

UNIT-V

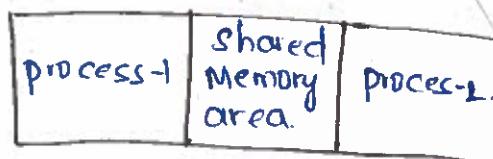
(1)

Task Communication.

Shared Memory:

The medium through which processes/tasks communicate with each other is known as inter process/task communication (IPC). The shared memory concept is kernel (operating system) dependent.

The information of the process to be communicated is returned by a process and ready by other process in shared memory area. The concept of shared memory is shown in fig(i).



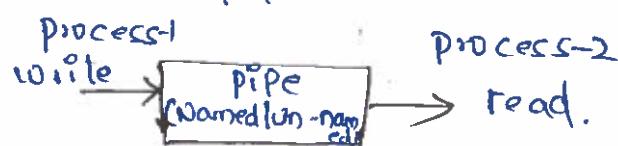
fig(i): concept of shared memory.

Eg: Notice board in which the information is published and read.

Pipes:

Pipe is an area of the shared memory which consists of two ends for the flow of information either in uni-directional (process at one end to write data and the process at the other end to read the data) or bi-directional (process can read and write at one end).

A unidirectional pipe is shown in fig(ii).



fig(ii) concept of pipe for ipc.

These pipes follow the 'client-server' architecture in which process which creates the pipe is called as 'pipe server' and the process which links to a pipe is called as 'pipe client'.

Pipes are operating system dependent and are classified as,

1. Anonymous pipes.

The pipes that are un-named and used for uni-directional transfer of data between two processes are referred to as anonymous pipes.

2. Named pipes.

The pipes that are named, and used for both unidirectional and bidirectional transfer of data between two processes are referred to as named pipes. Similar to anonymous pipes, the process creating a named pipe is called 'pipe server' and the process connecting to the named pipe is called 'pipe client'. In named pipes, process can appear as both client and server.

These pipes can be used for communication between processes that run on same machine or different machines.

Concept of memory mapped objects for IPC.

Memory mapped object is a type of shared memory technique used by various real-time operating systems. In this a shared block of memory is simultaneously accessed by multiple processes. When a mapping object is created physical storage is assigned to it which can be mapped to the virtual address space of process to route read and write operations and share data with other processes.

The concept of memory mapped object based shared memory technique for inter process communication is shown in fig.

- (2)
- 1. Save context info p-2
 - 2. Perform other OS operations if required
 - 3. Reload context info p-2 from PCB.

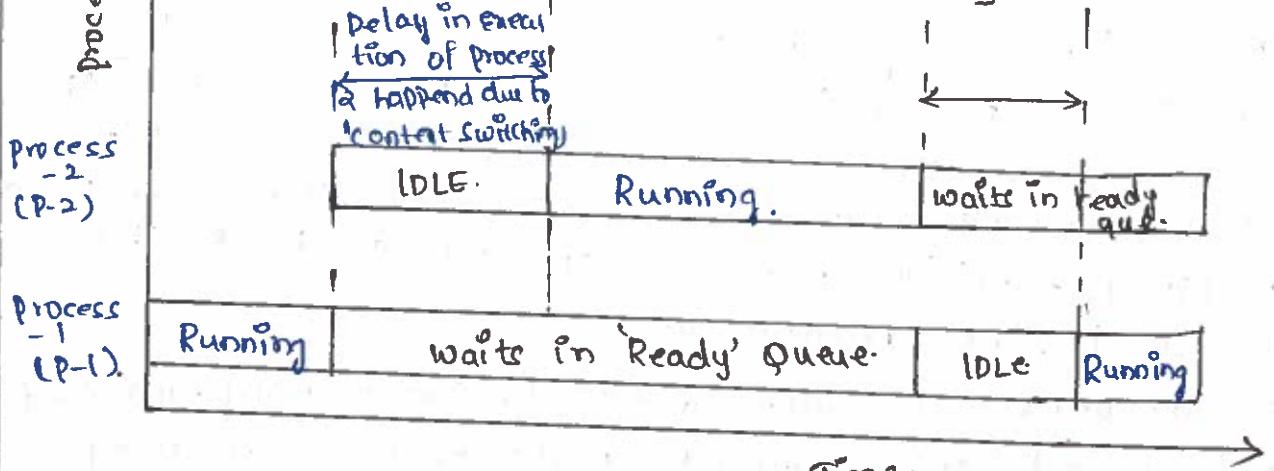


fig: concept of memory mapped object.

In order to share data between processes using memory mapping, various sequence of steps are required to be followed as shown below,

1. Call - the Create file mapping function to create the memory object, by using lpfile name parameter which passes a name for the memory-mapped object.
2. HANDLE Create file for mapping (LPCWSTR lpfilename,
3. DWORD dwDesiredAccess,
4. DWORD dwShareMode,
5. LPSECURITY_ATTRIBUTES lpSecurityAttributes ,
6. DWORD dwCreationDisposition,
7. DWORD dwFlagsAndAttributes ,
8. HANDLE hTemplate file .

In the above system call filename of the file to be read. The dwDesired Access parameter specifies the read write access permissions for the memory mapped object. It should either be zero or GENERIC-READ. The security attributes are required to NULL, whereas the hTemplate file parameter is signified by windows CE.

