Journal of Advanced Computing and communication Technologies *(ISSN:* 2347 - 2804*)*
Volume No.2 Issue No. 4, August 2014

Implementation and evaluation of novel scheduler of *UC/OS* (RTOS)

By

Vigya Jindal, Vivek Agarwal
Senior software Engineer, HCL tech Noida, India
Assistant professor, Galgotia college of Engineering and Technology, Gr Noida India
vigyafunty@gmail.com, Vigya.jindal@hcl.com, Vivek.ag86@gmail.com

ABSTRACT
At present, there are more than a billion embedded applications either real time or non-real time. Among them, many appli cations require Prioritized queue of periodic tasks. UC/OS, one of the most widely Used real-time kernels in industry, has preemptive scheduler and doesn't Support two tasks at same priority level. This work proposes a modification in scheduler of UC/OS (RTOS) to make it more exible in handling a periodic and periodic task. The proposed scheduler allows assigning same priority to more than one task. Tasks with same priority are in queue, where their position is decided on first come (created) first Served basis. Time-slicing is used for scheduling tasks in same queue. Emendations in the task management and scheduler have been explained in detailed. Evaluation, to observe the effect of modified scheduling on Overhead of system calls is done on an evaluation board.
.

General Terms
Scheduling, Feedback, priority, tasks, queue, pre-empted semaphore, Mutex, Message queue, and message box, Time management, Simple memory management, ISR, Kernel**,** 32 Bit processor, Micrim.

Keywords
OSIntExit, OSTimeTick, OSSched, OSSemPend, OSSemPost, UC/OS, TCB, OSTimeDly, OSTCBTSlice, OSRdyGrp, OSTCBRR, OSRdyTbl, OSMapTbl, OSTCBTSlice, OSTCBnext, OSTCBPrev, OSTCBProiTB1.

1. INTRODUCTION
As real time systems are becoming pervasive, need for real time operating systems is growing. Among all RTOSs available, for any embedded real time system, a RTOS is preferred according to compatibility between its characteristics and requirements of the application. One of the key characteristic is the scheduling technique being used in it. In several embedded applications, there are many tasks which are periodic as well as aperiodic. For example, avionics in an airplane has many modes, for a say, normal mode and emergency mode. There are certain periodic tasks, like actuator regulation, signal acquisition; action planning, monitoring etc. need to be executed with a frequency, in emergency mode as well as in normal mode. An interrupt causes airplane to go in emergency mode from normal mode. In such a case, there is a need of two priority level queue of tasks, one for emergency mode tasks and other for normal mode tasks. UC/OS[1] is an open-source real-time kernel designed by Micrim, Inc. It is a small but significantly powerful kernel. *UC/OS* is one of the most widely used real-time kernels in industry, as it has been licensed by many embedded system companies. It has many capabilities RTOS capabilities:
(1) Priority-based preemptive scheduling;
(2) Inter-task communications via semaphore, Mutex, Message queue, and message box;
(3) Time management;
(4) Simple memory management.
Unfortunately, UC/OS doesn't allow to assign same priority to more than one task. This paper proposes the combination of above two scheduling techniques in *UC/OS*, as a target platform, in order to make it more exible in handling both periodic and a periodic task. There are many amendments related to task management, scheduler and system calls related to time. Detailed discussion with reasoning on each amendment is done to avoid any obscurity. To evaluate the performance of new kernel in comparison to old one, it is ported to evaluation board having 32 bit processor.

2. Backgrounds
In general, any hard real time system has to handle both hard and soft tasks. How tasks are being handled, depends on scheduler of the RTOS. The two type of scheduling algorithms are;
1) A periodic- for tasks with irregular arrival times and
2) Periodic- for tasks with regular (constant) arrival time [4].
Preemptive scheduling, an example of a periodic scheduling, is more popular in RTOS in which a task can be preempted by a higher priority task. Timeline Scheduling (TS)[3], also known as a cyclic executive, is one of the most used approaches to handle periodic tasks. A task is only allowed to execute for a certain amount of time, in which other task can't preempt it. In this way, it prevents high priority task mono-policing CPU. The difference between preemptive and time-slicing method is being depicted through Fig.1. In next section, a combination of these scheduling methods has been proposed for UC/OS [2].

