RTOS Basic

Basic concepts of RTOS:

Why RTOS:

Multitasking:

- Allows you to break a complex problem into simpler pieces so,Focus on the development of each task rather than building a scheduler, Increase the Efficiency of usage the CPU

Services:

- Kernel services are provided for resource management, memory management, event handling, messaging, interrupt handling , and more.

Structure:

Structure design of your application,

Portability:

Your application will run on any platform the RTOS runs on. So you can build an application for multiple targets from the same codebase.

Security:

Some RTOS offers some security features, like encryption support for various communication protocols, memory protection unit (MPU) support.

Debugging :

Some IDEs (such as IAR Embedded Workbench) have plugins that show nice live data about your running process such as task CPU utilization and stack utilization

Support:

Some RTOSs provide you with technical support and give you access to the expertise of the engineers who developed each module, and they can advise and support you on your project.

Why not RTOS:

Resources:

RTOS will require extra resources, Processing overhead due to context switch complex algorithms, memory usage overhead due to OS source size.

Determinism:

RTOS decreases Determinism.

Design:

Needs very careful design, and more developing skills. It is never easy to port an RTOS

Debugging:

Complex debugging (due to race conditions on resources shared among tasks)

Cost:

Lots of RTOS needs expensive licenses, so product cost will increase.

Real Time Systems

•A real-time system is one that must process information and produce a response within a specified time, else risk severe consequences, including failure. That is, in a system with a real-time constraint it is no good to have the correct action or the correct answer after a certain deadline: it is either by the deadline or it is useless.

We can also say that any information processing activity or system which has to respond to externally generated input stimuli within a finite and specified period.

Hard real-time:

• A system that can meet the desired deadlines at all times (100 percent), even under a worst-case system load. In hard real-time systems, missing a deadline, even a single time, can have fatal consequences.

• Hard real-time systems are used in cases where a particular task, usually involving life safety issues, needs to be performed within a particular time frame, otherwise a catastrophic event will occur.

Soft real-time:

• A system that can meet the desired deadlines on average.

• a soft real-time system will give reduced average latency but not a guaranteed maximum response time.

• soft RTOS can miss a few deadlines (such as dropping a few frames in a video application) without failing the overall system

Task:

Task also called A thread, It's the basic block of an application written under RTOS, It's a simple program with an infinite loop. Task has its own stack, CPU registers, and priority.

```
void vATaskFunction( void *pvParameters )
{
for( ;; )
{ -- Task application code here. -- }
}
```

Multitasking:

Is the process of switching the CPU between several tasks.This maximizes the utilization of the CPU, Provide modular construction for our application, makes the application design is easier.

Atomic: *An operation is said to be atomic if it can be completed without interruption.*

Polling vs. Interrupt driven events

Polling	***Interrupt***
• We check an event through an infinite loop. • checks all devices in a round robin fashion. • The main drawback of this method is the application needs to wait and check whether the new information has arrived, so it is a waste of time for the processor. •Also It may miss some events.	• An external or internal event that interrupts the processor to inform it that a device needs its service. • When an event happens the processor jumps to the Interrupt Service routine(ISR). • ISR is a function executed once the related interrupt happens.

Resource:

A Resource is an entity used by the task, it can be:

• I/O device, • Printer, • Keyboard, • Display, • Variable, • Array, • Structure, • File.

Shared Resource:

-A shared Resource, is a resource that can be used by more than one task. Each task should have exclusive access to the shared resource to prevent data corruption. There are techniques to ensure exclusive access to resources like mutual exclusion.

Critical Section of Code:

• Also called critical region. Is a section of code that shouldn't be interrupted. Most RTOS Systems enable us to disable the interrupt before this section then enable it again after that.

RTOS Kernel:

Kernel is the core component of any OS,

kernels components:

Scheduler is a set of algorithms that determines which task executes when.

Objects are special kernel constructs that help developers create applications for real-time embedded systems.

