USN | | | | | | | | | | |

Department of AI-ML and AI-DS

Internal Assessment Test 3 – September 2023

Sub:	Microcontroller & Embedded Systems					Sub Code:	21CS43	Branch:	AI-ML & AI-DS
Date:	09-09-23	Duration:	90 min's	Max Marks:	50	Sem / Sec:	4th		OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1.	Write a short note on utility of LED and Seven segment display as actuators	[10]	CO4	L3
2.	Discuss the architecture and working of Inter Integrated Circuit or I2C or I^2C protocol	[10]	CO4	L3
3.	Explain the construction and working of Reset and Brownout protection circuit.	[10]	CO4	L3
4.	Discuss in detail about the OS architecture for RTOS based embedded system design	[10]	CO5	L2
5.	Elaborate the functions of the Real-Time Kernel	[10]	CO5	L2
6.	Explain the concept of Mailbox and the Sockets for IPC	[10]	CO5	L2
7.	Define Deadlock. Explain the Racing conditions due to deadlock and methods to handle deadlock	[10]	CO5	L3

USN | | | | | | | | | | |

Department of AI-ML and AI-DS

Internal Assessment Test 3 – September 2023

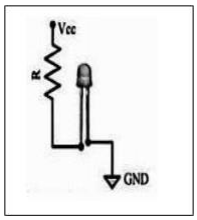
Sub:	Microcontroller & Embedded Systems					Sub Code:	21CS43	Branch:	AI-ML & AI-DS
Date:	09-09-23	Duration:	90 min's	Max Marks:	50	Sem / Sec:	4th		OBE

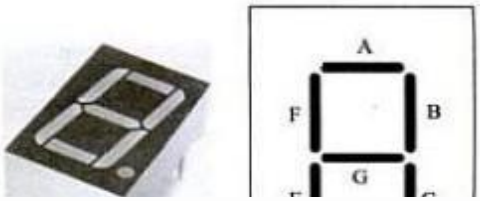
	Answer any FIVE FULL Questions	MARKS	CO	RBT
1.	Write a short note on utility of LED and Seven segment display as actuators	[10]	CO4	L3
2.	Discuss the architecture and working of Inter Integrated Circuit or I2C or I^2C protocol	[10]	CO4	L3
3.	Explain the construction and working of Reset and Brownout protection circuit.	[10]	CO4	L3
4.	Discuss in detail about the OS architecture for RTOS based embedded system design	[10]	CO5	L2
5.	Elaborate the functions of the Real-Time Kernel	[10]	CO5	L2
6.	Explain the concept of Mailbox and the Sockets for IPC	[10]	CO5	L2
7.	Define Deadlock. Explain the Racing conditions due to deadlock and methods to handle deadlock	[10]	CO5	L3

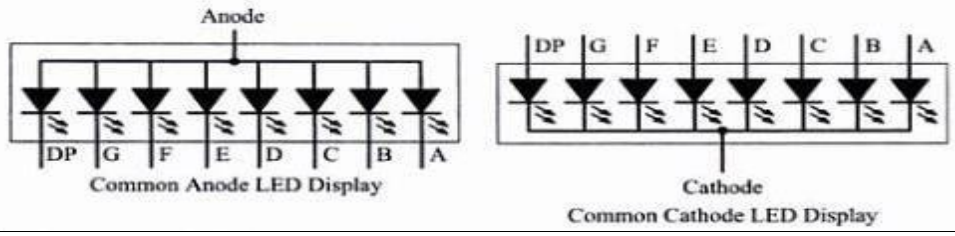
CI: **CCI:** **HOD:**

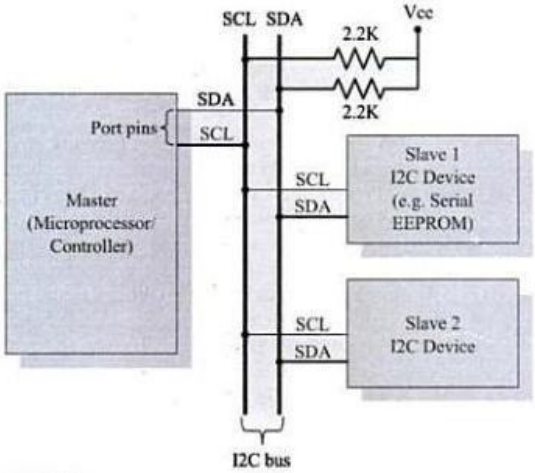
Internal Assessment Test 3 – September 2023