ALSO, a file which is open in GENERIC_WRITE mode can call the CreateFileForMapping function only once.

9. The process with which communication is to be performed, must pass the name of the memory-mapped Object.

10. call the CreateFileMapping function in the second process, by using the name of the object that has been passed for the first process. Since, the name of the memory-mapped object is global therefore, when the second process calls the CreateFileMapping function, Windows CE passes back the handle to the original object.

11. In order to gain access to the memory-mapped object, a function called MapViewOfFile is used in either of the two processes. Atlast for creating a view map, ViewOfFile function is called, which returns a pointer to the memory-mapped file.

Windows CE supports both named and unnamed file mapping objects. The unnamed or "page-file backed" memory-mapped files are neither backed by a page nor they are required to be unnamed when an unnamed object created and memory pointer is passed to several processes one process can close the unnamed object without informing the other process. Hence to avoid this memory error, windows CE supports naming of an unnamed object. The name of the object can be passed instead of a pointer so that the other process can access an object through its name. When both process close the object, windows CE deletes the object. Creating such a memory mapped file is achieved by eliminating the call to CreateFileForMapping and passing INVALID_HANDLE_VALUE in the handle field of CreateFileMapping. size of memory mapped region must be specified.

Message Passing.

(3)

Message passing is a synchronous/ asynchronous IPC mechanism which deals with exchanging of messages between processes. It quickly passes limited amount of data and is free from synchronization overhead depending on the operation of message passing between the process, it can be classified as follows.,

1. Mail Box.

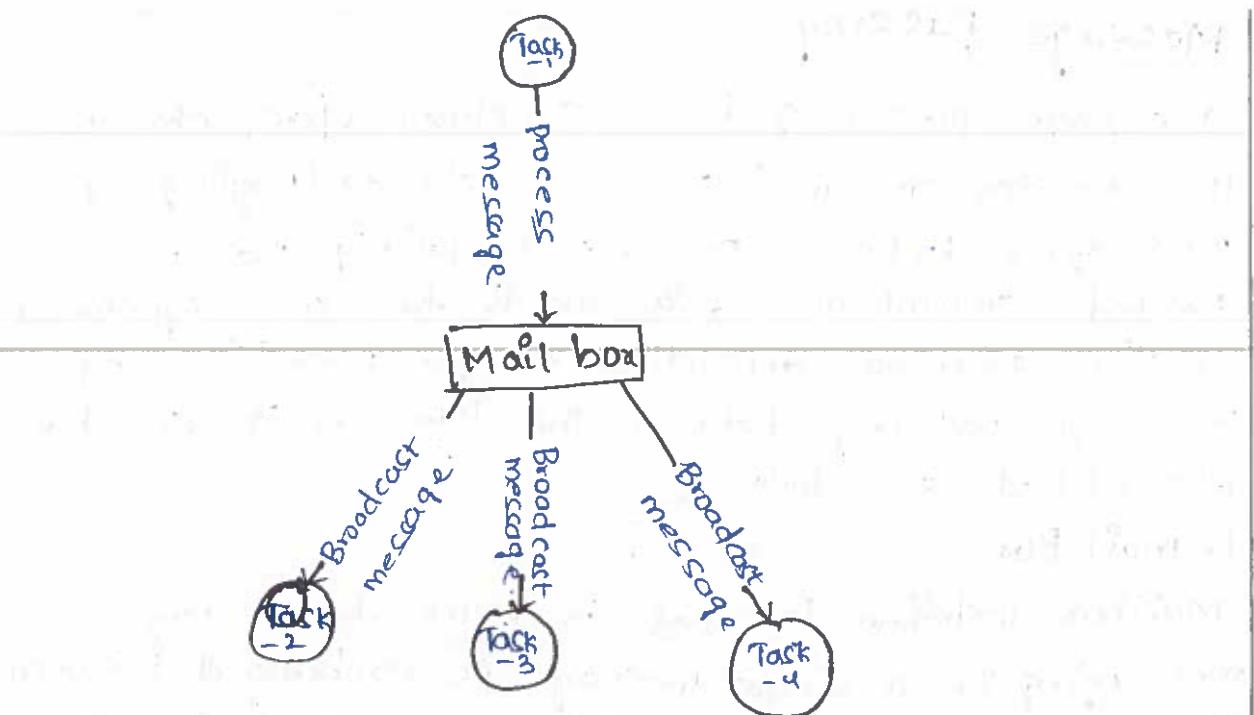
Mailbox technique is used in RTOS for one-way messaging i.e., a single message is exchanged between process. It quickly passes limited amount of data and is free) two tasks or between ISR and a task. When a task or thread posts a message to mailbox, the other thread reads the message from it. For this a mailbox must be created by message one task and subscribed by other task through os kernel provided API calls.

A task which creates mailbox is known as Mailbox Server, while task which subscribe the mailbox is known as mailbox client.

When mailbox server hosts messages to the mailbox, the subscribed client gets notified and reads the messages. Mailbox contains a pointer, pointing to mailbox and a wait list to hold the tasks.

Eg: Micro C by OS-II

The basic function of a mailbox can be depicted as shown in the following fig(1).