Services are operations that the kernel performs on an object

Priority Based Kernels:

Are Kernels that decide which task will run regarding it's priority. Most real time kernels are priority based.

Each task takes a priority based on it's importance. The Highest priority task is always ready to run.

There are two types of priority based kernels: Non-Preemptive Kernel & Preemptive Kernel.

RTOS Task Status:

Ready - Running - Blocked

Ready *===>* *Task has the highest priority* *===>* ***Running***

Ready *<===* *Task no longer has the highest priority* *<===* ***Running***

Running *===>* *Task is blocked due to a request for an unavailable resource* *===>* ***Blocked***

Running <=== *Task is unblocked and has the highest priority* ***<=== Blocked.***

Blocked ===> *Task is unblocked and has not the highest priority task* ***===> Ready.***

Non-Preemptive Kernel:

• An ISR Makes higher priority tasks ready to run. The ISR always returns to the interrupted task.

• The new higher priority task will run when the current task gives-up the CPU.

• This minimizes the data corruption risks in multitasking as each task finishes before the other begins.

• Task response time will equal to the time of the longest task.

• *Very few RTOS kernels now are using non preemptive kernels.*

Preemptive Kernel:

• *Here the highest priority task ready to run always gives the control to the CPU.*
• *ISR makes the highest priority task run. • The kernel next runs the highest priority task in the ready queue.*
• *Here the response time to the highest priority task is it's best.*
• *Corruption of data may happen for non protected shared resources.*

RTOS Scheduler:

• *Scheduler the core Component of any RTOS kernel, Its a set of algorithms that determines which task executes when.*
• *It keeps track of the status of each task, and decides which to run.*
• *In Most RTOSs the developer is the one who sets the priority of each task, regarding to this priority the scheduler will decide which task will run.*
• *The scheduler assumes that you knew what you were doing while setting tasks priority.*
• *A bad design for tasks priority, may lead to a high priority task hogs the processor for a long time, this is called* ***CPU starvation.***
• *It keeps track of the status of each task, and decides which to run.*
• *Scheduler has no control on tasks on the blocked status.If tasks are blocked the scheduler waits an event to unblock this tasks, like an external interrupt from pushing a button. If no events happened, surely it's a bad design from your side.*

If two tasks have the same priority are ready:

• *Some RTOSs make it illegal to set two tasks with the same priority, and here the kernel limits the number of tasks in an application to the number of priority levels.*
• *Others will time slice between the two tasks(Round robin).*
• *Some will run one task until it blocks, then run the other task.*

Task object data:

• *Each task has an associated:*
    ⇒ *a name, a unique ID, a priority, a task control block (TCB), a stack, and a task routine.*

Task Context:

• *Every task has its own context (it's own Data). Every Task Created has its own data structure called Task Control Block (TCB).*
• *Task saves its data like: tasks status, ID , priority, stack, pointer, Pointer to function(task itself) in it's TCB.*
• *Task context is also saved in it's own stack and CPU registers.*

Context Switching between tasks:

• ***Context switching:*** *Is how to switch the processor between the context of one task to the another, so the system must:*
    • *Save the state of the old process,*
    • *Then load the saved state for the new process,*
    • *The new process continues from where it left off just before the context switch.*
• *When the task is not running, its context is frozen within the TCB, to be restored the next time the task runs.*
• ***The Dispatcher :***
    • *Is the part of the scheduler that performs context switching.*
• ***Context Switch Time:***
    • *Is the time it takes for the scheduler to switch from one task to another.*
    • *frequent context switching makes a performance overhead.*

Non Reentrant Function:

• *Is a function that can't be used between more than one task. Can't be interrupted or a data loss will happen.*

Example:

```
        Static int c Errors;
void vCounterErrors(int cNewErrors)
        {
                cErrors += cNewErrors;
```

}

Reentrant Function:

• *Is a function that can be used between more than one task without fear of data corruption.*
• *Can be interrupted at any time and resumed without loss of data; Reentrant functions either:*
    • *Use local variables, Or use protected global variables.*
    • *Can't Call other non reentrant functions.*

Gray area of reentrancy:

• *Some Functions and some operations on a shared data are processor and compiler dependent.*

How to protect Shared Data?