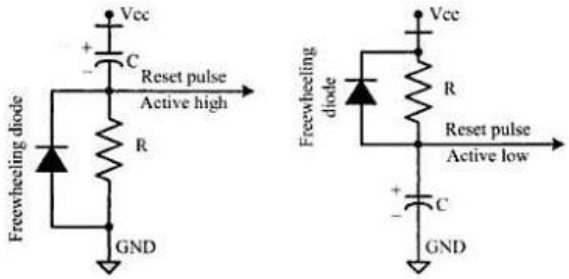
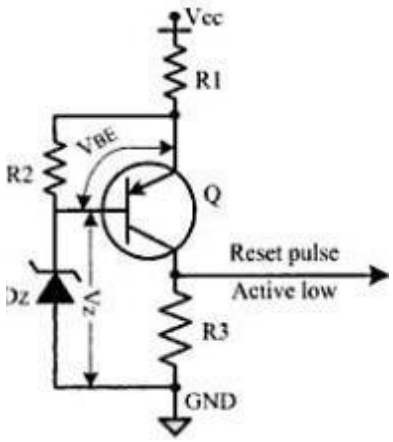
Sub:	Microcontroller & Embedded Systems					Sub Code:	21CS43	Branch:	AI-ML & AI-DS
Date:	09-09-23	Duration:	90 Minutes	Max Marks:	50	Sem / Sec:	4th		OBE

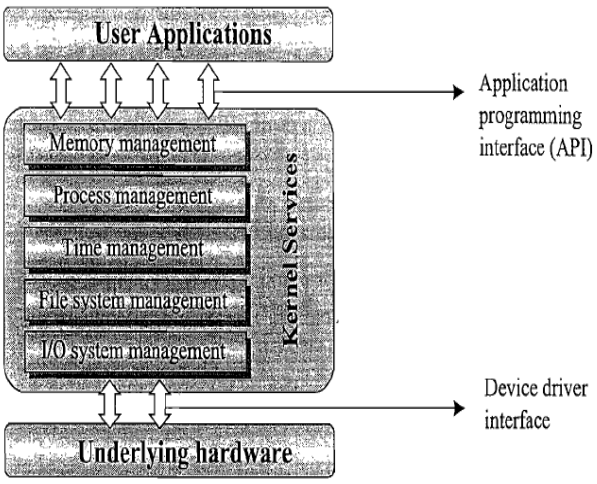
	Scheme and Solution	MARKS	CO	RBT
1	LED is a p-n junction diode and contains a CATHODE and ANODE For functioning the anode is connected to +ve end of power supply and cathode is connected to –ve end of power supply. The maximum current flowing through the LED is limited by connecting a RESISTOR in series between the power supply and LED as shown in the figure below  There are two ways to interface an LED to a microprocessor/microcontroller: 1.The Anode of LED is connected to the port pin and cathode to Ground : In this approach the port pin sources the current to the LED when it is at logic high(ie. 1) 2.The Cathode of LED is connected to the port pin and Anode to Vcc : In this approach the port pin sources the current to the LED when it is at logic high (ie. 1). Here the port pin sinks the current and the LED is turned ON when the port pin is at Logic low (ie. 0) A seven-segment display (SSD), or seven-segment indicator, is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot matrix displays.Seven-segment displays are widely used in digital clocks, electronic meters, basic calculators, and other electronic devices that display numerical information. The seven elements of the display can be lit in different combinations to represent the Arabic numerals. Often the seven segments are arranged in an oblique (slanted) arrangement, which aids readability. In most applications, the seven segments are of nearly uniform shape and size (usually elongated hexagons, though trapezoids and rectangles can also be used), though in the case of adding machines, the vertical segments are longer and more oddly shaped at the ends in an effort to further enhance readability.The numerals 6 and 9 may be represented by two different glyphs on seven-segment displays, with or without a 'tail'.[2][3] The numeral 7 also has two versions, with or without segment F.[4] The seven segments are arranged as a rectangle of two vertical segments on each side with one horizontal segment on the top, middle, and bottom. Additionally, the seventh segment bisects the rectangle horizontally. There are also fourteen-segment displays and sixteen-segment displays (for full alphanumerics); however, these have mostly been replaced by dot matrix displays. Twenty-two segment displays capable of displaying the full ASCII character set[5] were briefly available in the early 1980s, but did not prove popular.The segments of a 7-segment display are referred to by the letters A to G, where the optional decimal point (an "eighth segment", referred to as DP) is used for the display of non- integer numbers.	4+6	CO4	L3





