Project 1 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(), OSStart()
- 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(1) {
  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

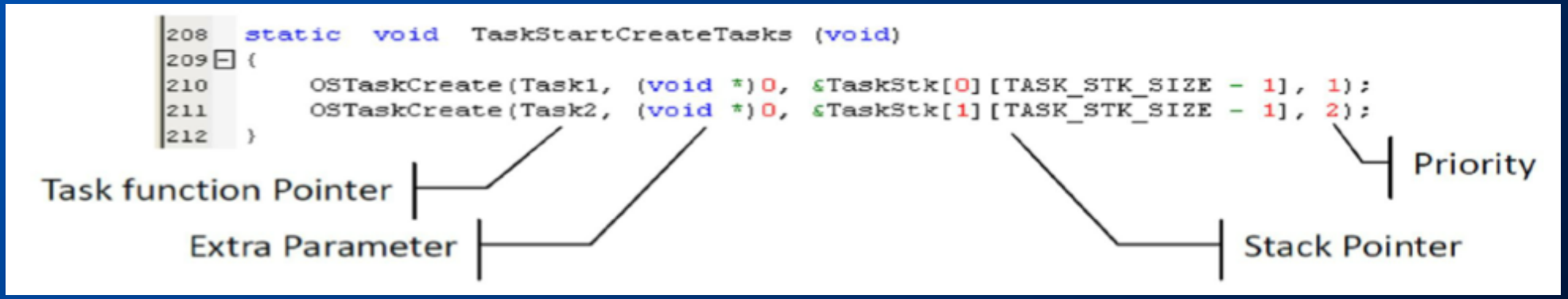
- How to create a task that execute exactly c units of time in every p units of time?
  - (c,p)
- Where in the kernel can we add code for observing the behaviors of context switches?
-

