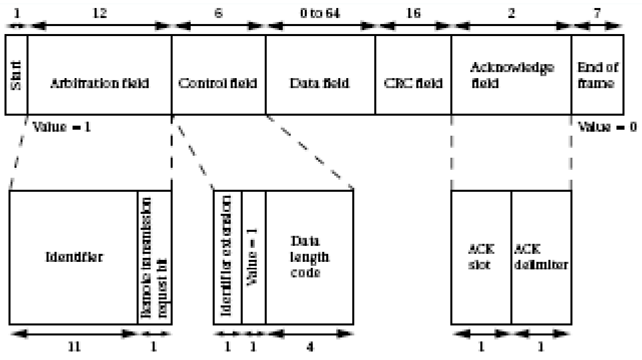
(i) I2C



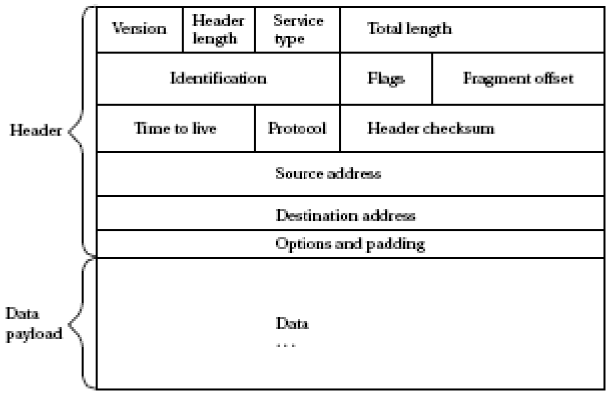
(ii) Ethernet



(iii) CAN



(iv) IP



4(a) **Define IDE. Explain briefly its role in embedded software development.** [2] CO1 L1

Integrated Development Environment (IDE) is a software package which bundles a Text Editor (Source code editor), Cross-Compiler (cross-platform development), linker and a debugger.
IDE provides an integrated environment for developing and debugging the target processor specific embedded firmware.

(b) **Explain the following terms:** [8] CO2 L2
(i) Disassembler (ii) Decompiler (iii) Debugging (iv) Boundary scan

Disassembler: Utility program which converts machine code into target processor specific assembly code/instructions.
Decompiler: Utility program for translating machine codes into corresponding high level language instructions.
Debugging: The process of diagnosing the firmware execution, monitoring the target processor's registers and memory while the firmware is running and checking the signals from various buses of the embedded hardware.
Boundary Scan: Technique for testing the interconnection among the various chips, which support boundary scanning, in a complex board containing too many interconnections and multiple planes for routing.

5(a) **What is Inter Process Communication?** [2] CO1 L1

Inter Process Communication (IPC) is the mechanism provided by the OS as part of the process abstraction through which the processes/tasks communicate with each other.

(b)	**Explain the following IPC techniques:** **(i) Shared Memory (ii) Message Passing (iii) Remote Procedure Call**	[8]	CO2	L2

1. Shared Memory
   Processes share some area of the memory to communicate by the process is written to the shared memory area. Other processes which require this information can read the same from the shared memory area.

   Some of the different mechanisms adopted by different kernels are as below:

   a. Pipes: Pipe is a section of the shared memory used by processes for communicating. Pipes follow the client-server architecture. A process which creates a pipe is known as a pipe server and a process which connects to a pipe is known as pipe client. It can be unidirectional, allowing information flow in one direction or it can be bidirectional, allowing bi-directional information flow. Generally, there are two types of pipes supported by the OS. They are:
      Anonymous pipes: They are unnamed, unidirectional pipes used for data transfer between two processes.
      Named Pipes: They are named, unidirectional or bidirectional for data exchange between two processes.
   b. Memory Mapped Objects: This is a shared memory technique adopted by some real-time OS for allocating shared block of memory which can be accesses by multiple processes simultaneously. In this approach, a mapping object is created and physical storage for it is reserved and committed. A process can map the entire committed physical area or a block of it to its virtual address space. All read-write operations to this virtual address space by a process are directed to its committed physical area. Any process which wants to share data with other processes can map the physical memory area of the mapped object to its virtual memory space and use it for sharing the data.