2	I2C was originally developed in 1982 by Philips for various Philips chips. The original spec allowed for only 100kHz communications, and provided only for 7-bit addresses, limiting the number of devices on the bus to 112 (there are several reserved addresses, which will never be used for valid I2C addresses). In 1992, the first public specification was published, adding a 400kHz fast-mode as well as an expanded 10-bit address space. Much of the time (for instance, in the ATMega328 device on many Arduino-compatible boards) , device support for I2C ends at this point. There are three additional modes specified: fast-mode plus, at 1MHz; high-speed mode, at 3.4MHz; and ultra-fast mode, at 5MHz. Each I2C bus consists of two signals: SCL and SDA. SCL is the clock signal, and SDA is the data signal. The clock signal is always generated by the current bus master; some slave devices may force the clock low at times to delay the master sending more data (or to require more time to prepare data before the master attempts to clock it out). This is called “clock stretching” and is described on the protocol page. Unlike UART or SPI connections, the I2C bus drivers are “open drain”, meaning that they can pull the corresponding signal line low, but cannot drive it high. Thus, there can be no bus contention where one device is trying to drive the line high while another tries to pull it low, eliminating the potential for damage to the drivers or excessive power dissipation in the system. Each signal line has a pull-up resistor on it, to restore the signal to high when no device is asserting it low.  The sequence of operation for communicating with an I2C slave device is: 1.Master device pulls the clock line (SCL) of the bus to ‘HIGH’ 2.Master device pulls the data line (SDA) ‘LOW’, when the SCL line is at logic ‘HIGH’ (This is the ‘Start’ condition for data transfer) 3.Master sends the address (7 bit or 10 bit wide) of the ‘Slave’ device to which it wants to communicate, over the SDA line. Clock pulses are generated at the SCL line for synchronizing the bit reception by the slave device. The MSB of the data is always transmitted first. The data in the bus is valid during the ‘HIGH’ period of the clock signal 4.Master sends the Read or Write bit (Bit value = 1 Read Operation; Bit value = 0 Write Operation) according to the requirement 5.Master waits for the acknowledgement bit from the slave device whose address is sent on the bus along with the Read/Write operation command. Slave devices connected to the bus compares the address received with the address assigned to them	10	CO4	L3

