



# Project I for uC/OS-II: EDF Scheduler



# Objective

- To implement an EDF scheduler in uC/OS- II

# Fixed-Priority Scheduling

- uC/OS-II supports fixed-priority scheduling
  - Easy to implement RM
- There is no EDF support
  - Tasks do not have fixed priorities
    - Priorities are fixed at job level
  - Job's “urgency” are determined upon their arrivals
  - Must associate every job with a deadline

# Adding Support for EDF

- Adding Support for EDF Identify where scheduling decisions are made
  - OS\_Sched, OSIntExit(), OSSStart()
- Add proper deadline information to task information (i.e., in TCB)
- Add code to pickup a ready job with the earliest deadline at the re-scheduling points

# Deadlines and Priorities

- Task creation should remain the same for a minimal invasive modification
- Your scheduler will pick up a ready task whose dead line is the earliest
  - Unlike priorities, the value domain of deadlines are infinite
  - But in this project, performance is not a concern; it is okay to use linear search

# Periodic Tasks

```
while(l) {  
    while(OSTCBCur->CompTime > 0)  
    {  
        // do nothing  
    }  
    ...  
    OSTimeDly(...);  
}
```

# Definition

- Implement two sets of periodic tasks.
  - TaskSet 1 = { t1(1,3), t2(3,6) }
  - TaskSet 2 = { t1(1,3), t2(3,6), t3(4,9) }
  - Task arrival times are all at 0
  - Show context switch behaviors
  - Show deadline violations if there is any

# Lessons

- How to create a task that execute exactly  $c$  units of time in every  $p$  units of time?
  - $(c,p)$
- Where in the kernel can we add code for observing the behaviors of context switches?

# Notice

- In real real-time applications, task periods are determined and task invocations are invoked by hardware interrupt
- Task computation time is determined by worst-case computation time analysis (WCET)
- In this project we are to emulate such behavior, and, more importantly, to get insights into how CPU time is allocated to tasks

# Periodic tasks

- Call OSTaskCreate to create a task

```
208 static void TaskStartCreateTasks (void)
209 {
210     OSTaskCreate(Task1, (void *)0, &TaskStk[0][TASK_STK_SIZE - 1], 1);
211     OSTaskCreate(Task2, (void *)0, &TaskStk[1][TASK_STK_SIZE - 1], 2);
212 }
```

