Implementation of Task Recursion and Scheduling on TI Stellaris Evalbot using Realtime Kernel Micrium OS-III

Saurabh Adhikari
Computer Engineering Department
Nanyang Technological University
Singapore,694941
saurabh009@e.ntu.edu.sg

Abhinav Kulkarni
Computer Engineering Department
Nanyang Technological University
Singapore,694941
abhinav004@e.ntu.edu.sg

***Abstract*—Scheduling is an important part of the micrium real time operating system as it is responsible for determining which task to be dispatched at what instant of time which ensures that the task meets its deadline. There are four hardware timers We have implemented the periodicity of the tasks using the hardware Systick/Clock timer running at 1000Hz which reduces the overhead and is accurate when compared with the others software timers. Rate Monotonic scheduling is used to schedule the tasks based on the respective release time of the tasks. To make our implementation more effective we have used different data structure each having its own advantages. For our implementation we have used Texas Instrument Stellaris Robotic Evaluation Board as a platform which uses LM3S9B92.**

I. Introduction

State-of-the-art realtime kernel Micrium OS-III offers a lot of features and services so that it can be used efficiently with todays processors. The new version of the micrium comes with the additional round robin scheduling of tasks holding equal priority. Micrium is a preemptive priority based kernel which supports multiple tasks having equal priority level. Each task holds a priority on the basis of which the the scheduling is done. The task with the highest priority is made ready to run. The task calls to the micrium functions that take care of the scheduling. Mainly there are two types of scheduler that are task-level schedulers and the ISR-level schedulers. The task level scheduler is called by the task-level code and the ISR-level schedulers called at the end of the ISR. OSSched() is the scheduler that is called by the task and if the interrupts call the scheduler than it will simply return. OSIntExit() is the scheduler that is called at the end of each ISR. Interrupts are handled by micrium using two methods namely Direct and Deferred Post. Both methods can be used in preemptive scheduling.

II. Task

A. Task Creation

OSTaskCreate() is the API that is used to create task and consist of multiple arguments. We can create our own API for creating the task. Once the task is created, it would be placed in the readyList and based on the priority the task would switch from ready to run state.

B. Task Management

After the task is completed it should delete itself by calling OSTaskDel() API. This means that the task is not deleted rather than that the resources allocated to the task like pointers, registers are released. This creates an additional overhead for deletion of a task in a multitasking system. Micrium task management services are divided into General, Signalling Task and Sending Message to a task. Different states in which the micrium task may exist are Dormant, Ready, Running,interrupted and Pending states.

III. Data Structure

A. Splay Tree

A very interesting data structure which is self balancing binary tree with the property that recently accessed elements are very quick to be accessed and the min key will always be found in the leftmost leaf node. Different Splaying operations are performed to balance the binary search tree. The time complexity for the insertion,deletion and searching is O(logn) for the splay tree.

B. Binomial Heap

Binomial Heap is the combination of the binary trees which consist of unique binary trees. The best thing about the Binomial is that it provides merging and union operation of the binary trees and heap respectively. They are also known as mergeable heap abstract data type. We have used the binary min heap in place of the binomial heap as moulding the binomial heap according for our functionality seems to be challenging but at the same time binary heap when implemented in min form have its own advantages in terms of the overhead of the searching and deletion of the min from the heap. Retrieving the minimum key from the min heap takes O(n) time whereas it takes O(logn) to retrieve the min key in the binomial heap.

IV. Recursion Implementation

The implementation of the recursive tasks was done exploiting and understanding OS_TickTask() which is periodic and repeats after every one millisecond since the frequency of the Systick at which it generates tick ISR is 1000Hz. OS_TickListUpdate() API updates the OSTickCtr. This means we need to add the OSTickCtr to the previous occurrence period of the task.

NextPeriod[i].period = heap.Task[min]->period;

NextPeriod[i].key = (heap.Task[min]->period + OSTickCtr);

NextPeriod[i].p_tcb = p_tcb;

Now the question arises why OS_TickTask() for recursion?

Recursion can be implemented by creating software timers too and make them work in the periodic mode. But these timers are not exactly accurate and why to add another overhead. Thus Systick/ Clock Timer which is one of the micrium hardware timer was used which is efficient.