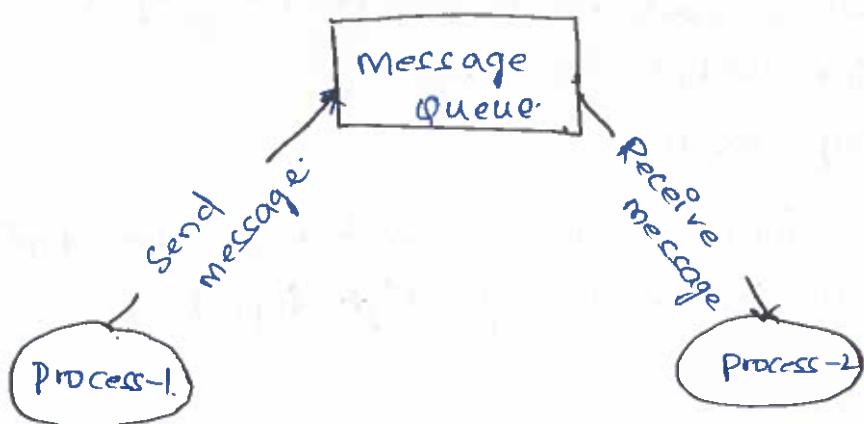


Fig(1).

a. Message Queues.

Message Queue is an inter task communication mechanism that supports two-way communication of messages between tasks or task and ISR. In every incoming message and outgoing message is allocated with the corresponding message queue.

It is similar to mailbox but the only difference is number of messages supported by them. In these the messages sending and receiving are OS kernel dependent and are shown in fig(2).



Fig(2)

The tasks that communicate with other tasks posts a message in FIFO queue called message queue. (4)

The kernel servers the messages one at a time by posting the message to the corresponding destination thread. Based on the behavior of message posting mechanism, the messaging mechanism is classified as,

Synchronous Messaging

In this, a thread that posts a message waits for message from a thread to which it is to be posted.

Asynchronous messaging

In this the message is posted in queue without waiting for a thread to which the message is to be posted.

3. Signaling.

Signaling is a primary communication between tasks or processes. It is handled by signal handler and implemented for inter process communication. Signaling provides signals of asynchronous notifications that doesn't carry data, for occurrence of scenario of other thread.

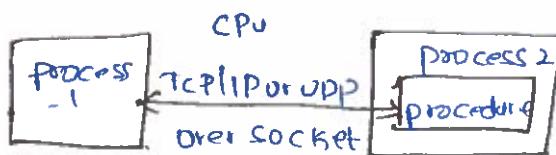
Remote procedure call.

Remote Procedure Call (RPC) is a mechanism in which a process in a system invokes a procedure on other or same system.

The remote procedure is shown in fig.



process running on different CPUs which are networked.



(fig1)

When a procedure on machine A invokes a procedure on machine B, then calling procedure on machine A is suspended and the called procedure on machine B is executed. Transfer of information from caller to callee takes place through parameters and the result can come back in the procedure result.

RPC is also referred as remote invocation or remote model invocation (RMI) in object oriented programming language. The CPU-process which initiates the request is referred as client, while the process in which the procedure that needs to be invoked remotely is referred as server.

RPC communication can be implemented on different invocation interfaces (Interface Definition Language IDL) and platforms Microsoft Interface Definition Language (MIDL) by following certain standard formats.

The communication in RPC is of two types:

1. Synchronous - RPC calls:

In this, the process blocks the remote procedure until it receives a response from other process.

2. Asynchronous - RPC calls:

In this, the calling process and remote process continue their execution of the procedure. callback functions are used to return the result from remote procedure.

RPC employs authentication mechanism like the IDS, public key cryptography etc. to protect the system from unauthorized access.

Sockets:

(5)

In RPC communications, sockets are referred as logical end points in a bidirectional communication link between two applications in a network. There are assigned with port numbers so that the network layer can deliver the data to the corresponding application. Socket implementation is OS kernel dependent.

Eg: Internet Sockets (INET), Unix Sockets.

INET Sockets are used in Internet communication protocol and are of two types,

i. Stream Sockets:

Stream Sockets are connection oriented, they establish reliable connection using TCP.

ii. Datagram Sockets:

It is also connection oriented. They establish a connection using UDP, which is unreliable.

In clients server communication model, client and server sites are provided with sockets having port numbers. A client sends a request to a known host name and port number of the server for the communication. On receiving the request from the client, the server using client's name and port number obeys the connection request. Hence the communication channel is established between client and server.

The host name and port number can be same for client and server running on same CPU. The communication link may be Ethernet, WiFi etc.

Task Synchronization.

The act of synchronising the access of shared resources by multiple process and enforcing proper sequence of operation among multiple process of a multitasking system is referred to as task / process synchronization.

Race condition.

A racing or race condition can be expressed as a situation where in multiple processes compete with each other in order to access the shared resources concurrently. The resultant value of the shared resource or data depends on the finally used process.