3. Proposed Scheduling
The combination of two scheduling techniques, mentioned in above section, is quite similar to Multi Feedback queue scheduling. It has same system of queues with different

priorities, but scheduling is disparate. In MFQS, if the task uses too much CPU time it will be moved to a lower-priority queue. Similarly, a task that waits too long in the lower-priority queue may be moved to a higher priority queue. In the case of proposed scheduling, priorities of queues are fixed and no task can move to another queue. If any task is ready in higher priority queue, then it can pre-empt low priority task at that instant. Otherwise, low priority queue will be scheduling it tasks with time-slicing method without any preemption. For example, in Fig.2 task j2 of queue with priority p2 is pre-empted by queue with priority p3 when it is being executed. After complete execution of j4, j2 again resumed which is followed by j3 in same queue with respect to time slicing. In Fig.2, tasks in queue with priority p2 is pre-empted by higher priority queue after their 1st period of time slicing is finished.

column of a row is pointing to tasks in the same queue. As it is shown in Fig.3 B, there is no effect of this emendation on other pointers(OSTCBNext, OSTCBPrev, OSTCBList) pointing to other TCBs. To recognize the position of the task in a queue, OSTCBRR is aggregated to OSTCB. In each queue task is allowed to execute for a constant time, called Quantum, if not preempted by higher priority queue. A constant array Q[] contains quantum of each queue. Position of a task in queue is decided on first come first served basis. For example, if a task1is created first and later task2, then task1 will have first position in queue and task2 will follow it. In _C/OS, each task that is ready to run is placed in a ready list consisting of two variables, OSRdyGrp and OSRdyTbl[]. To determine which priority (and thus which task) will run next, the scheduler determines the lowest priority number that has its bit set in OSRdyTbl[]. The relationship between OSRdyGrp and OSRdyTbl[] is shown in Fig.4 and is given by the following rules[2]:

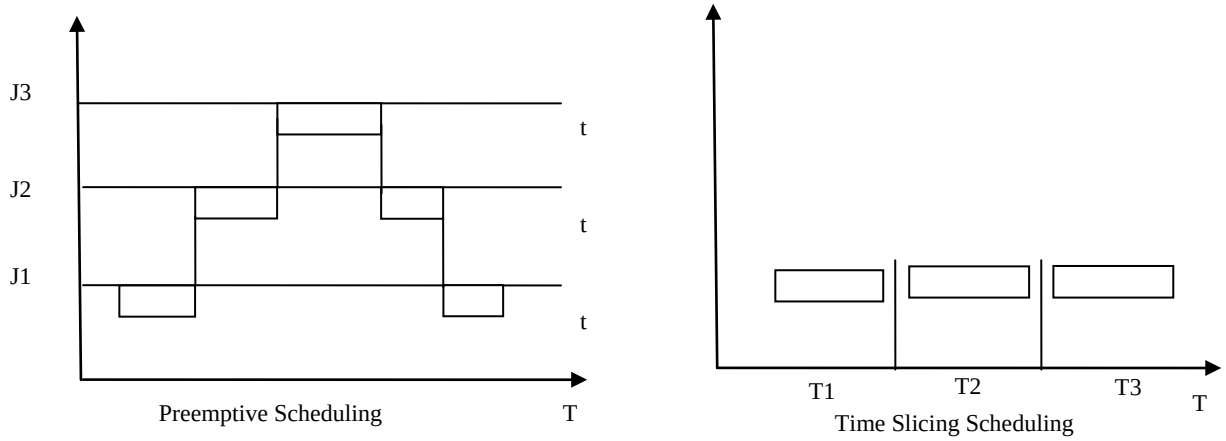


Fig.1. Example of Preemptive and Time-Slice Scheduling

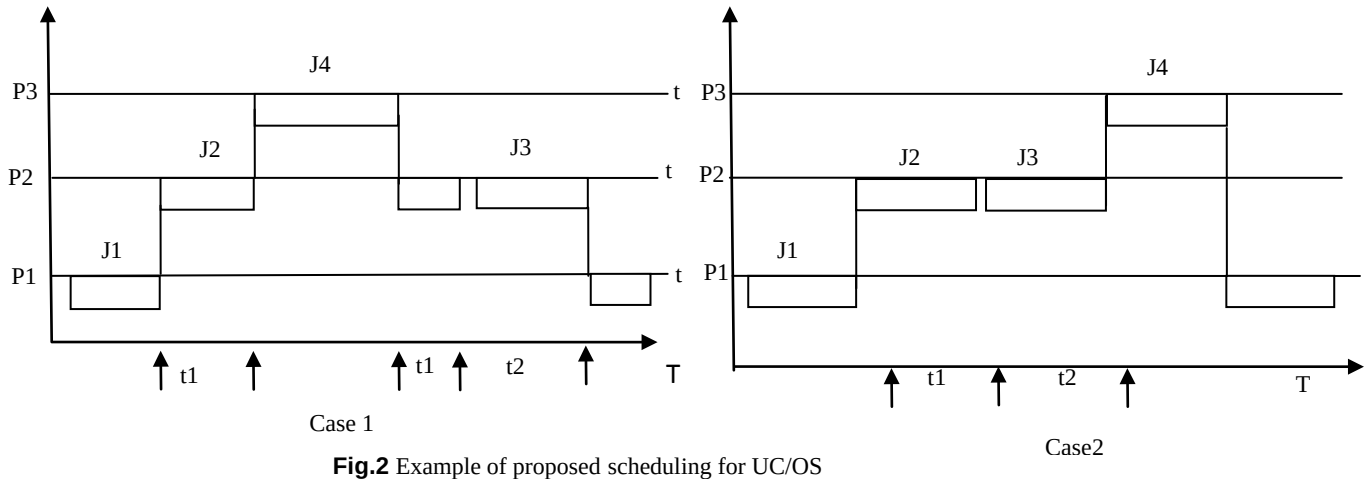


Fig.2 Example of proposed scheduling for UC/OS

4 Implementation

4.1 Task management

A Task Control Block (TCB) is a basic element, but indispensable, of any kernel structure. It is a data structure (OSTCB) which is used to maintain the status of a task when it is preempted. In original kernel, all tasks are located in OSTCBPrioTbl[]. Since tasks are linked as Fig.3 A is depicting, no two task can have same priority. To include one more task at same priority level [1], pointer array priority table is changed to two dimensional array OSTCBPrioTbl[][], in which each

Bit 0 in OSRdyGrp is 1 when any bit in OSRdyTbl [0] is 1.
Bit 1 in OSRdyGrp is 1 when any bit in OSRdyTbl [1] is 1.. Etc.
To make a task ready following code lines are used:

```
OSRdyGrp |= OSMapTbl [prio >> 3];

OSRdyTbl [prio >> 3] |= OSMapTbl [prio & 0x07];
```

But in case of new kernel, it doesn't make a task ready but queue of that particular priority. A Queue is said to ready in

OSRdyGrp and OSRdyTbl [] when at least 1 task of it is ready. To recognize which task of a queue is ready new global variable is introduced: RR[p][r], where p is priority of queue and r is the position which residing in new data member OSTCBRR of OSTCB structure.