	6.The Slave device with the address requested by the master device responds by sending an acknowledge bit (Bit value =1) over the SDA line 7.Upon receiving the acknowledge bit, master sends the 8bit data to the slave device over SDA line, if the requested operation is 'Write to device'. If the requested operation is 'Read from device', the slave device sends data to the master over the SDA line 8.Master waits for the acknowledgement bit from the device upon byte transfer complete for a write operation and sends an acknowledge bit to the slave device for a read operation 9.Master terminates the transfer by pulling the SDA line 'HIGH' when the clock line SCL is at logic 'HIGH' (Indicating the 'STOP' condition)			
3	**Reset circuit:** The Reset circuit is essential to ensure that the device is not operating at a voltage level where the device is not guaranteed to operate, during system power ON. The Reset signal brings the internal registers and the different hardware systems of the processor/controller to a known state and starts the firmware execution from the reset vector (Normally from vector address 0x0000 for conventional processors/controllers. The reset vector can be relocated to an address for processors/controllers supporting bootloader The reset signal can be either active high (The processor undergoes reset when the reset pin of the processor is at logic high) or active low (The processor undergoes reset when the reset pin of the processor is at logic low).  **Brownout Protection** Brown-out protection circuit prevents the processor/controller from unexpected program execution behavior when the supply voltage to the processor/controller falls below a specified voltage. The processor behavior may not be predictable if the supply voltage falls below the recommended operating voltage. It may lead to situations like data corruption A brown-out protection circuit holds the processor/controller in reset state, when the operating voltage falls below the threshold, until it rises above the threshold voltage Certain processors/controllers support built in brown-out protection circuit which monitors the supply voltage internally. If the processor/controller doesn't integrate a built-in brown-out protection circuit, the same can be implemented using external passive circuits or supervisor ICs  	10	CO4	L3

4	*Real-Time Operating System (RTOS) based Embedded System Design*  Fig. 10.1 The Operating System Architecture The Kernel •The kernel is the core of the operating system and is responsible for managing the system resources andthe communication among the hardware and other system services. •Kernel acts as the abstraction layerbetween system resources and user applications. •Kernel contains a set of system libraries and services. •For a general purpose OS, the kernel contains different services for handling the following. Process Management •Process management deals with managing the processes/tasks. •Process management includes setting up the memory space for the process, loading the process's code into the memoryspace, allocating system resources, scheduling and managing the execution of the process, setting upand managing the Process Control Block (PCB), Inter Process Communication and synchronisation, process termination/deletion, etc. Primary Memory Management •The term primary memory refers to the volatile memory (RAM) where processes are loaded and variables and shared data associated with each process are stored. •The Memory Management Unit (MMU) of the kernel is responsible for •Keeping track of which part of the memory area is currently used by which process •Allocating and De-allocating memory space on a need basis (Dynamic memory allocation). File System Management •File is a collection of related information. •A file could be a program (source code or executable), text files, image files, word documents, audio/video files, etc. •Each of these files differ in the kind of information they hold and the way in which the 1TFLinformation is stored. •The file operation is a useful service provided by the OS. •The file system management service of Kernel is responsible for •The creation, deletion and alteration of files □Creation, deletion and alteration of directories □Saving of files in the secondary storage memory (e.g. Hard disk storage) □Providing automatic allocation of file space based on the amount of free space available □Providing a flexible naming convention. for the files Secondary Storage Management •The secondary storage management deals with managing the secondary storage memory devices, if any, connected to the system. •Secondary memory is used as backup medium for programs and data since the main memory is volatile. •In most of the systems, the secondary storage is kept in disks (Hard Disk). •The secondary storage management service of kernel deals with □Disk storage allocation □Disk scheduling (Time interval at which the disk is activated to backup data) □Free Disk space management	[10]	CO5	L2

