RTOS_M11102155_PA2

Name : 楊鏡業

Student ID : M11102155

[Part 1] EDF Scheduler Implementation

Code Review :

The code I modify will wrapped between

```
#ifdef M11102155_PA2_PART_1_EDF

    # modify code !!!

#endif /* M11102155_PA2_PART_1_EDF*/
```

```
#if defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA2_PART_1_EDF)

    # modify code !!!

#endif /* M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF*/
```

ucos_ii.h :

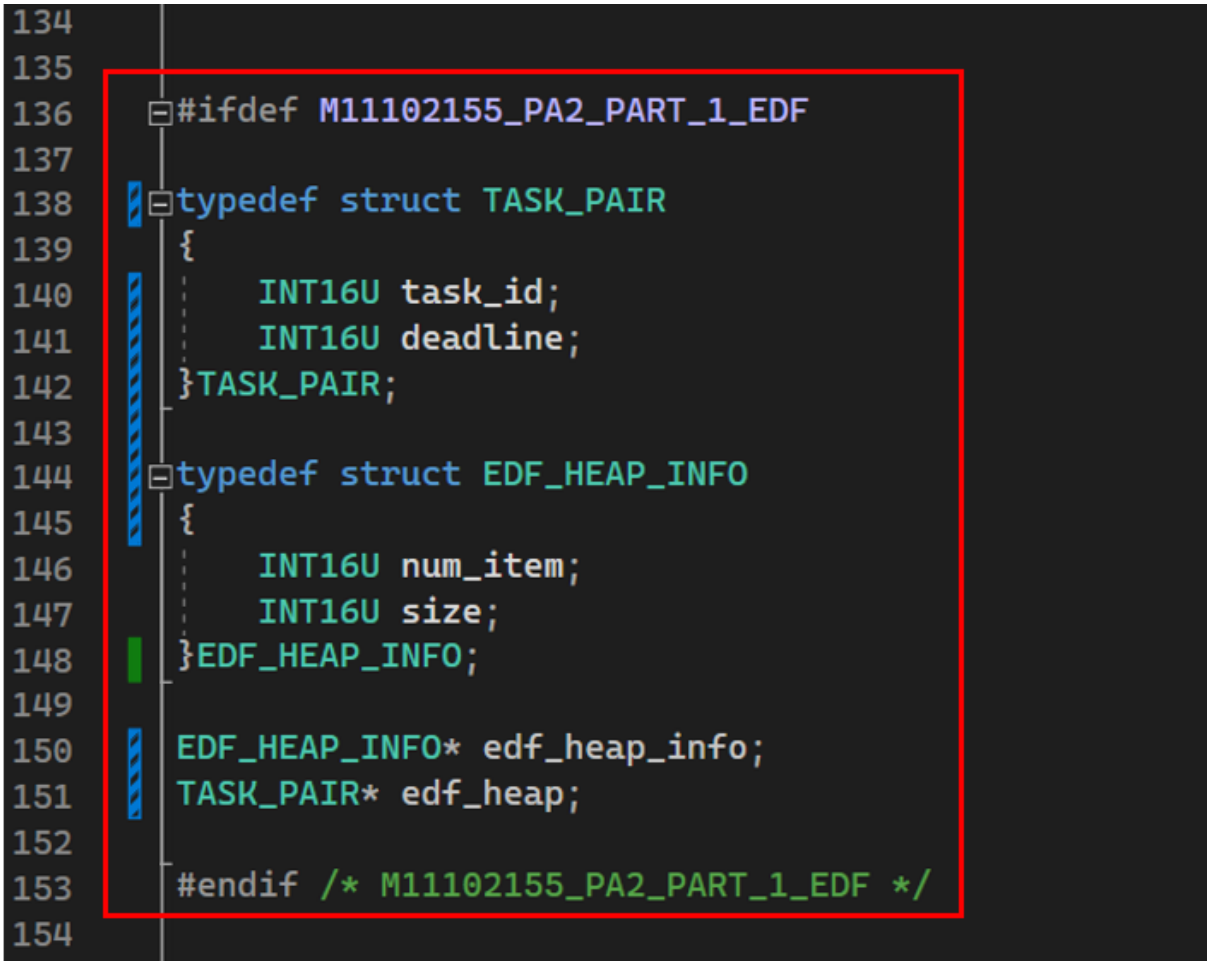
For using #ifdef M11102155_PA2_PART_1_EDF #endif in all code segament.

```
32  #ifndef   OS_uCOS_II_H
33  #define   OS_uCOS_II_H
34
35  //#define M11102155_HW1
36  //#define M11102155_PA1_PART_1
37  //#define M11102155_PA1_PART_2_RM
38  //#define M11102155_PA1_PART_3_FIFO
39  #define M11102155_PA2_PART_1_EDF
40
```