4.2 Time management

To implement timer for time-slicing, OSTimeTick, which is called by clock interrupt, at each tick checks every TCB for any OSTimeDly as well as OSTCBTSlice. If OSTimeDly is greater than zero than it decrements the delay and checks if task is suspended or not. If task is suspended it increases the delay by 1. In case of OSTCBTSlice, first it checks whether a task is current task or not. If it is current task then it will decrement the time in OSTCBTSlice. If the time becomes 0, then it reset to quantum value of that queue. Further, if next task is ready then context switching occurs[2]

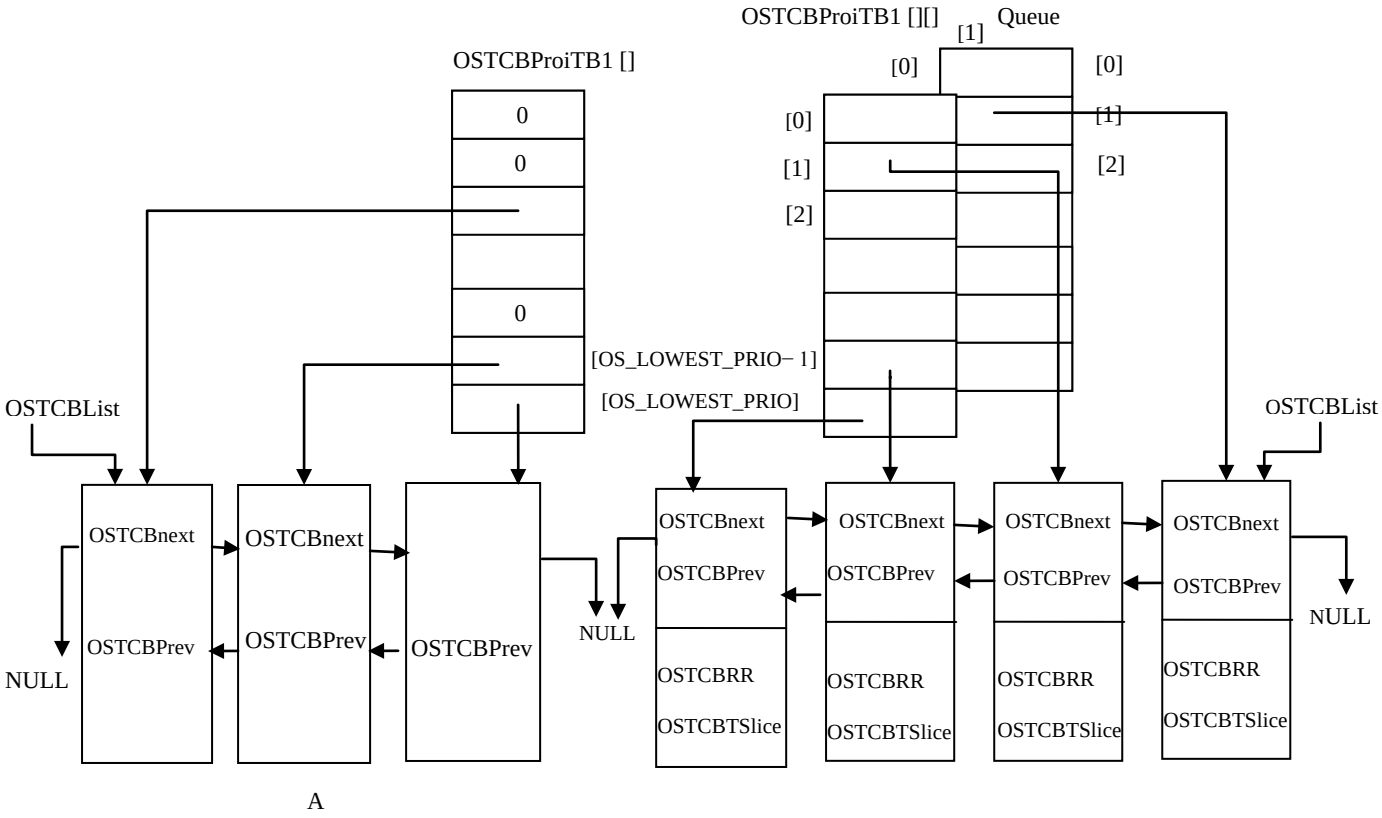


Fig 3: Task management A) Original B) New of *UC/OS*

for pseudo code 4 A 32 bit processor at 50 MHz (13.6 CPI) with LCC Byte code language. It can be observed that the execution time is increased a little bit. The reason being, emendations in system calls is done by adding few data members and constant array instead of redesigning of kernel which might be error prone, complex and require more effort and time.