• *Using **Mutual Exclusion access.***
• *Examples on Mutual Exclusion methods are:*
    • *Disable and enable interrupts,*
    • *Disabling Scheduling, and*
    • *Using Semaphores.*

Mutual Exclusion

• *Shared Data is important for tasks to communicate.*
• *Mutual Exclusion access is a must when using any shared resource.*

Disabling and Enabling the interrupts:

• *Most of Systems have this technique.*
• *Example: µC/OS-II uses to micros :*

```
OS_ENTER_CRITICAL();
//Disable interrupts,
 /*
      Read/Write to the shared resource,
 */
 OS_EXIT_CRITICAL();
  //Reenable interrupts.
```

• *Disabling the interrupt for a long time affects the response to your system which is known by Interrupt Latency.*
• ***Interrupt Latency:** is The time taken by a system to respond to an interrupt.*
• *So disable the interrupt for as little time as possible, This is the only way for a task to share a variable with ISR.*

Disabling and Enabling the Scheduling:

• *If we don't share data with any ISR, then it's better to disable and enable scheduling.*
• *While the scheduler is locked, the interrupts is enabled, and if an interrupt happens, the ISR is executed immediately.*
• *As the scheduling is disabled, when the ISR finishes, the kernel will return to the interrupted task, not the highest priority one.*
• *Example: µC/OS-II uses to micros :*

```
 OSSchedLock();
//Disable Scheduling,
 /*
   Read/Write to the shared resource,
 */
  OSSchedunLock();
 //Re Enable Scheduling.
```

• *Disabling the scheduler also is not the best solution,*

Semaphores:

• *Is a kernel object that one or more threads of execution can acquire or release for the purposes of synchronization or mutuelle exclusion.*
• *A semaphore is like a key that enables a task to carry out some operation or to access a resource.*

• When a task acquires the semaphore,
    • No other task can access the resource that is protected by the semaphore.
    • Other tasks to acquire the semaphore will be suspended until the semaphore is released by its current owner.
• When a semaphore is created, the kernel assigns to it:
    -An associated semaphore control block (SCB),,a unique ID,a value (binary or a count) depending on it's type, a task-waiting list.

Semaphore Types:

1.Binary Semaphore:

• It's value = 0, if it's not available.
• It's value = 1, if it's available.
• when a binary semaphore is first created, it can be initialized to either available or unavailable.
• Binary semaphores are global resources shared between tasks.
• Any task can release it, even if the task did not initially acquire it.

2.Counting Semaphore:

• When creating a counting semaphore, assign the semaphore a count that denotes the number of semaphore tokens it has initially.
• It's Count = 0, if it's not available.
• It's Count > 0, if it's available.
• counting semaphores are global resources shared between tasks.Any task can release a counting semaphore token, even if the task making this call did not acquire a token in the first place.

N.B:

• Using semaphore to access shared data doesn't affect the interrupt latency.
• If ISR or the current running task makes a higher priority task to run it will run immediately.

Deadlock:

Also called Deadly Embrace. When two tasks are waiting the resource held by the other.
• Example:

1. Task1 has an exclusive access to Resource1,
2. And Task2 has exclusive access to Resource2.
3. If Task1 needs an exclusive access to Resource2,
4. And Task2 needs exclusive access to Resource1.

⇒ Both the tasks will be blocked, and a DeadLock happens.

Avoid Deadlock:

- Through a timeout, If the resource is not available for a certain time, the task will resume executing.

Resource Synchronization:

- Determines whether access to a shared resource is safe, and, if not, when it will be safe.
- Access by multiple tasks must be synchronized to maintain the integrity of a shared resource.

Semaphore for Task Synchronization:

- Semaphores are useful either for synchronizing execution of multiple tasks or for coordinating access to a shared resource.