A simple example for 'race condition' is shown in figure,

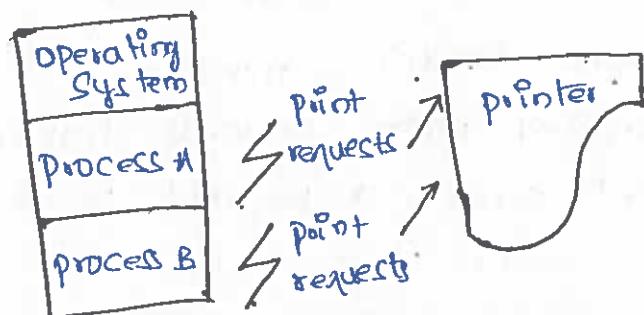


Fig: Race condition

Consider two processes, process A and process B running on an operating system. Suppose, process 'A' requests the printer to print a line and on the same time process B also requests for the print. As there is a single printer available the point out of process A may either precede or follow the print out of process B. This situation leads to race condition and the resultant print out will have each process interpersed on the printed paper.

Before using the shared resource (printer), if a process explicitly requests for its use, the race condition problem can be solved. When process makes use of a resource completely, it releases the resource. The 'request' and 'release' operations are facilitated by operating system and are handled by the traffic controller. If a process requests for a resource which is already in use, the process is blocked automatically and when this resource is released by the process using it, it is again made available to the blocked process.

Task communication issues.

The different task communication synchronization issues encountered in inter process communication are discussed below.

1. Race condition.

A racing or race condition can be expressed as a situation wherein multiple processes compete with each other in order to access the shared resources concurrently. The resultant value of the shared resource or data depends on the finally used process.

2. Deadlock.

Deadlock refers to situation in which one process is waiting for a resource which is currently under the control of some other process. Hence, it results in the permanent blocking of the process.

3. Dining philosopher's problem.

The 'dining philosopher's problem' is the most suitable example for explaining the synchronization issues in resource utilization. It is an analogy of process which compete for shared resources that result in racing, dead lock, starvation and live lock.

4. producer-consumer problem.

The problem in which any two process concurrently access a shared buffer with fixed size is referred as producer-consumer/bounded buffer problem. In this, a data producing thread/process is called as producer thread/process, while the data consuming thread/process is called as consumer thread/process.

5. Reader-writer's problem.

This problem is observed in process that compete for shared resources i.e., trying to read and write shared data concurrently.

6. priority inversion.

The priority inversion problem arises when a resource is shared by two or more tasks. In this case, a situation arises where a higher priority task has to wait till a lower priority task is executed. medium priority task continues its execution by pre-empting low priority task as it doesn't require shared memory. Priority Inversion is a combination of blocking based process synchronization (deals with access of shared memory concurrently) and pre-emptive priority scheduling (deals with high priority tasks execution first).

7. priority Inheritance.

Priority Inheritance is a type of mechanism used for handling priority inversion problem. It is a lock based process synchronization technique in which a shared resource which is currently in use by a process, cannot be accessed by another process. In this, the priority of a low priority task which is currently holding a resource requested by a high priority task, Inheriting the priority temporarily. Raising the priority of a low priority task to the priority of task requesting shared resource held by low priority.

-task, eliminates the pre-emption (release) of a low priority task by other tasks. Hence the delay encountered in waiting for the resource requested by the high priority task is reduced. When the low priority task releases the shared resource its boosted priority is brought back to the original value.

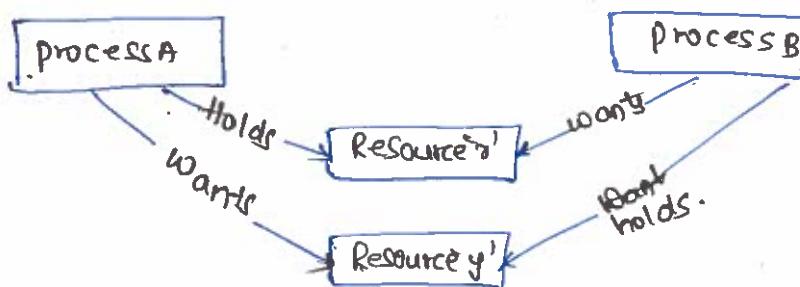
8. Priority ceiling.

Priority ceiling is a mechanism used for handling priority inversions. It is a technique in which a priority is associated with a shared resource. The priority of the highest priority task which utilises the shared resource is referred as ceiling priority. The priority of the task is temporarily boosted to the priority of the shared resource when the resource is being held by the task. This prevents the pre-emption of a task by other medium priority tasks which results in priority inversion. Once the task completes its access to the shared resource, its priority is brought back to the original level.

Different conditions causing deadlock.

Dead lock refers to situation in which one process is waiting for a resource which is currently under the control of some other process. Hence, it results in the permanent blocking of the processes.

The scenario which leads to dead lock is shown in fig. below.



In the above figure, process A is holding resource X and wants resource Y and process B is holding resource Y and wants resource X. Hence these two processes compete with each other to get the resources held by the other processes. A process cannot access the resource held by other process as they are locked, this results in dead lock.

Conditions favouring Deadlock.

A deadlock can occur if the following four conditions hold simultaneously in a system,

1. Mutual Exclusion.

It refers to the use of a resource by only one process at any time. If another process requests that resource, the requesting process must be delayed till the resource is released.

2. Hold and Wait

There must exist at least one resource that is held by a process and the process is waiting to acquire additional resources that are currently held by other processes.