The data structure with which we implemented recursion was using a splay tree as well as binary min heap. There were few limitation while implementation of recursion using the splay tree which we will cover later.

V. Scheduler Implementation

Rate Monotonic is the technique based on which the frequently occurring task is assigned a high priority. The task with the highest rate of execution will be provided a high priority. We implemented Rate Monotonic scheduling using the Splay tree. Splay tree is a self adjusting binary search tree. The task with the minimum period value will be fetched from the binary min heap. The TCB of that particular task consist of the priority value which are assigned in the order the task are fetched from binary min heap and put in the splay tree.

The recent accessed value are splayed to the root in splay tree as a result of which there will be a minimum priority value at the leftmost leaf of the splay tree. This TCB has to be inserted into the readylist. OS_RdyListInsertTail() API is used to insert the TCB into the OS ready List and the OS_PrioInsert() function will be used to set the bit corresponding to the priority level in the bitmap table.

The task from our splay tree will move to the readylist after setting it's priority using OS_PrioInsert() API. After this the task from the readylist will be removed by using OS_RdyReadyListRemove() API.

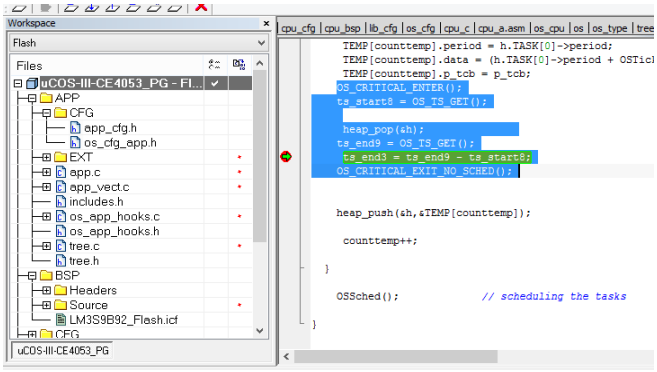
VI. API

Following are the API used for the achieving the recursion:

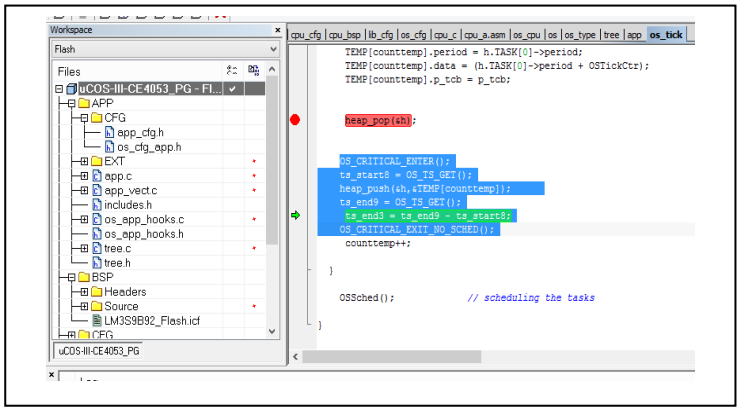
API	Function
OSRecTaskCreate()	App used to create the task
Insert_Ready_List()	App called in OSTickTask()

VII. Overheads

Overheads associated while performing the recursion using the min heap.



Overhead for the single pop: 94 ticks



Overhead for the single push: 82 ticks

VIII. LIMITATIONS

1). The implementation of the recursion using the splay tree was having a lot of overhead and moreover more than two tasks with same period value was inconsistent. The average overhead for the deletion of node from tree was coming out to be 242 ticks.

2). During working out with recursion using splay tree we found that few of the tasks with the same period were getting inserted into the Ready List properly.

Note: when the function is being inserted into the ready list, every time its stack is being reinitialized using a loop from top to bottom. This happens since the task is completely deleted along with its particular from the tree(no local copy is maintained). This causes too much overhead.

REFERENCES

[1] https://doc.micrium.com/display/osiiidoc/uC-OS-III+Reference+Manual

[2] http://www.geeksforgeeks.org/data-structures/