Priority Inversion:

- Priority inversion is a situation in which a low-priority task executes while a higher priority task waits.

===> Solution for priority inversion:

Priority inheritance:

- The Priority Inheritance Protocol is a resource access control protocol that raises the priority of a task.

Race Condition:

- when two or more tasks have access to a shared resource and they try to edit it at the same time. To prevent race conditions from occurring, you would typically put a lock around the shared data to ensure only one thread can access the data at a time.

CPU Starvation:

CPU starvation: occurs when higher priority tasks use all of the CPU execution time and lower priority tasks do not get to run. In a preemptive multitasking environment, If higher priority tasks are not designed to block, CPU starvation can result.

Mutex:

- Mutex is short for MUTual EXclusion. Mutex is a special type of binary semaphore used for controlling access to the shared resource.

- Mutexes include a priority inheritance mechanism to avoid extended priority inversion problem. This ensure the higher priority task is kept in the blocked state for the shortest time possible, Priority inheritance does not cure priority inversion! It just minimizes its effect in some situations. Hard real time applications should be designed such that priority inversion does not happen in the first place.

Mutex vs Binary Semaphore

Ownership:

Mutex semaphore is "owned" by the task that takes it, so when a task locks (acquires) a mutex only it can unlock (release) it. Binary semaphore has no owner, can be unlocked by any task,

Usage:

a mutex is a locking mechanism used to synchronize access to a resource. Semaphore is signaling mechanism ("I am done, you can carry on" kind of signal).

Determinism vs Responsiveness

Determinism: determine the time that every task will be executed,

Responsiveness: How your application be responsive to (External or Internal) events,

RTOS Increase responsiveness and decrease determinism.

Explain real-time communications.

ANS : Real-time communications (RTC) is any mode of telecommunications in which all users can exchange information instantly or with negligible latency.

RTC can take place in half-duplex or full duplex modes.In half-duplex RTC, data can be transmitted in both directions on a single carrier or circuit but not at the same time. In full-duplex RTC, data can be transmitted in both directions simultaneously on a single carrier or circuit. RTC generally refers to peer-to-peer communications, not broadcast or multicast.

What are the rules you follow when you are writing critical sections of code?

a)Use Atomic Instructions

b) Remember to enable interrupts

c) Make the critical section code as small as possible. (Prefer not more than 20 instructions)

d) Prefer not to call other functions from the critical

Check the example below.

```
fnA()
{
/* Critical Section Start */
Disable_Interrupt();
Some Instructions A ....
Call FnB();
/* do Something B */
Some Instructions B ....
/* Critical Section End */
}
fnB()
{
/* Critical Section Start */
Disable_Interrupt();
Some Instructions ..
Enable_Interrupts();
/* Critical Section End */
}
```

Now the Enable_Interrupts in fnB() will enable the interrupts and hence "Some Instructions B" in fnA() which should have been in the critical section will no more be in the critical section because the interrupts are already enabled!!

What Is A Non Reentrant Code?

Reentrant code is code which does not rely on being executed without interruption before completion. Reentrant code can be used by multiple, simultaneous tasks. Reentrant code generally does not access global data. Variables within a reentrant function are allocated on the stack, so each instance of the function has its own private data. Non -reentrant code, to be used safely by multiple processes, should have access controlled via some synchronization method such as a semaphore.

How Is Rtos Different From Other Os?

A RTOS offers services that allow tasks to be performed within predictable timing constraints.

Is Unix A Multitasking Or Multiprocessing Operating System?

Unix is a multitasking operating system, multiprocessing means it can run on multiple processors, the multi-processing os coordinates with multiple processors running in parallel.

What Is A Core Dump?

A core dump is the recorded state of the working memory of a computer program at a specific time, generally when the program has terminated abnormally includes the program counter and stack pointer, memory management information, and other processor and operating system flags and information a fatal error usually triggers the core dump, often buffer overflows, where a programmer allocates too little memory for incoming or computed data, or access to null pointers, a common coding error when an unassigned memory reference variable is accessed.

