Communication Software and Middleware

CPE 545A

Project: Timer Manager

Name: Prerakkumar Doshi

CWId: 10411489

Email: pdoshi5@stevens.edu

Timer Manager Design

Design Consideration

- The Timer Manager is designed considering the requirement for the Real Time Operating System being developed in-house.
- Whole design is modular and all the functions are separated in two groups:
  - o Public APIs – Available for the user to Create, Start, Get Info, Stop and Delete Timers etc o Private functions – Required for internal use to handle the various internal operations

OS Tick

- In the RTOS, there will be provision to have the OS Tick from the kernel But here to Demonstrate the Timer Manager Functionality, one Linux Posix timer is created with the 0.1 ms resolution
- Therefore, at every 0.1 ms, RTOSTmrSignal() function will be called
- RTOSTmrSignal() function send the signal to RTOSTmrTask indicating that it's time to update the Timers
- This signaling mechanism is designed by using the Semaphore
- RTOSTmrTask will wait for the Semaphore by sem_wait() and RTOSTmrSignal() will send the signal by using the sem_post()

Timer Pool

- The Timer pool design is user friendly
- User can enter the required number of timers for the OS run time
- At the time of Initialization, It will create the Timer pool by allocating the Memory from Heap and it will create the Link list
- The head of the Timer Pool will be stored in FreeTmrListPtr
- The number of free timers available at any time will be indicated by FreeTmrCount

- When calling the RTOSTmrCreate() Function, it will allocate the Timer object from the pool and decrement the FreeTmrCount

Hash Table

- If the number of timers required is very large and if we are using the normal approach, then all the timers will be in a link list.
- It will be very time consuming to go through the whole link list for checking the RTOSTmrMatch value with the RTOSTmrTickCtr
- To improve the design and to reduce the time for checking the Timer hit, Hash Table is adopted in the design
- The size of hash table is kept 10 and hash function = (RTOSTmrMatch % 10)
- Therefore at the time of adding any Timer, it will first calculate the index and then it will add the timer object in the hash_table[index] link list
- By this, the whole timer objects will be divided in the different link list.
- At every OS Tick, RTOSTmrTask would need to calculate the index based on RTOSTmrTickCtr % 10 and check the timer objects available in that link list for the timer expired

Timer Task

- Timer task is in the centre of whole design
- It is created as a separate thread which will run continuously
- It is signaled at every OS Tick and it will increment the RTOSTmrTickCtr
- On every execution, it will calculate the index by index = RTOSTmrTickCtr % 10 and goes to that hash table index
- It will check whether any timers running or not
- If any Timers are available in the Link list, it will go through the each objects and compare the RTOSTmrMatch with the

RTOSTmrTickCtr

- If it is equal then it will call its callback function and remove the object from the hash table
- After that, it will check the timer object's type. If it is periodic then it will again calculate the RTOSTmrMatch = RTOSTmrTickCtr + RTOSTmrPeriod and will add timer object in the required hash table based on the index

Resource Protection

- In the Multithreaded environment, Many tasks can call the Timer APIs to Create, Start, Get info, Stop and Delete the timer objects
- In these scenario, It can cause the issue if the resources like Timer Pool and Hash Table are not protected
- To protect the resources from the different functions being used at the same time, Mutex is used for the protection
- hash_table_mutex – protects the Hash Table Resources
- timer_pool_mutex – protects the Timer Pool Resources

Public Function (APIs)

- RTOSTmrCreate – Function to Create the Timer
- RTOSTmrStart - Function to start the Timer
- RTOSTmrStop - Function to Stop the Timer
- RTOSTmrDel – Function to Delete the Timer
- RTOSTmrNameGet - Function to get the Name of a Timer
- RTOSTmrRemainGet - To Get the Number of ticks remaining in time out
- RTOSTmrStateGet - To Get the state of the Timer
- RTOSTmrSignal - Function called when OS Tick Interrupt occurs which will signal the RTOSTmrTask() to update the Timers

Private Functions

- RTOSTmrInit - Timer Initialization Function
- RTOSTmrTask - Timer Task to Manage the Running Timers
- OSTickInitialize - Function to Setup the Timer of Linux which will provide the Clock Tick Interrupt to the Timer Manager Module
- Create_Timer_Pool - Create Pool of Timers
- alloc_timer_obj - Allocate a timer object from free timer pool
- free_timer_obj - Free the allocated timer object and put it back into free pool
- init_hash_table - Initialize the Hash Table
- insert_hash_entry - Insert a Timer Object in the Hash Table
- remove_hash_entry - Remove the Timer Object entry from the Hash Table

Time Display

- Time is displayed in the Local time with Time and Data format
- The function required to display the time is

    time()
    localtime()
    asctime()