	Protection Systems •Most of the modern operating systems are designed in such a way to support multiple users with different levels of access permissions (e.g. Windows XP with user permissions like •'Administrator', 'Standard', 'Restricted', etc.). Protection deals with implementing the security policies to restrict the access to both user and system resources by different applications or processes or users. •In multiuser supported operating systems, one user may not be allowed to view or modify the whole/portions of another user's data or profile details. •In addition, some application may not be granted with permission to make use of some of the system resources. •This kind of protection is provided by the protection services running within the kernel. Interrupt Handler •Kernel provides handler mechanism for all external/internal interrupts generated by the system. •These are some of the important services offered by the kernel of an operating system. •It does not mean that a kernel contains no more than components/services explained above. •Depending on the type of the operating system, a kernel, may contain lesser number of components/services or more number of components/services. •In addition to the components/services listed above, many operating systems offer a number of add-on system components/services to the kernel. •Network communication, network management, user-interface graphics, timer services (delays, timeouts, etc.), error handler, database management, etc. are examples for such components/services. •Kernel exposes the interface to the various kernel applications/services, hosted by kernel, to the user applications through a set of standard Application Programming Interfaces (APIs). •User applications can avail these API calls to access the various kernel application/services.			
5	The Real-Time Kernel •The kernel of a Real-Time Operating System is referred as RealTime kernel. •In complement to the conventional OS kernel, the Real-Time kernel is highly specialised and it contains only the minimal set of services required for running the user applications/tasks. The basic functions of a Real-Time kernel are listed below: Task/Process management Task/Process scheduling Task/Process synchronisation Error/Exception handling Memory management Interrupt handling Time management Task/Process management •Deals with setting up the memory space for the tasks, loading the task's code into the memory space, allocating system resources, setting up a Task Control Block (TCB) for the task and task/process termination/deletion. •A Task Control Block (TCB) is used for holding the information correspondingto a task. •TCB usually contains the following set of information. Task ID: Task Identification Number Task State: The current state of the task (e.g. State = 'Ready' for a task which is ready to execute) Task Type: Task typeIndicates what is the type for this task. The task can be a hard real time or soft real time or background task. Task Priority: Task priority (e.g. Task priority = 1 for task with priority = 1) Task Context Pointer: Context pointer-Pointer for context saving Task Memory Pointers: Pointers to the code memory, data memory and stack memory for the task Task System Resource Pointers: Pointers to system resources (semaphores, mutex, etc.) used by the task Task Pointers: Pointers to other TCBs (TCBs for preceding, next and waiting tasks) Other Parameters Other relevant taskparameters •The parameters and implementation of the TCB is kernel dependent. •The TCB parameters vary across different kernels, based on the task management implementation. •Task management service utilises the TCB of a task in the following way	10	CO5	L2

	•Creates a TCB for a task on creating a task •Delete/remove the TCB of a task when the task is terminated or deleted •Reads the TCB to get the state of a task •Update the TCB with updated parameters on need basis (e.g. on a context switch) •Modify the TCB to change the priority of the task dynamically Task/Process Scheduling •Deals with sharing the PU among various tasks/processes. •A kernel application called 'Scheduler' handles the task scheduling. •Scheduler is nothing but an algorithm implementation, which performs the efficient and optimal scheduling of tasks to provide a deterministic behaviour. Task/Process Synchronisation •Deals with synchronising the concurrent access of a resource, which is shared across multiple tasks and the communication between various tasks. Error/Exception Handling •Deals with registering and handling the errors occurred/exceptions raised during the execution of tasks. •Insufficient memory, timeouts, deadlocks, deadline missing, bus error, divide by zero, unknown instruction execution, etc. are examples of errors/exceptions. Errors/Exceptions can happen at the kernel level services or at task level. •Deadlock is an example for kernel level exception, whereas timeout is an example for a task level exception. •The OS kernel gives the information about the error in the form of a system call (API). •GetLastError() API provided by Windows CE RTOS is an example for such a system call. •Watchdog timer is a mechanism for handling the timeouts for tasks. •Certain tasks may involve the waiting of external events from devices. •These tasks will wait infinitely when the external device is not responding and the task will generate a hang-up behaviour. •In order to avoid these types of scenarios, a proper timeout mechanism should be implemented. •A watch- dog is normally used in such situations. •The watchdog will be loaded with the maximum expected wait time for the event and if the event is not triggered within this wait time, the same is informed to the task and the task is timed out. •If the event happens before the timeout, the watchdog is resetted.			
6	**Mailbox** •Mailbox is an alternate form of 'Message queues' and it is used in certain Real- Time Operating Systems for IPC. •Mailbox technique for IPC in RTOS is usually used for one way messaging. •The task/thread which wants to send a message to other tasks/threads creates a mailbox for posting the messages. •The threads which are interested in receiving the messages posted to the mailbox by the mailbox creator thread can subscribe to the mailbox. •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 mailbox creation, subscription, message reading and writing are achieved through OS kernel provided API calls, Mailbox and message queues are same in functionality. •The only difference is in the number of messages supported by them. •Both of them are used for passing data in the form of message(s) from a task to another task(s). Mailbox is used for exchanging a single message between two tasks or between an Interrupt Service Routine (ISR) and a task. •Mailbox associates a pointer pointing to the mailbox and a wait list to hold the tasks waiting for a message to appear in the mailbox. •The implementation of mailbox is OS kernel dependent. •The MicroC/OS-II implements mailbox as a mechanism for inter-task communication. •Figure given below illustrates the mailbox based IPC technique	[5+5]	CO5	L2