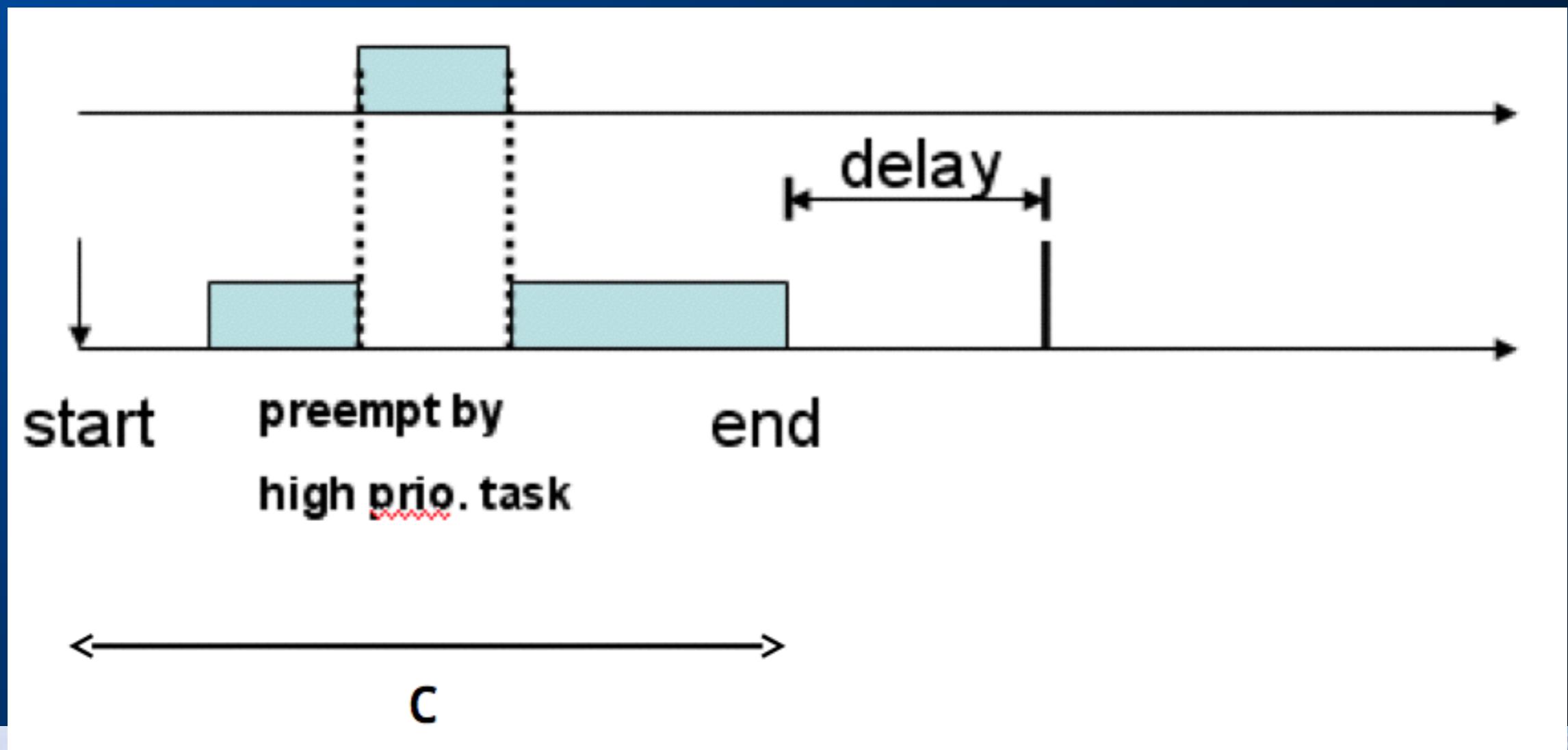
Task function Pointer  
Extra Parameter  
Priority  
Stack Pointer

- A straightforward emulation of (c,p)

```
while(1)
{
    Start=OSTimeGet() ;
    While(OStimeGet()-start < c) ;
    OSTimeDly (p-c) ;
}
```