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

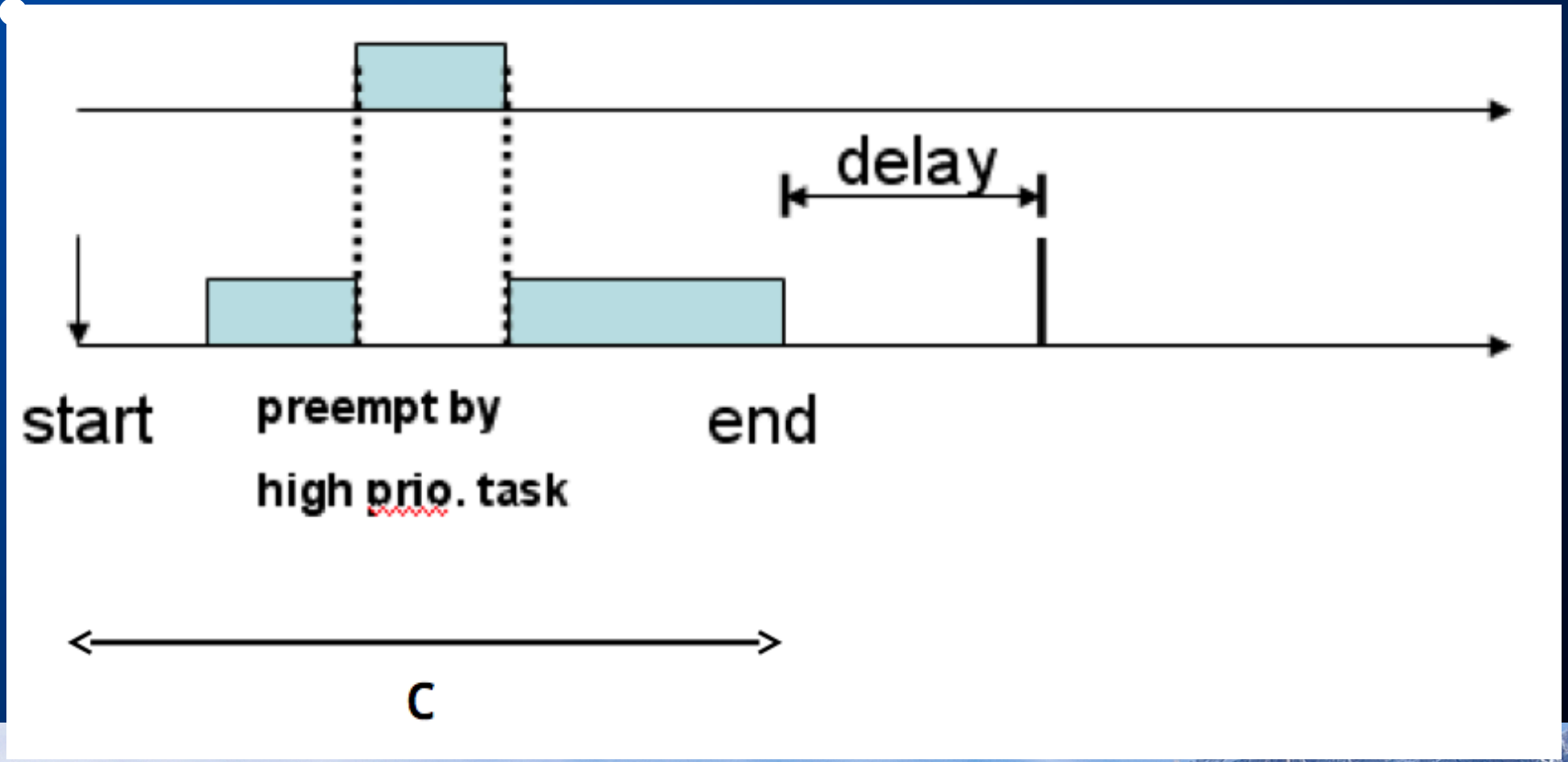


- A straightforward emulation of (c,p)