4.3 Scheduling

In scheduling, tasks are handled with respect to their status (ready, waiting, pending or suspended). Scheduler first checks task scheduling is enabled and not ISR level. If scheduler is not locked then it get the highest priority ready queue. Further, it checks which task in queue is ready. Finally, it checks if highest priority task is current task or not. In case it is not then context switching occurs[3].

5 Evaluations

To write any real time application code, it is necessary to know the overhead of the kernel i.e. execution time for system calls. To evaluate the execution time of new UC/OS, it is ported to Spartan 3 which has X32[6] soft core [4] and provide a Real time platform.

5.1 Experiment and results

In the experiment, only those system calls are tested for execution time which are amended. A system call is tested by calling it through main function and introducing X32 clock to measure execution time in _s. In Table1, results and comparison between original and amended functions are stated. 3 Check appendixes A.2

6 Future Works

This paper has opened up the scope for lot of future work. Only semaphore with essential system calls is changed. Several optional system calls is left unchanged and disabled while evaluating. To change kernel with full features on, these systems calls should be change. An additional feature and also a famous problem, priority inversion protocol, can easily be implemented by doing minor amendment in OSSemPend (a,b,c) and OSSemPost().

7 Conclusions

In this paper, novel scheduler is implemented for _C/OS in order to make it more exible. Modified kernel can execute periodic tasks, which are of same importance; more efficiently. UC/OS has already been modified once in [5] which is focused on implementing priority inheritance semaphore. In [5],
Amendments in other necessary system calls are not stated and others are vague. This paper has explained the details of

necessary amendments. Scalability can still be maintained by using #if def for new instructions. After modification of kernel a negligible change in overhead is observed which signifies that amendment is done prudently. This new kernel will be a new option for those embedded applications in which periodic activities represent the major computational load in a real-time control system.

