REAL TIME OPERATING SYSTEMS

Lesson-12:

Real Time Operating System based System design Principles

Design with the ISRs and Tasks

- The embedded system hardware source calls generates interrupts
- ISR ─ only post (send) the messages
- Provides of nesting of ISRs, while tasks run concurrently
- A task─ wait and take the messages (IPCs) and post (send) the messages using the system calls.
- A task or ISR should not call another task or ISR

Each ISR design consisting of shorter code

- Since ISRs have higher priorities over the tasks, the ISR code should be made short so that the tasks don’t wait longer to execute.
- A design principle is that the ISR code should be optimally short and the detailed computations be given to an IST or task by posting a message or parameters for that.
- The frequent posting of the messages by the IPC functions from the ISRs be avoided

Design with using Interrupt Service Threads or Interrupt Service tasks

- In certain RTOSes, for servicing the interrupts, there are two levels, fast level ISRs and slow level ISTs, the priorities are first for the ISRs, then for the ISTs and then the task
- In certain RTOSes, three levels

Design with using Interrupt Service Threads

- ISRs post the messages for the ISTs and do the detailed computations.
- If RTOS is providing for only one level, then use the tasks as interrupt service threads

Design Each Task with an infinite loop

- Each task has a while loop which never terminates.
- A task waits for an IPC or signal to start.
- The task, which gets the signal or takes the IPC for which it is waiting, runs from the point where it was blocked or preempted.
- In preemptive scheduler, the high priority task can be delayed for some period to let the low priority task execute

Design in the form of tasks for the Better and Predictable Response Time Control

- provide the control over the response time of the different tasks.
- The different tasks are assigned different priorities and those tasks which system needs to execute with faster response are separated out.

Response Time Control

- For example, in mobile phone device there is need for faster response to the phone call receiving task then the user key input.
- In digital camera, the task for recording the image needs faster response than the task for down loading the image on computer through USB port

Design in the form of tasks Modular Design

- System of multiple tasks makes the design modular.
- The tasks provide modular design

Design in the form of tasks for Data Encapsulation

- System of multiple tasks encapsulates the code and data of one task from the other by use of global variables in critical sections getting exclusive access by mutex

Design with taking care of the time spent in the system calls

- expected time in general depends on the specific target processor of the embedded system and the memory access times.
- Some IPCs takes longer than the other

Design with Limited number of tasks

- **Limit the number of tasks and select the appropriate number of tasks to increase the response time to the tasks, better control over shared resource and reduced memory requirement for stacks**
- The tasks, which share the data with number of tasks, can be designed as one single task.

Use appropriate precedence assignment strategy and Use Preemption

- Use appropriate precedence assignment strategy and Use Preemption in place of Time Slicing

Avoid Task Deletion

- Create tasks at start-up only and *avoid creating and then deleting tasks later at the later times*.
- Deleting only advantage─ availability of additional memory space

Use CPU idle CPU time for internal functions

- Read the internal queue.
- Manage the memory.
- Search for a free block of memory.

Design with Memory Allocation and De-Allocation By the Task

- If memory allocation and de-allocation are done by the task then the number of functions required as the RTOS functions is reduced.
- This reduces the interrupt-latency periods. As execution of these functions takes significant time by the RTOS whenever the RTOS preempts a task.
- Further, if fixed sized memory blocks are allocated, then the predictability of time taken in memory allocation is there

Design with taking care of the Shared Resource or Data among the Tasks

- The ISR coding should be as like a reentrant function or should take care of problems from the shared resources or data such as buffer or global variables
- if possible, instead of disabling the interrupts only the task-switching flag changes should only be prevented. [It is by using the semaphore.]

Design with limited RTOS functions

- Use an RTOS, which can be configured. RTOS provides during execution of the codes enabling the limited RTOS functions. For example, if queue and pipe functions are not used during execution, then disable these during run
- Scalable RTOS
- Hierarchical RTOS

Use an RTOS, which provides for creation of scalable code

- Only the needed functions include in the executable files, and those functions of kernel and RTOS not needed do not include in the executable files

Use an RTOS, which is hierarchical

- The needed functions extended and interfaced with the functionalities
- Configured with specific processor and devices

Encapsulation Using the Semaphores

- Semaphores, queues, and messages should not be not globally shared variables, and let each should one be shared between a set of tasks only and encapsulated from the rest
- A semaphore encapsulates the data during a critical section or encapsulates a buffer from reading task or writing into the buffer by multiple tasks concurrently

Encapsulation Using Queues

- A queue can be used to encapsulate the messages to a task at an instance from the multiple tasks.
- Assume that a display task is posted a menu for display on a touch screen in a PDA
- Multiple tasks can post the messages into the queue for display.
- When one task is posting the messages and these messages are displayed, another task should be blocked from posting the messages.
- We can write a task, which takes the input messages from other tasks and posts these messages to the display task only after querying whether the queue is empty

Summary

We learnt

- There are certain design principles observed when using the RTOS

End of Lesson 12 of Chapter 8