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

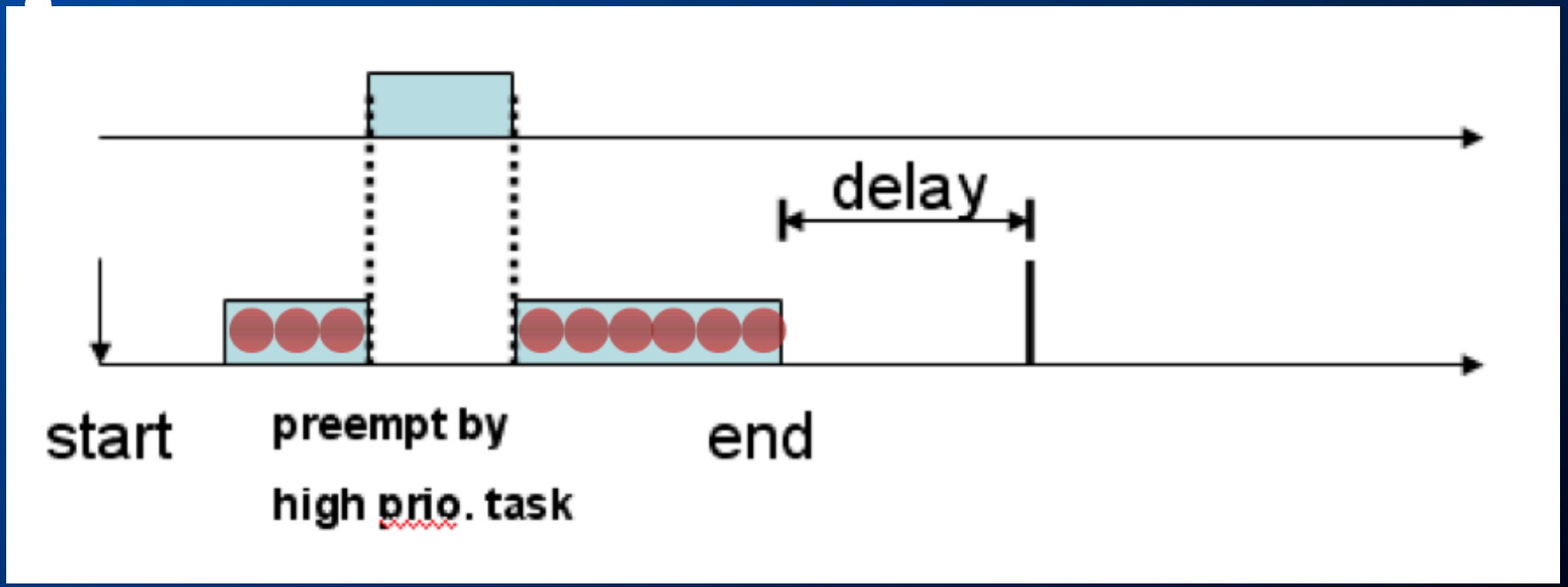
Periodic task

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



Periodic task

- C = clock ticks spend by the task
- delay= p-(end-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 t icks 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

- OSIntExit()
  - Defined in OS_CORE.C
  - This function will manage the scheduling after the sy stem has come back from the calling of ISR
  - We need to print out the “preempt” event here