What Is Stack Overflow And Heap Overflow?

- stack overflow occurs when when the program tries to access memory that is outside the region reserved for the call stack. call stack contains the subroutines called, the local variables.

- A heap overflow is a form of buffer overflow; it happens when a chunk of memory is allocated to the heap and data is written to this memory without any bound checking being done on the data. This is can lead to overwriting some critical data structures in the heap such as the heap headers, or any heap-based data such as dynamic object pointers, which in turn can lead to overwriting the virtual function table.

What Is A Flat Memory Model And A Shared Memory Model?

- In a flat memory model the code and data segment occupies single address space.

- In a shared model the large memory is divided into different segments and needs a qualifier to identify each segment.

- In a flat memory model the programmer doesn't need to switch for data and code.

What Is Paging, Segmentation Y Do We Need It?

Paging: *Paging is a technique where in the OS makes available the data required as quickly as possible. It stores some pages from the aux device to main memory and when a prog needs a page that is not on the main memory it fetches it from aux memory and replaces it in main memory. It uses specialised algorithms to choose which page to replace from in main memory*

Caching: *It deals with a concept where the data is temporarily stored in a high speed memory for faster access. This data is duplicated in cache and the original data is stored in some aux memory. This concepts brings the average access time lower.*

Segmentation: *Segmentation is a memory management scheme. This is the technique used for memory protection. Any accesses outside the permitted area would result in segmentation fault.*

Virtual Memory: *This technique enables non contiguous memory to be accessed as if it were contiguous.*

Why Do We Require Semaphore?

For process synchronization, it is a mechanism to invoke the sleeping process to become ready for execution. Its mechanism where a process can wait for resources to be available.typical example is producer consumer process. The producer process creates resources and signals the semaphore saying resources are available. Consumer process waiting on the semaphore gets the signal that a resource is available.

Write A Small Piece Of Code Protecting A Shared Memory Variable With A Semaphore?

```
int global_i;
void increment_shared_memory
{
wait(semaphore);
global_i++;
signal(semaphore);
}
```

What Are The Different Inter Process Communications?

semaphore, mutex, message passing, shared memory, socket connections.

What is the difference between IRQ and FRQ?

FIQ is the fast interrupt i.e. the latency time taken is less because in the FIQ mode (in case of ARM architecture where these modes are encountered) the mode has an additional set of 8 general purpose registers which means In case of the banked registers there is no need to store these registers into stack when the interrupt occurs. That means r0 to r7 only needs to be stored while moving from User mode to FIQ mode whereas while moving from to IRQ mode r0 to r12 needs to be stored.

What are the major concerns about any RTOS selection?

- Interrupt Latency means the time taken by the processor to pass the control to ISR after the interrupt is raised. Certainly this is the hardware feature.

- Footprint of the OS matters because with the same compiler and same optimization techniques, Different OS's will have a different footprint. So this can be looked upon while selecting the RTOS.

- RTOS can be chosen looking at various API support it provides. Synchronization support, Scheduler algos, and memory management of the OS.

Write a code to connect Hardware interrupt to ISR?

In Keil

```
void serial_ISR() interrupt 4 using 0
{
}
```

In AVR studio

```
SIGNAL(Inteerupt_no)
{
}
```

hardware interrupt normally redirects the uC to predefined addresses. In case of high end processors the interrupt table will decide the interrupt vector address and whenever interrupt pin goes low, the table is searched for type and source of interrupt.

SHORT NOTES ON INTERRUPT HANDLING

1) An interrupt is generated by some HW source or SW source (timer or software event).

2) CPU invokes the kernel interrupt handler.

3) Kernel interrupt handler invokes the vector handler which returns the vector number. (Vector handler has the mapping from vector number to interrupt source).

4) Kernel invokes the mask handler which masks all existing equal or low priority interrupts.

5) An interrupt routine associated with the vector number is invoked.

6) Kernel invokes the mask handler again to restore the old mask.