3. No preemption.

Resources can't be preempted, that is, a resource can be released voluntarily by the process holding it. After that, process has completed its task.

4. Circular wait.

There exists a list of waiting processes ($P_0, P_1 \dots P_n$) such that process P_0 is waiting for resource currently under the usage by process P_1 , P_1 is waiting for a resource that is held by P_2 , P_2 is waiting for a resource that is held by P_3 and so on. Finally, a process P_n is waiting for the resource held by P_0 .

the first three conditions result in a sequence of events that finally leads to an unresolvable circular wait which is actually the main cause for the occurrence of deadlock.

(8)

A good operating system predicts a deadlock situation before its occurrence and acts accordingly to prevent it. The response of an operating system to a deadlock condition is inconsistent - therefore it adopt various techniques to detect and prevent deadlock condition.

Different methods of handling deadlocks.

The different methods of handling deadlocks are as follows,

1. Ignore deadlock.

In this method, deadlocks are ignored by simply assuming that the designed system is deadlock free. By doing so expenses are minimized because the cost of removing a deadlock is more when compared to the chance of deadlock occurrence. In real there is no deadlock free system.

2. Detect and Recover Deadlocks.

In this approach, the deadlock situation is first detected and then recovered. This is analogous to the deadlock condition that crops-up at a traffic junction. The vehicles coming from different directions contend to cross the junction resulting in deadlock (traffic jam). Once traffic jam occurs, the only solution at hand to cross the junction is to back up the vehicles from one direction and allow the vehicles from opposite direction. This technique is known as 'back up cars' technique and is shown

The operating systems are equipped with resource graph are started in their memory which gets updated regularly on each request and release of a resource. These resource graphs are studied by graph analyzer algorithms for detecting the deadlock conditions. When a deadlock occurs the system stops the process or resource to avoid dead locking cycle.

3. Avoid Deadlocks.

An operating system can avoid deadlock by using techniques that distribute the resources carefully.

4. Prevent Deadlocks.

Deadlocks can be prevented by dispriring any of the following four conditions.

(i) When a process requests a resource, make sure that it does not hold any other resources. This can be detected from the following set of guidelines.

(a) A process requests a resource, make sure that it [does not hold any other resource].

(b) A process ~~test~~ should request and be allocated with all the resources required before it starts execution.

(c) The process (which doesn't hold any resource) requesting per resource allocation is granted.

(ii) Make sure that resource releasing can be carried out at operating system level. This can be detected from the following set of rules,

(a) When a request for a new resource made by the process is left unfulfilled, all the resources that a process currently holds must be released.

(b) The resources which are preempted must be added to the resources list, that describes the resources required by a process for completing

(c) when a process attains both the old resources and new resources then it must be rescheduled for execution.

9

Practising these conditions may lead to negative effective like low resource utilisation (Livelock) and starvation of processes.

* Livelock:-

A condition in which a process always does something but is unable to make any progress in the completion of execution is referred as 'livelock'. This is clearly explained by using 'The dining philosophers' problem.

* Starvation.

The condition in which a process does not get the CPU or system resources that are required to continue its execution for a long duration is referred to as 'starvation'. This mainly occurs due to the bi product of deadlock prevention, scheduled policies favouring high priority tasks etc.

Starvation in the process scheduling context.

The condition in which a process does not get the CPU or system resources that are required to continue its execution for a long duration is referred to as 'starvation'. This mainly occurs due to the bi product of deadlock prevention, scheduled policies favouring high priority tasks etc.

Elimination of Starvation.

The problem of starvation arise with the way adopted by the total bandwidth server algorithm in providing the back-ground time to the TB₁. In particular the deadline of the back-logged total bandwidth server is kept arbitrary. There are many schemes to eliminate the starvation and enhance the fairness one among it is

keep the deadline of backlogged server in vicinity of the current time.

The problem can be minimized by using constant utilization servers. The budget of the server is not replenished prior to current server deadline. It implies backlogged constant utilization server deadline ϵ_0 , having size m is less than ϵ_1 , margin of time measured from the current time and the length of starvation in any server to by $\max(\epsilon_1, m\Delta t)$. The background time can be utilized by the server by adding rule R3 to the constant utilization server budget replenishment rules.

Replenishment Rules of a starvation-free constant utilization/Background server.

R3 use the rules of constant utilization server in order to replenish the budget of every backlogged server in any busy interval system systems.

Replenish the budget of every backlogged server each time in a busy interval terminates.

process Synchronisation in IPC & Mutual Exclusion in context of Synchronisation.

Role of process synchronisation in IPC.

Process synchronization plays a crucial role in interprocess communication. Some of its functions are mentioned below.

It is used to prevent the conflicts (race, deadlock, starvation, livelock etc) that may arise during resource access in a multitasking environment.

proper sequence of operations across the process is ensured using process synchronisation. A typical example for this is a 'producer-consumer problem' which requires appropriate sequencing of its operations. In this problem the access to shared buffer by different processes is

easy whereas the condition that writing process waiting to a shared buffer only when it is empty and consumer thread reading from the buffer only when it is full is complex to implement. Therefore, a proper synchronisation is required to execute sequence of operations.

Through proper synchronisation communication between processes could be made effective.

(10)

Mutual Exclusion in the context of process synchronization.