2. Message Passing
   Message passing is an (a)synchronous information exchange mechanism used for Inter Process/Thread communication. The major difference between shared memory and message passing is that through shared memory lots of data can be shared whereas only limited amount of data is passed through message passing. Also, message passing is relatively fast and free from synchronization overheads compared to shared memory. Based on the message passing operation between the processes, message passing is classified into:

   a. Message Queue: Usually the process which wants to talk to another process posts the message to a First-In-First-Out (FIFO) queue called ‘message queue’, which stores the message temporarily in a system defined memory object, to pass it to the desired process. Messages are sent and received through *send* and *receive* methods. The messages are exchanged through the message queue. It should be noted that the exact implementation is OS dependent. The messaging mechanism is classified into synchronous and asynchronous based on the behavior of the message posting thread. In asynchronous messaging, the message posting thread just posts the message to the queue and it will not wait for an acceptance (return) from the thread to which the message is posted. Whereas in synchronous messaging, the thread which the message is posts the message enters waiting state and waits for the message result from the thread to which the message is posted. The thread which invoked the send message becomes blocked and the scheduler will not pick it up for scheduling.

b. Mailbox: Mailbox is an alternative to ‘message queues’ used in certain RTOS for IPC, usually used for one way messaging. The thread which creates the mailbox is known as ‘mailbox server’ and the threads which subscribe to the mailbox are known as ‘mailbox clients’. The mailbox server posts messages to the mailbox and notifies it to the clients which are subscribed to the mailbox. The clients read the message from the mailbox on receiving the notification. The process of creation, subscription, message reading and writing are achieved through OS kernel provided API calls.

c. Signaling: Signaling is a primitive way of communication between processes/threads. *Signals* are used for asynchronous notifications where one process/trhead fires a signal, indicating the occurrence of a scenario which the other process(es)/thread(s) is waiting. Signals are not queued and they do not carry any data.

3. Remote Procedure calls and Sockets

Remote Procedure Call (RPC) is the IPC mechanism used by a process to call a procedure of another process running on the same CPU or on a different CPU which is interconnected in a network. In object oriented language terminology RCP is also known as *Remote Method Invocation (RMI)*. RPC is mainly used for distributed applications like client-server applications. The CPU/process containing the procedure which needs to be invoked remotely is known as server. The CPU/process which initiates an RPC request is known as client.

Sockets are used for RPC communication. Socket is a logical endpoint in a two-way communication link between two applications running on a network. Sockets are of different types, namely, Internet Sockets (INET), UNIX sockets, etc. The INET sockets works on internet communication protocols, such as TCP/IP and UDP. They are classified into stream sockets and datagram sockets.

Stream sockets are connection oriented, and they use TCP to establish a reliable connection.

Datagram sockets rely on UDP for communication. The UDP connection is unreliable when compared to TCP.

6(a)	**What is Task synchronization?**	[2]	CO1	L1

The act of making processes aware of the access of shared resources by each process to avoid conflicts is known as Task synchronization. Task synchronization is essential for 1. Avoiding conflicts in resource access (racing, deadlock, livelock, starvation) in a multitasking environment, and 2. Ensuring proper sequence of operation across processes. E.g. producer-consumer problem.