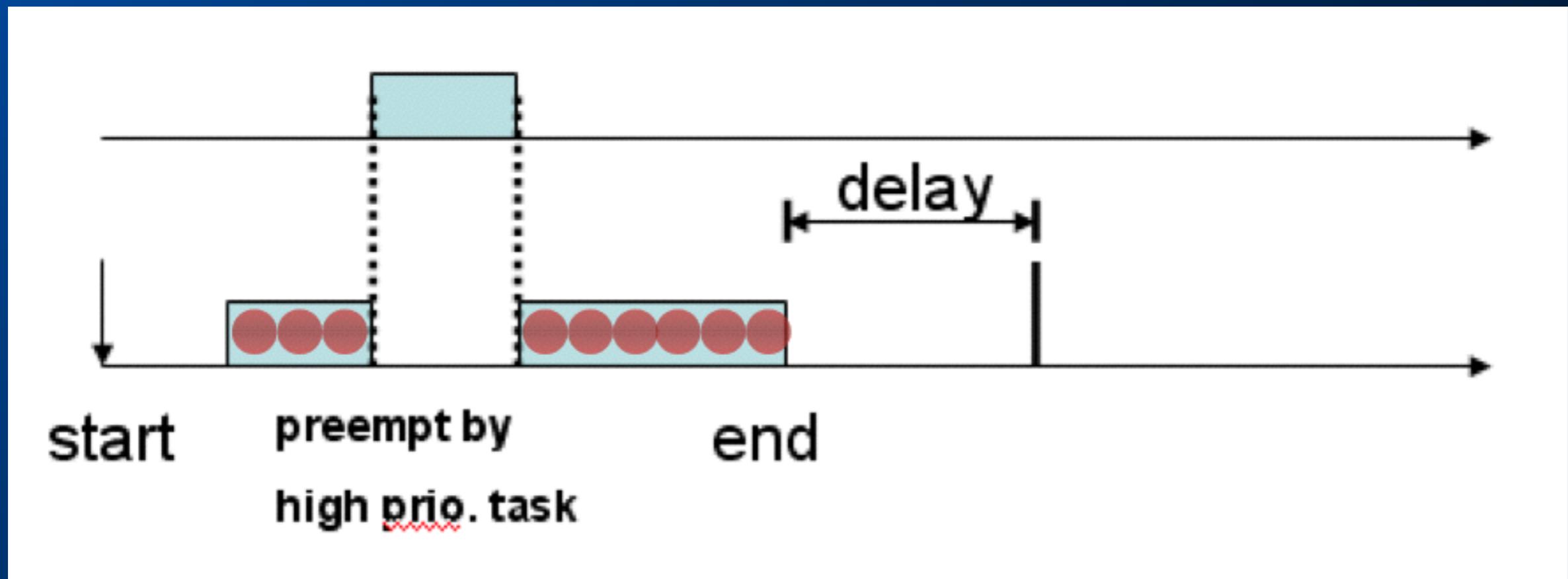
# Periodic task

- Problem : the task did not receive “c” units of CPU time if there is preemption between [start,end]



# Periodic task

- $C$  = clock ticks spend by the task
- $\text{delay} = p - (\text{end}-\text{start})$



# Periodic task

```
void Task()
{
    int start ;//the start time
    int end ;//the end time
    int toDelay;
    start=0 ;
    while(1) {
        while(OSTCBCur->compTime>0) //C ticks
        {
            // do nothing
        }
        end=OSTimeGet() ;// end time
        toDelay=T-(end-start) ;
        start=start+T ;// next start time
        OSTCBCur->compTime=C ;// reset the counter (c ticks for computation)
        OSTimeDly (toDelay); // delay and wait (T-C) times
    }
}
```

Use a counter of residual ticks

# Os\_tcb

- Struct os\_tcb
  - A per-task data structure, defined in uCOS-II.h
  - Add a variable compTime to store the residual clock ticks of a task
    - replenished to “c” at the beginning of every period
  - Add a variable for task’s deadline

# OSTimeTick

- OSTimeTick()
  - Defined in OS\_CORE.C, called every time when a clock interrupt arrives
  - Add a piece of code in OSTimeTick to decrement the compTime counter in os\_tcb
    - The current task has consumed 1 tick

# OSInitExit

- OSInitExit()
  - Defined in OS\_CORE.C
  - This function will manage the scheduling after the system has come back from the calling of ISR
  - We need to print out the “preempt” event here

# OS\_Sched

- OS\_Sched()
  - Defined in OS\_CORE.C
  - OS\_Sched() is called when a task is voluntarily giving up its possession of the CPU
  - We need to print out the “complete” event here