Modify the system end time as 40 ticks and the path of testing dataset.

```
73  /* End time for the simulation */
74  #ifdef M11102155_PA2_PART_1_EDF
75  #define SYSTEM_END_TIME 40
76  #else
77  #define SYSTEM_END_TIME 30
78  #endif /* M11102155_PA2_PART_1_EDF */
79
80
81  /* Input File */
82  FILE* fp;
83
84  #ifdef M11102155_PA2_PART_1_EDF
85  #define INPUT_FILE_NAME "./EDF/TaskSet1.txt"
86  //#define INPUT_FILE_NAME "./TaskSet.txt"
87  #endif /* M11102155_PA2_PART_1_EDF */
88
```

I create a Heap for EDF scheduling, each node of the heap stores the task id and deadline of the task, the heap is minimum heap and the key is deadline. Using EDF_HEAP_INFO for storing the infomations of the heap.



```
134
135
136   #ifdef M11102155_PA2_PART_1_EDF
137
138   typedef struct TASK_PAIR
139   {
140       INT16U task_id;
141       INT16U deadline;
142   }TASK_PAIR;
143
144   typedef struct EDF_HEAP_INFO
145   {
146       INT16U num_item;
147       INT16U size;
148   }EDF_HEAP_INFO;
149
150   EDF_HEAP_INFO* edf_heap_info;
151   TASK_PAIR* edf_heap;
152
153   #endif /* M11102155_PA2_PART_1_EDF */
154
```

app_hooks.c :

I create some operations of EDF minimum heap.

“ EDFHeapSwap() “, is using for swapping two nodes in EDF minimum heap, and “ EDFHeapify() “ is using when a heap delete the top item, task with minimum deadline, after swapping the top item and the bottom item, we must continue comparing with it’s deadlien with child node, and swap if child node has less deadline, keep doing until hit the bottom of the heap.