(b)	**Explain the synchronization issues in resource utilization. Using Dining Philosopher's problem, mention the solutions for those issues.**	[8]	CO2	L2

Synchronization issues:

1. Racing: It is the situation in which multiple processes compete (race) each other to access and manipulate shared data concurrently. In a race condition, the final value of the shared data depends on the process which acted on the data finally.
2. Deadlock: It is a condition in which a process is waiting for resource held by another process, which in turn is waiting for a resource held by the first process. In this state, none of the processes are able to make any progress in their execution due to the cyclic dependency of resources. This ends up in none of the resources being utilized.
   The following conditions favour a deadlock occurrence:
   Mutual exclusion: The criteria that only one process can hold a resource at a time.
   Hold and Wait: The condition in which a process holds a shared resource by acquiring the lock controlling the shared access and waiting for an additional resources held by other processes.
   No Resource Preemption: The criteria that OS cannot take back a resource from a process which is currently holding it and the resource can only be released voluntarily by the process holding it.
   Circular Wait: A process is waiting for resource held by another process. E.g., Pa blocked by Pb for resource. Pb blocked by Pa for resource.
3. Livelock: It is a condition in which a process always does something but is unable to make any progress towards execution completion. Here, progress

seems to happen all the time but actually no real execution takes place. This is similar to the situation 'always busy, doing nothing'. E.g. Both Pa and Pb needs x and y for completion.
Step 1: Pa holds x, Pb holds y
Step 2: Pa drops x, Pb drops y
Repeat steps 1 and 2

4. Starvation: It is a condition in which a process does not get the resources required to continue its execution for a long time.

Task synchronization issues in dining philosopher's problem:
Race condition: When a philosopher is about t o pick up the right fork, the philosopher sitting to his right tries to grab the left of his, which happens to be the right fork of the former.
Deadlock: All the philosophers involve in brainstorming together and eat together. Each philosopher picks up the left fork first, and unable to proceed further because right fork is missing.
Livelock: All philosophers pick the left fork at the same time, but notice the right fork is missing. Hence, they put back their left fork. As soon as they see the forks on the table they repeat the same process.
Starvation: Both deadlock and livelock discussed above leads to starvation.

Solution to dining philosophers addressing task synchronization issue:
Round robin allocation and FIFO allocation are valid solutions to the dining philosophers' problem. But these enable only one philosopher to eat at any given time, which is sub-optimal solution.
A better solution will be to impose rules in accessing forks as follows:

1. When a philosopher is hungry, s/he picks the left fork first.
2. If the right fork is not available, the philosopher puts down the left fork, and waits for a random interval of time before making the next attempt.

Another solution is that, each philosopher requires a semaphore (mutex) before picking up any fork. When a philosopher feels hungry s/he checks whether the philosopher of the left and right is already using the fork (checking the associated semaphore). If the forks are in use by the neighboring philosophers, the philosopher waits till the forks are available. A philosopher when finished eating puts the forks down and informs the philosophers sitting to the left and right, who are hungry, by signaling the semaphores associated with the forks.

7(a) **Define RTOS.** [2] CO1 L1

RTOS stands for Real-Time Operating System, which is a type of operating system that implements policies and rules concerning time-critical allocation of system resources. RTOS decides which applications should run in which order, and how much time needs to be allocated for each application. E.g.

Windows CE, QNX, VxWorks MicroC/OS-II.

(b) **Explain the different functional and non-functional requirements while choosing RTOS.** [8] CO2 L2

1. Functional Requirements
    a. Processor Support: It is not necessary that all RTOS support all kinds of processor architecture. It is essential to ensure the processor support by the RTOS.
    b. Memory Requirements: The OS requires ROM memory for holding the OS files and it is normally stored in a non-volatile memory like FLASH. OS also requires working memory RAM for loading the OS services. Since embedded systems are memory constrained, it is essential to evaluate the minimal ROM and RAM requirements for the OS under consideration.
    c. Real-time capabilities: It is not mandatory that the OS for all embedded systems need to be real-time and all embedded systems are 'real-time' in behavior. The task/process scheduling policies plays an important role in the 'real-time' behavior of an OS. Analyze the real-time capabilities if the OS under consideration and the standards met by the OS for real time capabilities.
    d. Kernel and Interrupt Latency: The kernel of the OS may disable interrupts while executing certain services and it may lead to interrupt latency. For and embedded system whose response requirements are high, this latency should be minimal.
    e. Inter Process Communication and Task Synchronization: The implementation of IPC and Synchronization is OS kernel dependent. Certain kernels may provide a bunch of options whereas others provide very limited options. Certain kernels implement policies for avoiding priority inversion issues in resource sharing.
    f. Modularization support: Most of the OS provide a bunch of features. At times it may not be necessary for an embedded product for its functioning. It is very useful if the OS supports modularization in the developer can choose the essential modules and re-compile the OS image for functioning. E.g. Windows CE.
    g. Support for Networking and Communication: The OS kernel may provide stack implementation and driver support for a bunch of communication interfaces and networking. Ensure that the OS under consideration provides support for all the interfaces required by the embedded product.
    h. Development and Debugging Support: Certain OS include runtime libraries required for running applications written in languages like Java and C#. A Java Virtual Machine (JVM) customized for the OS is essential for running java applications.