# Printing messages

- Print messages
  - `printf("\n%10d Preempt ",timestamp);`

# Result output

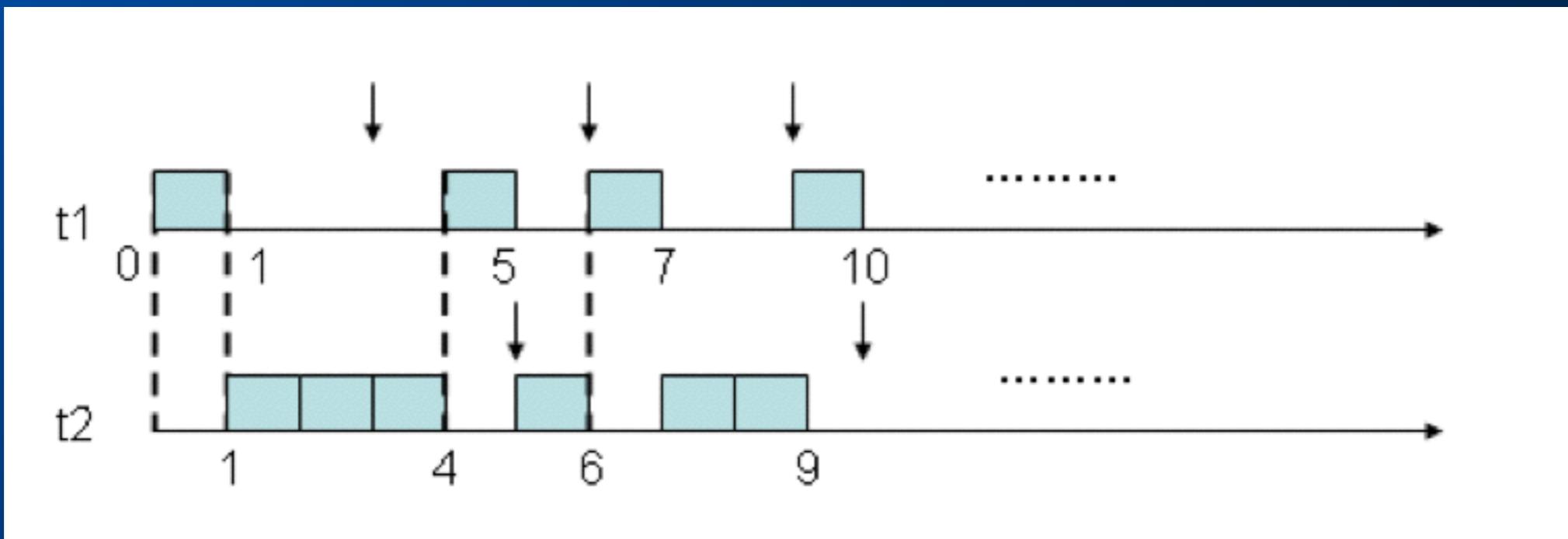
- expected output:

Current time	Event	[From Task ID]	[To Task ID]
Time tick (priority)	Preempt	TaskID(priority)	TaskID

```
6% C:\SOFTWARE\uCOS-II\EX2_x86\ABC45\TEST.EXE
112 Complete      1      2
113 Complete      2      63
114 Preempt       63     1
115 Complete      1      2
117 Preempt       2      1
118 Complete      1      2
119 Complete      2      63
120 Preempt       63     1
121 Complete      1      2
123 Preempt       2      1
124 Complete      1      2
125 Complete      2      63
126 Preempt       63     1
127 Complete      1      2
129 Preempt       2      1
130 Complete      1      2
131 Complete      2      63
132 Preempt       63     1
133 Complete      1      2
135 Preempt       2      1
136 Complete      1      2
137 Complete      2      63
138 Preempt       63     1
139 Complete      1      2
```