The act of preventing a task/process from access to a shared resource which is already held by another task/process is referred to as mutual exclusion. It provides a critical section which is a code memory area containing the program instructions or accessing a shared resource. Mutual exclusion mechanism supports a access to critical section of code by synchronizing the access to shared resources.

For example, consider two processes, Process 1 and process 2 which runs on a multi-tasking systems. Process 1 is currently running and enters its critical section. Before completion of operation of process, it gets preempted and schedules processes 2 for execution because the priority of process 2 is high when compared to process 1. Thus process 2 also gains access to the critical section used by process 1 and continues its execution. This result in racing condition. Mutual exclusion is used to block a processes mutually accessing critical section, mutual exclusion policy can be implemented either through busy waiting (spin lock) or keep and wake-up technique.

Test and Set Lock Based Mutual Exclusion Technique.

The test and set lock (TSL) based mutual exclusion technique combines the three different operations i.e., reading, testing and setting into a single step.

The main function of TSL is that it copies the value of lock variable and sets it to a non-zero value. i.e., the process/thread tests the lock variable to see whether its state is '0' and sets its state to '1', if its state is '0' for acquiring the lock.

The TSL instruction is supported by intel 486 and above family of processor with a special instruction CMPXCHQ (Compare and Exchange),

The syntax of compare and exchange instruction is,

CMPXCHQ dest,src

Here, 'dest' represents the destination which can be register or memory location and 'src' represents the source which is always a register. The instruction CMPXCHQ compares the accumulator (EAX register) with 'dest'. If the contents of both are equal, then source is loaded in the destination. If the contents are not equal, then 'dest' is loaded in accumulator.

Ex:

if (accumulator == destination)

{

ZF = 1; // Set the zero flag of EFLAGS register

destination = source;

else

{

ZF = 0; // Reset the zero flag of EFLAGS register

accumulator = destination;

}

3.

At the H&B processor level, the accumulator is loaded with '0' and a general purpose register with '1'. The memory location which holds the lock variable is then compared with accumulator with the help of CMPXCHG instruction.

(10)

Mutex Based process Synchronisation Under windows OS

Mutex is an abbreviation of mutual exclusion and it is used for task and resource synchronisation.

→ The mutual exclusion can be carried by the functions like disabling the scheduler and interrupts. It checks the presence of the resource by setting and testing the value of a global variable, after disabling the interrupts.

→ mutex is a special binary semaphore.

→ A task obtains a mutex for using a resource and releases it once the usage is finished. → mutex can be in either blocked or unblocked state.

→ A mutex can be handled only by the owner task and not by other tasks.

→ It can be obtained by any number of times by the task owning it as long as the mutex is in locked state.

→ At the same time, the task owning it, has to release it as many times as it has acquired the at the mutex. Once a task obtains a mutex that particular task cannot be deleted.

When process/thread can create a mutex object and other processes/threads of the system can use this mutex object for synchronization of the access to critical section.

→ When process/thread does not own the mutex object it sets its state to signalled and when it is owned, it is set to non-signalled.

→ The real time applications of mutex concept is described using hotel accommodation system.

Eg:
consider an example of hotel accommodation system

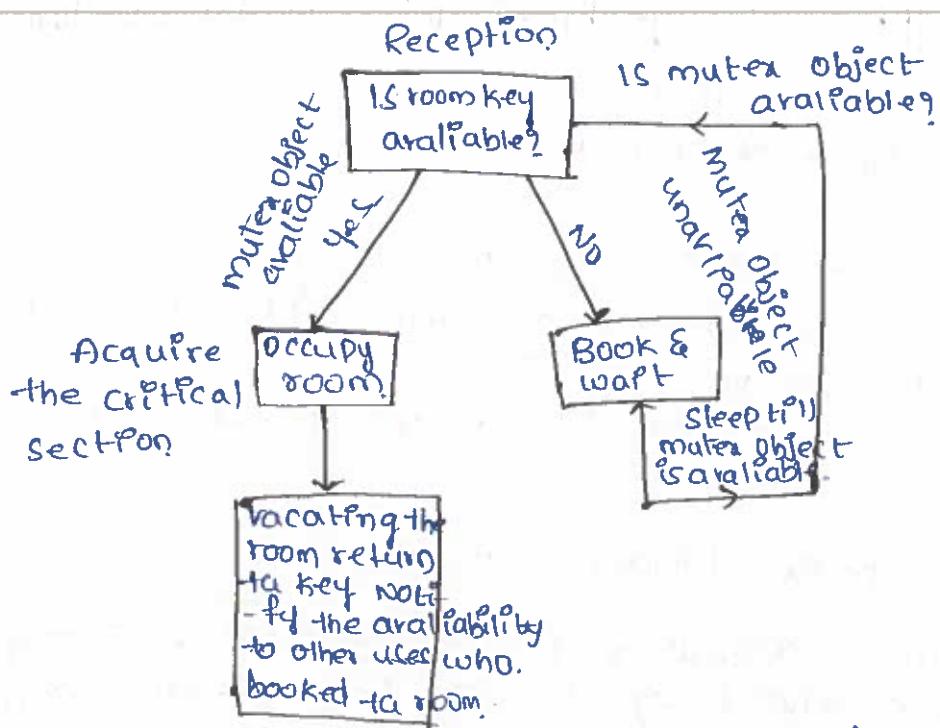


fig : concept of Binary Semaphore.