THREAD POOLING

Thread pool is a collection of managed threads usually organized in a queue, which execute the tasks in the task queue.

Creating a new thread object every time you need something to be executed asynchronously is expensive. In a thread pool you would just add the tasks you wish to be executed asynchronously to the task queue
and the thread pool takes care of assigning an available thread, if any, for the corresponding task. As soon as the task is completed, the now available thread requests another task (assuming there is any left).

When would you choose bottom up methodology?

- In Bottom Up Methodology, we build the bottom-most layer first and then ascend onto the actual application i.e. all the smaller chunks of code or assisting functions are build first and then the bigger function which uses these are built. Bottom Up methodology is followed when we are definite about the number, type and functionality of the Bottom most functions(Assisting Functions) to be defined. Once all these functions are defined, we build the bigger system above it.

- Top down approach is followed when we are not very sure about the number/type of functions that need to be implemented, so the application is started to be implemented and as and when smaller functions are needed, they are built.Top down methodology is one where overall picture of the system is defined first. Then we go into the details of the sub-systems in the next level and so on. So to brief this, first the complex system is seen as a whole and then this is broken down to smaller simpler pieces.

Explain what is the difference between mutexes and semaphores?

1.Mutexes:

- A mutex object enables one thread into a controlled section, forcing other threads which tries to gain access to that section to wait until the first thread has moved out from that section.

- Mutex can only be released by thread which had acquired it

- Mutex will always have a known owner

- Mutex is also a tool that is used to provide deadlock-free mutual exclusion (either consumer or producer can have the key and proceed with their work)

- Mutexes by definition are binary semaphores, so there are two states locked or unlocked.

2. Semaphores

- Semaphore allows multiple access to shared resources

- A semaphore can be signaled from any other thread or process.

- While for semaphore you won't know which thread we are blocking on

- Semaphore is a synchronization tool to overcome the critical section problem- Semaphores are usually referred to counted locks

Explain whether we can use semaphore or mutex or spinlock in interrupt context in Linux Kernel?

- Semaphore or Mutex cannot be used for interrupt context in Linux Kernel. While spinlocks can be used for locking in interrupt context.

What type of scheduling is there in RTOS?

- RTOS uses preemptive scheduling. In preemptive scheduling, the higher priority task can interrupt a running process and the interrupted process will be resumed later.

What is priority inversion?

- If two tasks share a resource, the one with higher priority will run first. However, if the lower-priority task is using the shared resource when the higher-priority task becomes ready, then the higher-priority task must wait for the lower-priority task to finish. In this scenario, even though the task has higher priority it needs to wait for the completion of the lower-priority task with the shared resource. This is called priority inversion.

What is priority inheritance?

Priority inheritance is a solution to the priority inversion problem. The process waiting for any resource which has a resource lock will have the maximum priority. This is priority inheritance. When one or more high priority jobs are blocked by a job, the original priority assignment is ignored and execution of critical section will be assigned to the job with the highest priority in this elevated scenario. The job returns to the original priority level soon after executing the Critical section.

How many types of IPC mechanism do you know?

- Different types of IPC mechanism are -

- *Pipes*
- *Named pipes or FIFO*
- *Semaphores*
- *Shared memory*
- *Message queue*
- *Socket*

What is spin lock?

If a resource is locked, a thread that wants to access that resource may repetitively check whether the resource is available. During that time, the thread may loop and check the resource without doing any useful work. Suck a lock is termed as spin lock.

What is the difference between binary semaphore and mutex?

The differences between binary semaphore and mutex are as follows -

1. Mutual exclusion and synchronization can be used by binary semaphore while mutex is used only for mutual exclusion.

2. A mutex can be released by the same thread which acquired it. Semaphore values can be changed by other thread also.

3. From an ISR, a mutex can not be used.

4. The advantage of semaphores is that, they can be used to synchronize two unrelated processes trying to access the same resource.

5. Semaphores can act as mutex, but the opposite is not possible.