# Tasksets for Tests

- Task set1={ t1(1,3) ,t2(3,5) }

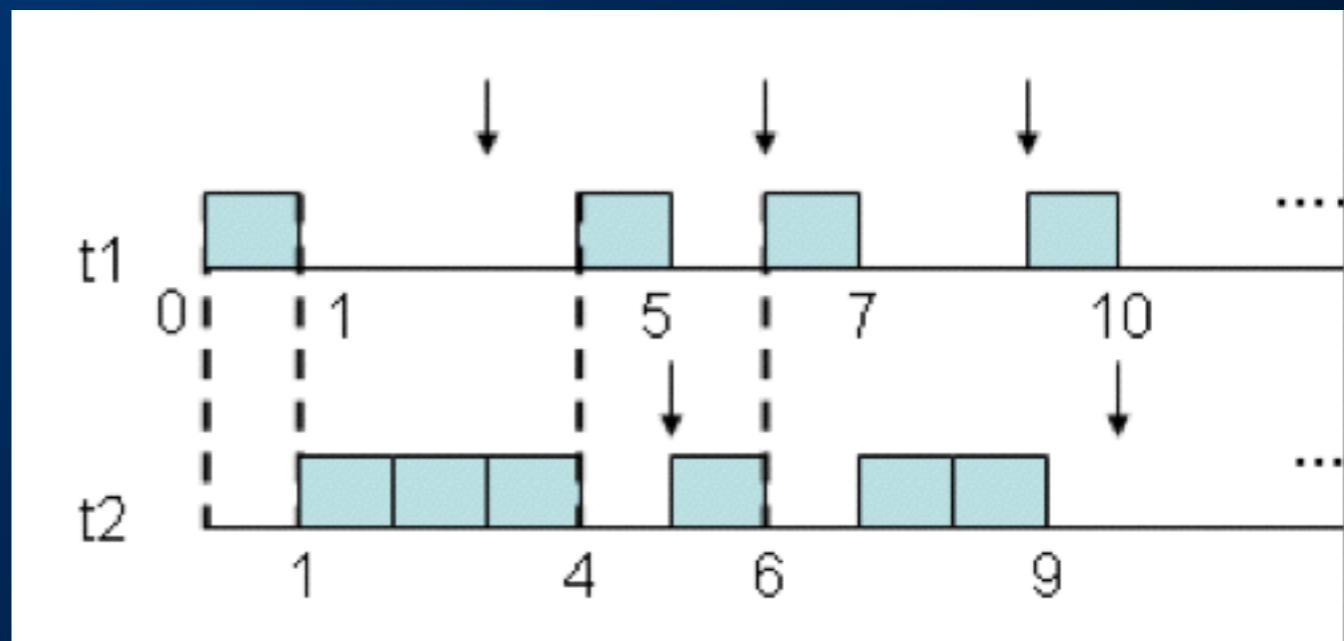


- Task set2={ t1(1,4) ,t2(2,5) ,t3(2,10)}

# Output

- Following the below format

Time	event	from	to
0	Preempt	63	1
1	Complete	1	2
4	Complete	2	1
5	Complete	1	2
6	Preempt	2	1



# Detailed Hints

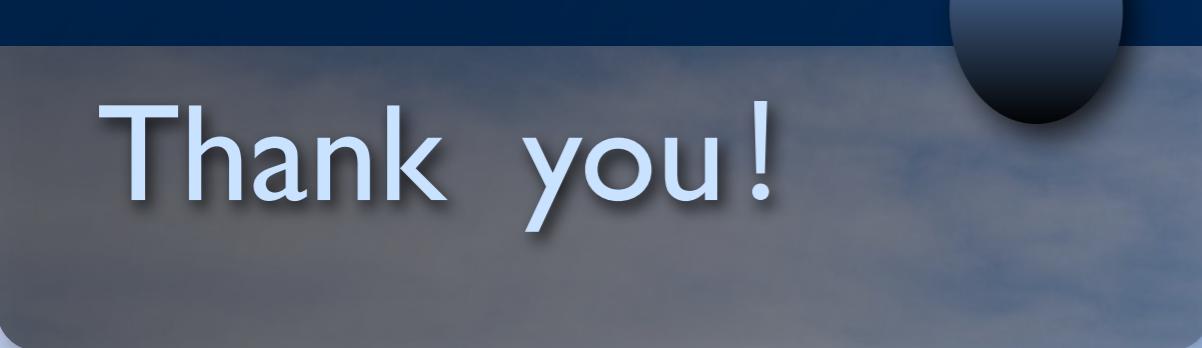
- Add new member in TCB for task deadline information
  - You can pass them to tasks upon creation via the user-provided parameter
- Upon re-scheduling, visit the TCB list linearly; find the ready task whose deadline is the earliest
  - Rescheduling points are OSIntExit, OS\_Sched, OSSStart
- If two or more jobs have same deadline, then they are scheduled in the FIFO manner.

# Bonus!!!

- to implement the improved EDF
- to implement the EDF ucOS on an embedded platform

# 实验 - EDF Scheduler for ucOS-II

- 目标： 在ucOS-II上实现EDF调度。
- 要求：
  - 在pc上的ucOS-II移植版本上实现， 参见Moodle上Lecture Notes下的实验目录uCOSII下的源码。
  - MISRA C 2004 Guidelines for the use of the C language in critical systems
- 提交
  - 提交格式： 学号\_姓名.rar
  - 完整的文档说明实现步骤， 包括所增加的代码与注释。
  - 如果有bonus， 请详细说明，并附上相应参考资料。
  - 为保证公平性， 有bonus的工作要在moodle上相应主题上回复说明，并将相应代码及文档发送给助教。以助教收到邮件时间为准。相同内容只给第一位提交同学加分（2分）。



Thank you!