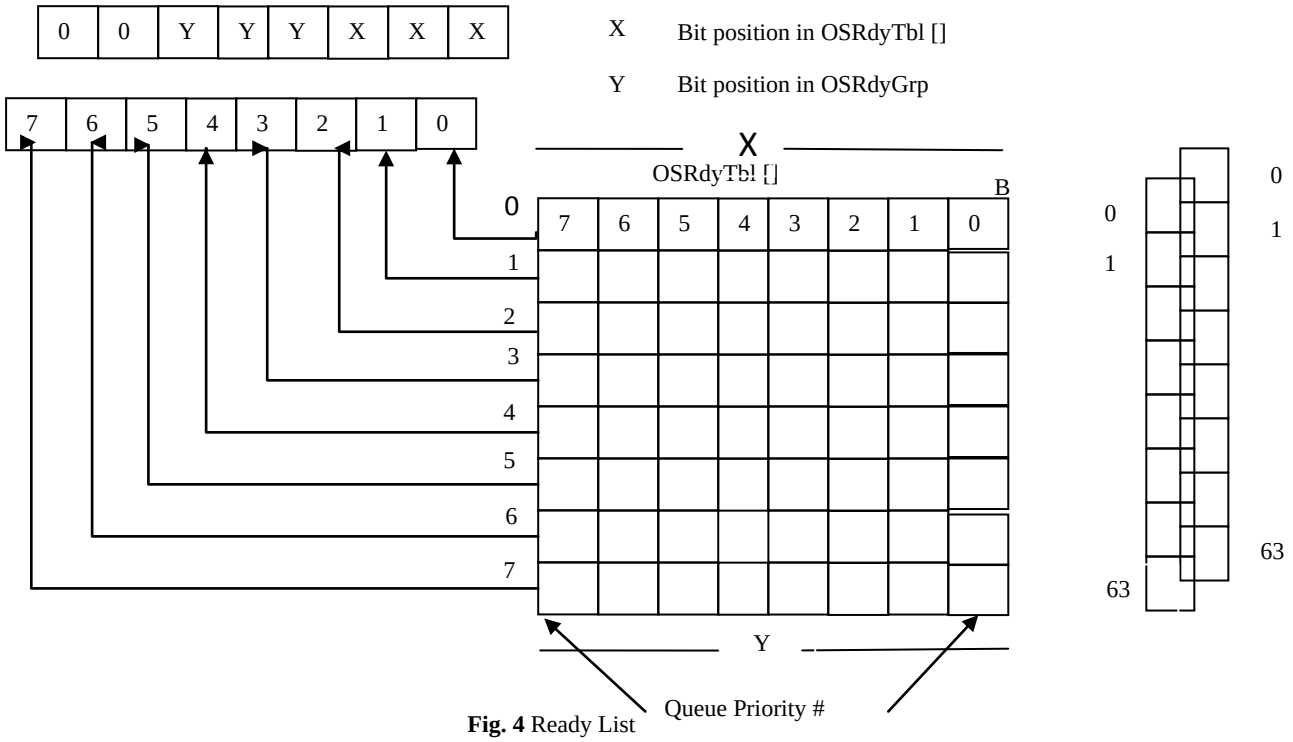
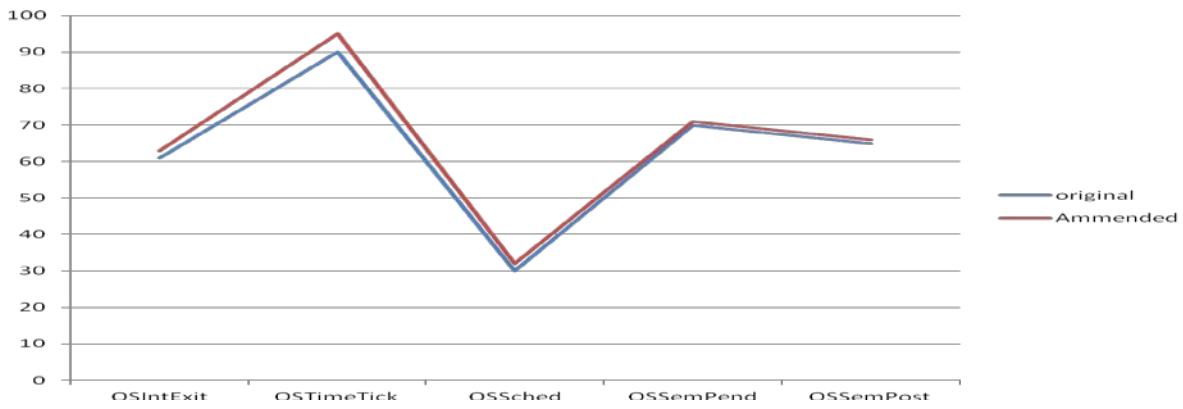


Fig. 4 Ready List

System Cal Original (*US*)	Original (US)	Amended (US)
OSIntExit	61	63
OSTimeTick	90	95
OSSched	30	32
OSSemPend	70	71
OSSemPost	65	66

TablTable 1



Graph 1

References

1. *UC/OS* website, http:*nn*www.ucos.com

2. Jean Labrosse, _*C/OS-II, The Real-Time Kernel*, R D Publications
3. Buttazzo, G. *"Real-Time Operating Systems: Problems and Novel Solutions*\, Springer-Verlag, pp. 37-51, 2002.
4. Buttazzo, G. *Hard Real-time Computing Systems*, Kluwer Academic Publishers, 1997
5. Jae-Ho Lee, Heung-Nam Kim ,*Implementing priority inheritance semaphore on UC/OS real-time kernel* ,IEEE, 2003.
6. http://x32.ewi.tudelft.nl/
7. http://micrium.com/rtos/ucosii
8. http://www.rtos.com/
9. www.keil.com/rl-arm/rtx_rtosadv.asp
10.http://www.engineersgarage.com/articles/rtos-real-time-operating-system