→ A customer wants to avail a room in a hotel and enquires about its availability. If a room is available, the key is handed over to the customer and he/she occupies the room.

→ If a room is unavailable, the customer has to book and wait till it is available. The customer who occupied the room, before vacating, returns the key and modify the availability to other users who booked the room thereby the customer who was waiting wakes up and occupy the vacant room.

In terms of binary semaphore concept, key is the mutex object, book and wait is the 'Sleep' process, room is the 'critical section'.

Device Drivers.

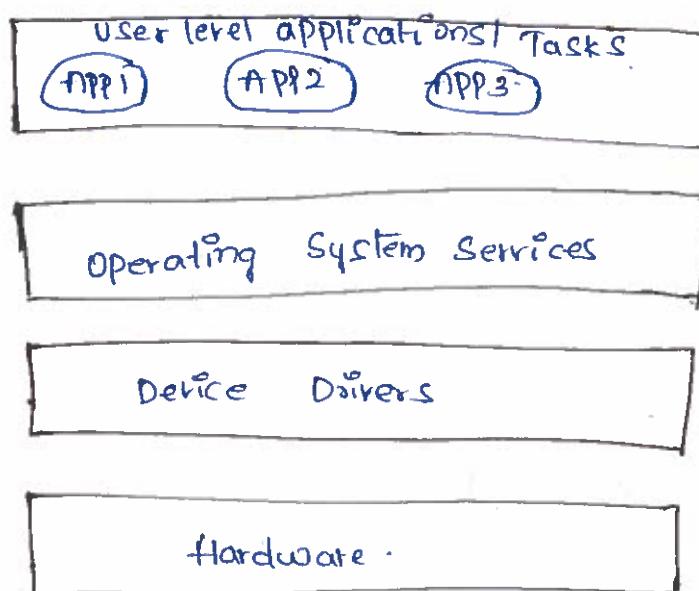
A device driver is a part of software that builds a connection between the operating system and the hardware.

Role of Device Drivers.

(12)

In an OS based product architecture, the device related access is done through OS kernel in the form of API which routes to the respective hardware peripheral.

The role of the device driver in an RTOS based system is shown in figure below.



Device driver initiates and manages the communication with hardware peripherals. The initialization of embedded products (Wi-Fi module, file systems) and protocols for communication with device driver is different. Each hardware requires separate device driver component.

Device drivers which are available in OS are called built-in drivers or on-board drivers, whereas the device drivers which are installed to access the device are known as installed drivers. In this, the drivers are loaded and unloaded from memory of OS based on connection and disconnection.

Drivers which run in user space are known as user mode drivers and which run on kernel space are known as kernel mode drivers.

- Device drivers support device initialization/configuring different registers of device), interrupt configuration (configuring interrupts associated with hardware), interrupt handling and processing (handling an interrupt by using ISR and then processing by invoking interrupt service thread (IST)) and client interfacing (implementation of inter process communication for communicating an synchronising with user application and drivers).

Applications.

Due to the wide range varieties of modern hardware and operating systems, device drivers can be interfaced with,

1. Printers.
2. Image scanners.
3. Digital cameras
4. Video adapters
5. Network and sound cards.
6. Bus mastering on modern systems.
7. Low bandwidth I/O buses of different kind such as mice, Keyboard and USB.
8. Computer storage devices like hard disk, CD-Rom and floppy disk.

Serial port Driver.

A serial port device driver drives the serial port of a PC. The driver employs a UART 8250 or UART 16550 IC in a PC along with 8086 processor to control the data transmission and reception. The UART comprises registers of control, data and status which are illustrated in the following example.

Development of a software for a driver in assembly also necessitates the availability of the following information,

1. Address for All the Available Registers

The addresses for a physical device are allocated using the hardware of the device and its interfacing circuit. This makes a device 'the owner' of the specific addresses.

(B)

For instance, the fixed addresses, allocated to the IBM PC hardware are,

(i) Timer.

It is addressed between 0x00H0-5F.

(ii) Keyboard

Keyboard is addressed between 0x00600 - 6FD.

(iii) Real-time clock (System clock).

It is addressed between 0x0070-7F.

(iv) Serial - Port COM port-1

This port is addressed between 0x03F8-3FF

(v) Serial com port-2

It is addressed between 0x02F8-2F.

2. A common address can have both input buffer and output buffer register. This is due to the availability of two different signals RD and WR at the control bus, during the read and write instructions. Depending on the action to be carried out, the physical device selects the suitable registers (i.e., either input buffer or output buffer register).

For instance, the register SBUF at 8051 micro-controller address both input-serial buffer and output-serial buffer.

3. A same address is assigned to multiple register of a physical device.

For instance, in a PC com2 serial device, the two registers namely RBR and TRL (i.e., receiver data buffer register and transmitter holding register) are allocated with the same address of ~~0x00~~ 0x2F8.

4. The purpose of each control register bit in a control register and each status flag in a status register should be known.

5. The addresses of the control bits ~~or~~ and status flags must be known. If these two share a common address, the processor performs two different operations.

(a) During read instructions, the processor reads the status from the address.