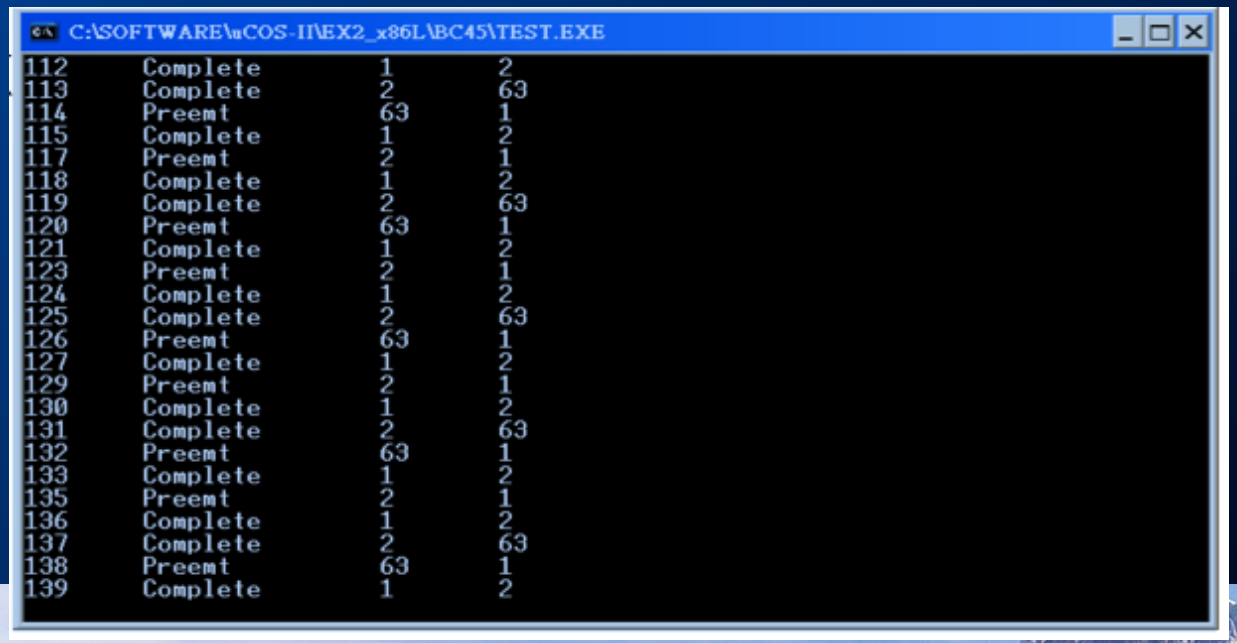
OS_Sched

- 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

Result output

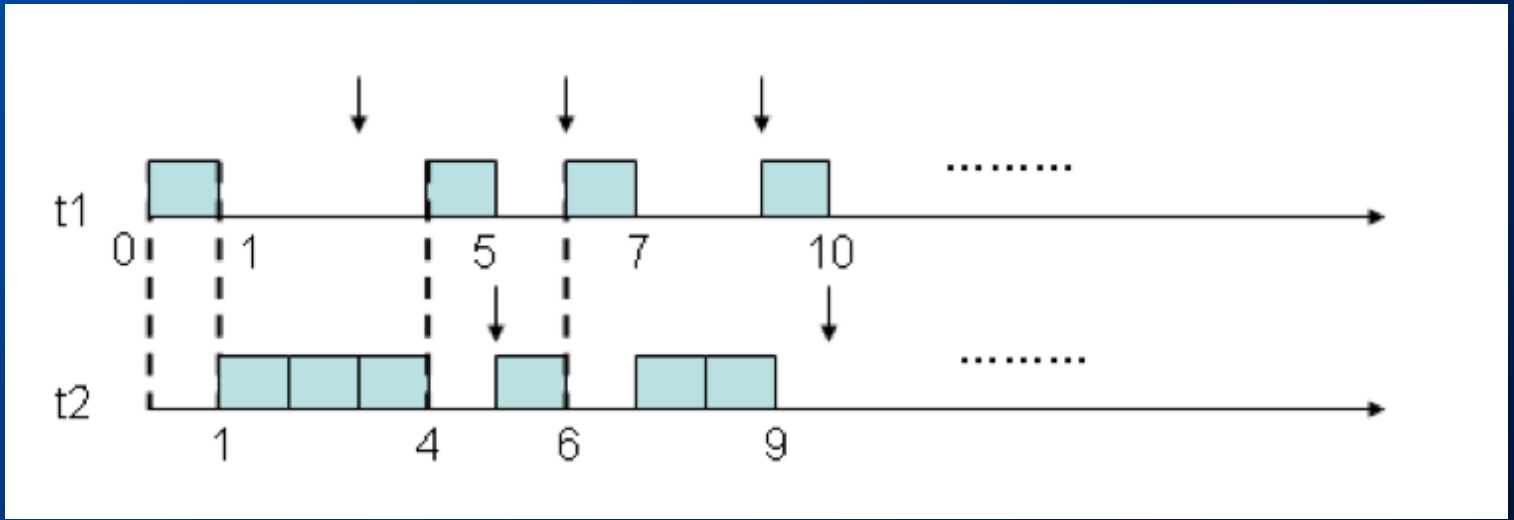
- expected output:

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



Tasksets for Tests

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

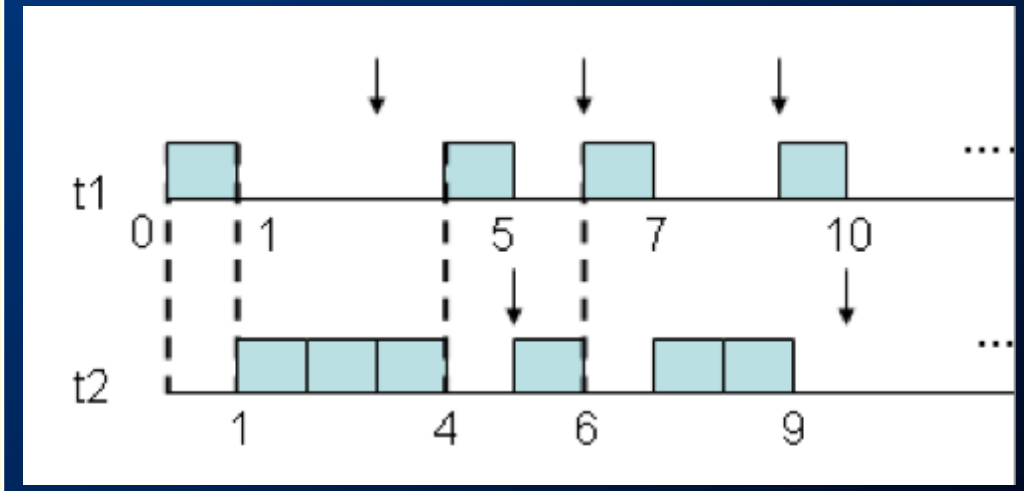


- Task set2={ t1(1,4) , t2(2,5) , t3(2,10)}

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
  - ptcb->OSTCBStat is OS_STAT_RDY?
  - Rescheduling points are OSIntExit, OS_Sched, OSStart
- Before a task delays for the next period, advance its deadline to the next period

Thank you!