Real Time Operating Systems

LY ETRX
Sem VIII

Arati Phadke
AY 2021-22

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Somaiya TRUST

Computer Organization and Architecture

Course Outcomes

At the end of successful completion of the course the student will be able to

CO1: Understand real time operating system concepts.

CO2: Create various instances of a task and send and receive data to and from a queue

CO3: Implement the APIs within an Interrupt Service Routine

CO4: Develop algorithms for real time processes using FreeRTOS

CO5: Understand memory allocation schemes

Module 3

Interrupt Management

- mechanism of interrupts,
- priority of interrupts
- wait for interrupts,
- time-triggered and event-triggered systems,
- watchdog timer,
- implementation of interrupts
- Interrupts API in FreeRTOS
- Deferred Servicing of Interrupts

Interrupts in RTOS

(A practical introduction to real-time systems for undergraduate engineering)

Sources of Interrupts

- Real-time clock
- Watchdog timer.
- UARTs (universal asynchronous receiver/transmitters),
- CAN (controller area network) bus,
- Pulse-width modulators,
- Other peripherals through I^2C (inter-integrated circuit) buses
- SPI data flash memory,
- Brown-out detection.
- Divide by zero, memory protection, memory management etc

Interrupts in RTOS

Mechanism of interrupts

- interrupting the processor,
    - a single line used by all of the devices,
    - a line dedicated for each device
- stop the execution of the current task or thread:
    - complete current instruction
- execute code relevant to the interruption
    - Selecting ISR, Interrupt Vector, storing return address etc.
- return to the normal execution of code.
    - Special return from Interrupt instruction

Interrupts in RTOS

Characteristics of interrupt service routines (ISRs)

An interrupt service routine should be as short as possible, to
1. allow the system to quickly return to normal operation,
2.minimize the possibility of other interrupts arriving
3. reduce the probability of a programming bug during the ISR
ISRs should not call functions, but if necessary, such functions must be re-entrant.
All calls to non-reentrant functions be wrapped in blocks of code where all interrupts are disbaled and enabled back after non-reentrant function has returned

Interrupts in RTOS

- Masking of interrupts
- Nesting of Interrupts
- Waiting for an interrupt
  - A global variable is shared by the interrupt service routine (ISR) and the task or thread wanting to wait on the interrupt and this global variable is false (yielding the processor if it is not ready),
  - The task or thread continually checks that variable to see if it is set to true,
  - When the ISR services the routine, it accesses the device, copies whatever information is necessary and then sets the shared global variable to true.
  - Variable needs to be volatile
- Time triggered systems : sampling frequency , multiple events
- Event Triggered Systems
- Watchdog timer: possibility of deadlock, possibility of killing some tasks, reset

Interrupts in RTOS

(161204 Mastering the FreeRTOS Real Time Kernel-A Hands-On Tutorial Guide)

- How should the event be detected? : interrupts Vs polling
- How much processing should be performed inside the interrupt service routine (ISR), and how much outside? **Short ISRs**
- How events are communicated to the main (non-ISR) code, and how can this code be structured to best accommodate processing of potentially asynchronous occurrences?
- FreeRTOS provides features that allow the chosen strategy to be implemented in a simple and maintainable way.

Interrupts in RTOS

Priority in task and ISR

- A task is a software feature that is unrelated to the hardware on which FreeRTOS is running.
- The priority of a task is assigned in software by the application writer, and a software algorithm (the scheduler) decides which task will be in the Running state.
- Although written in software, an interrupt service routine is a hardware feature because the hardware controls which interrupt service routine will run, and when it will run.
- Tasks will only run when there are no ISRs running, so the lowest priority interrupt will interrupt the highest priority task, and there is no way for a task to pre-empt an ISR.
- Setting of priority of Interrupts and RTOS kernel in FreeRTOS

FreeRTOS API from an ISR

- Interrupt safe API (FromISR)
- If the same version of an API function could be called from both a task and an ISR then:
    - The API functions would need additional logic to determine if they had been called from a task or an ISR. The additional logic would introduce new paths through the function, making the functions longer, more complex, and harder to test.
    - Some API function parameters would be obsolete when the function was called from a task, while others would be obsolete when the function was called from an ISR.
    - Each FreeRTOS port would need to provide a mechanism for determining the execution context (task or ISR). Architectures on which it is not easy to determine the execution context (task or ISR) would require additional, wasteful, more complex to use, and non-standard interrupt entry code that allowed the execution context to be provided by software.