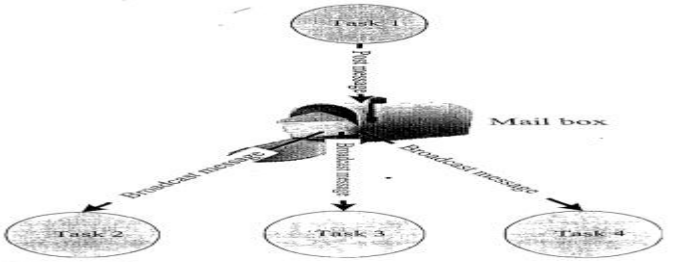


Fig. 10.21 Concept of Mailbox based indirect messaging for IPC

•Socket is a logical endpoint in a two-way communication link between two applications running on a network.
•A port number is associated with a socket so that the network layer of the communication channel can deliver the data to the designated application.
•Sockets are of different types, namely, Internet sockets (INET), UNIX sockets, etc.
•The INET socket works on internet communication protocol.
•TCP/IP, UDP, etc. are the communication protocols used by INET sockets. INET sockets are classified into:
1. Stream sockets
2. Datagram sockets
•Stream sockets are connection oriented and they use TCP to establish a reliable connection. On the other hand, Datagram sockets rely on UDP for establishing a connection.
•The UDP connection is unreliable when compared to TCP.
•The client-server communication model uses a socket at the client side and a socket at the server side.
•A port number is assigned to both of these sockets.
•The client and server should be aware of the port number associated with the socket.
•In order to start the communication, the client needs to send a connection request to the server at the specified port number.
•The client should be aware of the name of the server along with its port number.
•The server always listens to the specified port number on the network. Upon receiving a connection request from the client, based on the success of authentication, the server grants the connection request and a communication channel is established between the client and server. The client uses the host name and port number of server for sending re- quests and server uses the client's name and port number for sending responses.

7	A race condition produces incorrect results whereas a deadlock condition creates a situation where none of the processes are able to make any progress in their execution resulting in a set of deadlock processes. A situation similar to traffic jam issues is illustrated below	[1+4+5]	CO5	L3

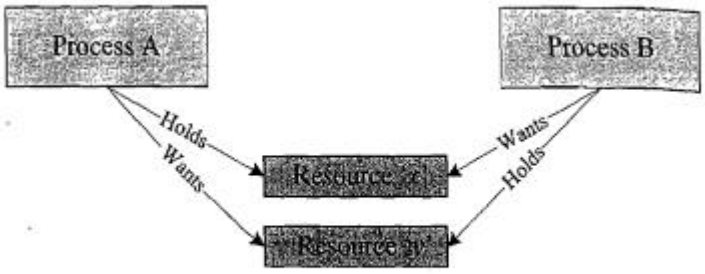


Fig. 10.25 Scenarios leading to deadlock

In its simplest form 'deadlock' is the condition in which a process is waiting for a resource held by another process which is wait- ing for a resource held by the first process (Fig. 10.25). To elaborate: Process A holds a resource x and it wants a resource y held by Process B. Process B is currently holding resource y and it wants the resource x which is currently held by Process A. Both hold the respective resources and they compete each other to get the resource held by

the respective processes. The result of the competition is 'deadlock'.
None of the competing process will be able to access the resources held by other processes since they are locked by the respective processes (If a mutual exclusion policy is implemented for shared resource access, the resource is locked by the process which is currently accessing it).
Mutual Exclusion: The criteria that only one process can hold a resource at a time. Meaning processes should access shared resources with mutual exclusion. Typical example is the accessing of display hardware in an embedded device.
Hold and Wait: The condition in which a process holds a shared resource by acquiring the lock control- ling the shared access and waiting for additional resources held by other processes.
No Resource Preemption: The criteria that operating system 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 a resource which is currently held by another process which in turn is waiting for a resource held by the first process.