(b) During write instructions, the processor writes the control bits to that specific address.

6. The co-existence of the control bits and flags in the same register must be known.

7. The auto-reset of the status flag upon the execution of ISR should be known.

8. Before returning into the interrupted process, the change in control bits (such as set or reset) is to be considered.

9. Information regarding the list of actions to be carried out by the driver at the different registers (such as data buffers, control registers and status registers) should be known.

How To choose an RTOS.

The important functional and non-functional requirements that needs to be analysed in the selection of an RTOS for an embedded design are describe as follows.

1. Functional Requirements

(i) Processor Support

The selection of an RTOS should be analysed by the processor. All RTOS may not support all kinds of processor architecture. Therefore, before choosing an RTOS, it is important to check whether the RTOS supports the processor.

(ii) Memory Requirements

The operating system (OS) requires two types of memory

(a) ROM memory

ROM memory is required for holding the OS files and is stored in FLASH memory which is non-volatile.

14

(b) RAM memory

RAM memory is required for loading the OS services.

Since Embedded systems are memory constrained, it is important to analyze the minimal ROM and RAM requirement for the RTOS.

(iii) Real-time capabilities.

All Embedded operating systems may or may not be 'Real-Time' in nature. In the 'Real-time' behaviours of an OS, the task/process scheduling policies play an important role to analyse and meet the standards of OS.

(iv) Kernel and Interrupt Latency.

The kernel, which is the core element of OS may deactivate the interrupts while executing certain services. This may cause interrupt latency which should be minimum for an embedded system whose response requirements are high.

(v) Inter-process communication and Task Synchronisation.

This requirement is dependent on the OS kernel. Few kernels may provide a set of options whereas others provide very limited options. In order to avoid priority inversion issues in resource sharing policies are implemented by certain kernels.

(vi) Modularisation support

Majority of the OS provide a set of features which may not be necessary for functioning of certain embedded products.

Therefore, it is essential for the OS to support modularisation where in the developer can select the important modules and re-compile the OS image required for its functioning.

9.

(vii) Networking and Communication Support -

For a set of communication interfaces and networking, the OS kernel supporting all interfaces of embedded product provides stack implementation and driver support.

(viii) Development Language Support -

The OS may include run time libraries to run built-in libraries like JRM, .NET Compact framework (.NETCF) etc. If they are unavailable, then the third party vendor is checked for the OS under consideration.

a. Non-functional Requirements

(i) Off the Shelf or custom developed.

In the selection of an RTOS, the possible options could be complete development of OS satisfying the needs of an embedded system or the use of an off-the-shelf operating system, which is readily available. This off-shelf OS could be either a commercial product or open source product which closely resembles that system requirements. By customizing an open source OS, the required features can be obtained. The selection between these two is entirely dependent on the cost of development, licensing fees for the OS, development time and availability of skilled resources.

(ii) COST

Before selecting the OS, the following requirements should be evaluated in terms of cost -

(a) Total cost for developing or buying the OS.

(b) maintaining - the cost in terms of commercial product.

(iii) Availability of Development and Debugging tools.

This requirement plays an important role in the selection of an OS for embedded design. The availability of tools for supporting the development may be limited for an operating system whose performance is superior than other.

(15)

(iv) Ease of ~~use~~ Use.

This is another important feature for selecting an RTOS which depends on easy operation of commercial RTOS should be easy to operate.

(v) After Sales.

In commercial embedded RTOS, the services offered after sale for bug fixes, critical patch updates and support ~~for~~ for production issues through e-mail, customer care services etc., should be analyzed completely:

about 1000 feet above the sea level.

The "Giant" is a very large tree, about 100 feet high.

The trunk is very thick, about 10 feet in diameter.

The bark is smooth and greyish brown.

The leaves are large, about 10 inches long and 5 inches wide.

The flowers are small, yellowish green, and are produced in clusters.

The fruit is a small, round, yellowish orange, about 1 inch in diameter.

The tree is found in the forests of South America, particularly in Brazil and Argentina.

The wood is very hard and durable, and is used for making furniture, boats, and houses.

The bark is used for making dyes and tanning leather.

The leaves are used for making tea and medicine.

The flowers are used for making perfume and medicine.

The fruit is eaten raw or cooked, and is very sweet and juicy.

The tree is also used for making charcoal and firewood.

The bark is used for making dyes and tanning leather.

The leaves are used for making tea and medicine.

The flowers are used for making perfume and medicine.

The fruit is eaten raw or cooked, and is very sweet and juicy.

The tree is also used for making charcoal and firewood.

The bark is used for making dyes and tanning leather.

The leaves are used for making tea and medicine.

The flowers are used for making perfume and medicine.

The fruit is eaten raw or cooked, and is very sweet and juicy.

The tree is also used for making charcoal and firewood.

The bark is used for making dyes and tanning leather.

The leaves are used for making tea and medicine.

The flowers are used for making perfume and medicine.

The fruit is eaten raw or cooked, and is very sweet and juicy.

The tree is also used for making charcoal and firewood.

The bark is used for making dyes and tanning leather.

The leaves are used for making tea and medicine.

The flowers are used for making perfume and medicine.

The fruit is eaten raw or cooked, and is very sweet and juicy.