2. Non-Functional Requirements
    a. Custom Developed or Off the Shelf: Depending on the OS requirement, it is possible to go for the complete development of an OS suiting the embedded system needs or use an off the shelf, readily available OS, which is either a commercial product or an Open Source product, which is in close match with the system requirements. Sometimes it may be possible to build the required features by customizing the Open Source OS. The decision is purely dependent on the development cost, licensing fees for the OS, development time and availability of skilled resources.
    b. Cost: The total cost for developing or buying the OS and maintaining it in terms of commercial product and custom build needs to be evaluated before taking a decision on the selection of OS.
    c. Development and Debugging Tools Availability: The availability of development and debugging tools is a critical decision making factor in the selection of an OS for embedded design. Certain Operating Systems may be superior in performance, but the availability of tools for supporting the development may be limited.
    d. Ease of Use: How easy is it to use a commercial RTOS is another important feature that needs to be considered in the RTOS selection.
    e. After Sales: For a commercial embedded RTOS, after sales in the form of e-mail, on-call services, etc for bug fixes, critical patch updates and support for production issues, etc should be analyzed thoroughly.

8(a) **List the different files generated during the cross-compilation of an Embedded C file.** [2] CO1 L1

1. List file (.lst)
2. Preprocessor output file
3. Object file (.obj)
4. Map file (.map)
5. Hex file (.hex)

(b) **Explain the following terms:**
(i) Host (ii) Target (iii) ICE (iv) OCD [8] CO2 L2

Host: In typical embedded computing system development, it is common to do at least part of the software development on a PC or workstation. This system is known as the host.
Target: The hardware on which the code will finally run is known as the target.
ICE: In-circuit emulator is a hardware device for emulating the target CPU for debug purpose.
OCD: In On-Chip Debugging, the processors/controllers incorporate dedicated debug module to the existing architecture for controlling debugging.

Course Outcomes		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1:	Identify and describe the hardware & software components and functional & non-functional features of embedded systems.	1	-	-	-	-	-	-	-	-	-	-	-
CO2:	Explain the functionalities and various challenges faced in embedded computing.	1	1	-	-	-	-	-	-	-	-	-	-
CO3:	Apply the principles of system design process and implement each design phase.	2	1	-	-	1	-	-	-	-	-	-	-
CO4:	Analyze existing embedded system applications, and their relationship between different hardware and software components.	2	3	-	2	2	2	2	-	1	1	-	-
CO5:	Test designs at different levels using verification and validation techniques.	2	2	-	2	3	-	-	-	1	1	-	-
CO6:	Design solutions to overcome limitations in existing embedded system application.	3	3	3	3	3	2	2	2	3	3	2	2

Cognitive level	KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

PO1 - *Engineering knowledge*; PO2 - *Problem analysis*; PO3 - *Design/development of solutions*; PO4 - *Conduct investigations of complex problems*; PO5 - *Modern tool usage*; PO6 - *The Engineer and society*; PO7- *Environment and sustainability*; PO8 – *Ethics*; PO9 - *Individual and team work*; PO10 - *Communication*; PO11 - *Project management and finance*; PO12 - *Life-long learning*

GOOD LUCK!