Deadlock Handling
The OS may adopt any of the following techniques to detect and prevent deadlock conditions.
Ignore Deadlocks:
Always assume that the system design is deadlock free. This is acceptable for the reason the cost of removing a deadlock is large compared to the chance of happening a deadlock.
UNIX: is an example for an OS following this principle.
A life critical system cannot pretend that it is deadlock free for any reason.
Detect and Recover:
This approach suggests the detection of a deadlock situation and recovery from it. This one is similar to the deadlock condition that may arise at a traffic junction.
When the vehicles from different directions compete to cross the junction, deadlock (traffic jam) condition is resulted.deadlock (traffic jam) is happened at the junction, the only solution is to back up the vehicles from one direction and allow the vehicles from opposite direction to cross the junction. If the traffic is too high, lots of vehicles may have to be backed up to resolve the traffic jam. This technique is also known as **'back up cars' technique** (Fig. 10.26).

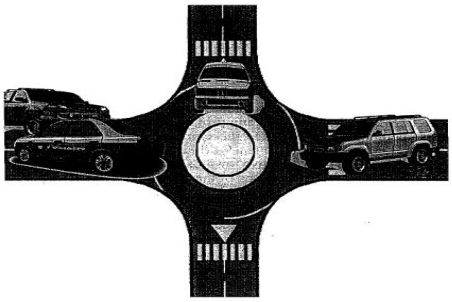


Fig. 10.26 *'Back up cars'* technique for deadlock recovery

Operating systems keep a resource graph in their memory. The resource graph is updated on each resource request and release. A deadlock condition can be detected by analysing the resource graph by graph analyser algorithms. Once a deadlock condition is detected, the system can terminate a process or preempt the resource to break the deadlocking cycle.
Avoid Deadlocks: Deadlock is avoided by the careful resource allocation techniques by the Operating System. It is similar to the traffic light mechanism at junctions to avoid the traffic jams.
Prevent Deadlocks: Prevent the deadlock condition by negating one of the four conditions favouring the deadlock situation.
Ensure that-a process does not hold any other resources when it requests a resource. This can be achieved by implementing the following set of rules/guidelines in allocating resources to pro- cesses.
1. A process must request all its required resource and the resources should be allocated before the process begins its execution.
2. Grant resource allocation requests from processes only if the process does not hold a resource currently.
Ensure that resource preemption (resource releasing) is possible at operating system level. This can be achieved by implementing the following set of rules/guidelines in resources allocation and releasing.

	1. Release all the resources currently held by a process if a request made by the process for a new resource is not able to fulfil immediately. 2. Add the resources which are preempted (released) to a resource list describing the resources which the process requires to complete its execution. 3. Reschedule the process for execution only when the process gets its old resources and the new resource which is requested by the process. Imposing these criterions may introduce negative impacts like low resource utilisation and starvation of processes. **Livelock** • The Livelock condition is similar to the deadlock condition except that a process in livelock condition changes its state with time. • While in deadlock a process enters in wait state for a resource and continues in that state forever without making any progress in the execution, in.alivelock condition a process always does something but is unable' to make any progress in the execution completion. • The livelock condition is better explained with the real world example, two people attempting to cross each other in a narrow corridor. • Both the persons move towards each side of the corridor to allow the opposite person to cross. Since the corridor is narrow, none of them are able to cross each other. Here both of the persons perform some action but still they are unable to achieve their target, cross each other. **Starvation** • In the multitasking context, starvation is the condition in which a process does not get the resources required to continue its execution for a long time. • As time progresses the process starves on resource. • Starvation may arise due to various conditions like byproduct of preventive measures of deadlock, scheduling policies favouring high priority tasks and tasks with shortest execution time, etc.			