FreeRTOS API from an ISR

Disadvantage of interrupt safe API

- sometimes it is necessary to call a function that is not part of the FreeRTOS API, but makes use of the FreeRTOS API, from both a task and an ISR.
- Defer interrupt processing to a task, so the API function is only ever called from the context of a task.
- If you are using a FreeRTOS port that supports interrupt nesting, then use the version of the API function that ends in “FromISR”, as that version can be called from tasks and ISRs

Deferred Interrupt Processing

- Short ISRs
  - Even Highest priority tasks can only run if no interrupts are being serviced by the hardware.
  - ISRs can disrupt (add ‘jitter’ to) both the start time, and the execution time, of a task.
  - Depending on the architecture on which FreeRTOS is running, it might not be possible to accept any new interrupts, or at least a subset of new interrupts, while an ISR is executing.
  - The application writer needs to consider the consequences of, and guard against, resources such as variables, peripherals, and memory buffers being accessed by a task and an ISR at the same time.
  - Some FreeRTOS ports allow interrupts to nest, but interrupt nesting can increase complexity and reduce predictability. The shorter an interrupt is, the less likely it is to nest.

Deferred Interrupt Processing

- An interrupt service routine must record the cause of the interrupt, and clear the interrupt. Other processing necessitated by the interrupt can often be performed in a task, allowing the interrupt service routine to exit as quickly as is practical.
- This is called ‘deferred interrupt processing’, because the processing is ‘deferred’ from the ISR to a task.
- Deferring interrupt processing to a task also allows the application writer to prioritize the processing relative to other tasks in the application, and use all the FreeRTOS API functions.
- If the priority of the task to which interrupt processing is deferred is above the priority of any other task, then the processing will be performed immediately, just as if the processing had been performed in the ISR itself.

Deferred Interrupt Processing

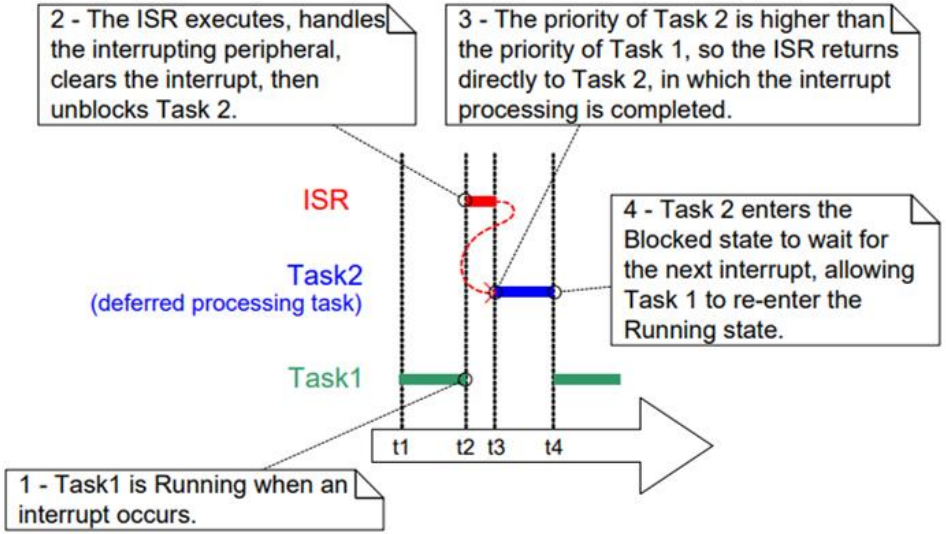

Figure 48 Completing interrupt processing in a high priority task

Interrupt Processing Demo

https://youtu.be/qsflCf6ahXU

https://youtu.be/qsflCf6ahXU