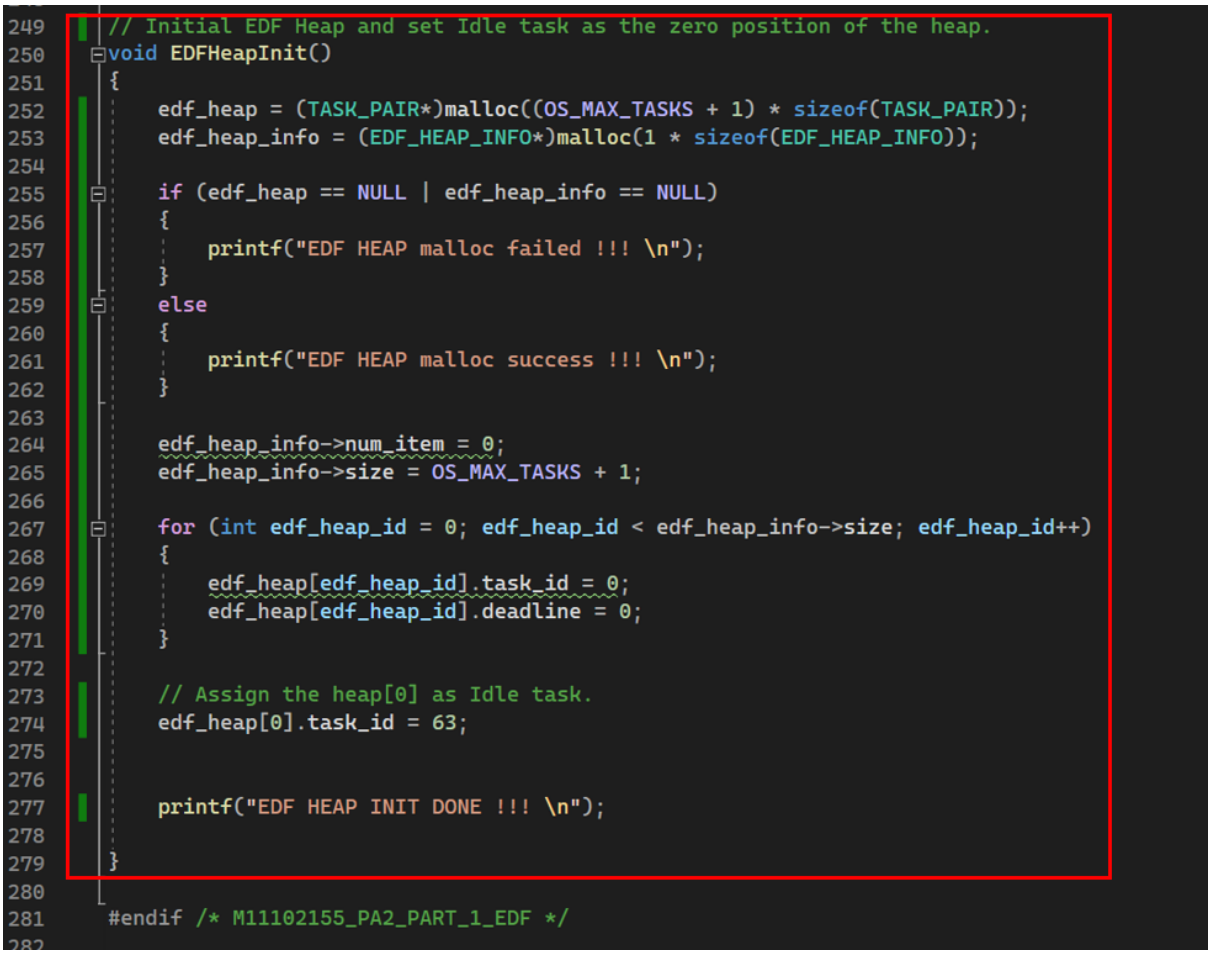
```
177
178
179  #ifdef M11102155_PA2_PART_1_EDF
180
181  // Doing EDF Heap's node swapping.
182  void EDFHeapSwap(TASK_PAIR* a, TASK_PAIR* b)
183  {
184      TASK_PAIR temp = *b;
185      *b = *a;
186      *a = temp;
187  }
188
189  // Doing the Heapify.
190  void EDFHeapify(int location)
191  {
192      int mini = location;
193      int left = (location* 2);
194      int right = left + 1;
195
196      if ((left <= edf_heap_info->num_item) && (edf_heap[left].deadline < edf_heap[mini].deadline))
197      {
198          mini = left;
199      }
200
201      if ((right <= edf_heap_info->num_item) && (edf_heap[right].deadline < edf_heap[mini].deadline))
202      {
203          mini = right;
204      }
205
206      if (mini != location)
207      {
208          EDFHeapSwap(&(edf_heap[location]), &(edf_heap[mini]));
209          EDFHeapify(mini);
210      }
211  }
```

“ EDFHeapDelete() “, is use for deleting the top item of the heap, node with minumum deadline.

“ EDFHeapInsert() “, is use for inserting a new item into the EDF minimum heap, and start to compare the new item with it’s parent nodes, if the deadline is less than it’s parents then swap, keep comparing until it’s dealine is greater than it’s parent or hit the top of the heap.

```
212
213  // Delete the task from EDF Heap.
214  void EDFHeapDelete()
215  {
216      if (edf_heap_info->num_item > 0)
217      {
218          //printf("TICK %d :: delete < %d , %d >\n", OSTime, edf_heap[1].task_id, edf_heap[1].deadline);
219          EDFHeapSwap(&(edf_heap[1]), &(edf_heap[edf_heap_info->num_item]));
220          edf_heap_info->num_item--;
221          EDFHeapify(1);
222      }
223  }
224
225  // Insert the task into EDF Heap (minimum heap order as the deadline).
226  void EDFHeapInsert(INT16U insert_task_id, INT16U insert_task_deadline)
227  {
228      //printf("TICK %d :: insert < %d , %d >\n", OSTime, insert_task_id, insert_task_deadline);
229      edf_heap_info->num_item++;
230      edf_heap[edf_heap_info->num_item].task_id = insert_task_id;
231      edf_heap[edf_heap_info->num_item].deadline = insert_task_deadline;
232
233      INT16U location = edf_heap_info->num_item;
234
235      while ((edf_heap[location].deadline < edf_heap[(location / 2)].deadline) && ((location / 2) > 0))
236      {
237
238          EDFHeapSwap(&(edf_heap[location]), &(edf_heap[(location / 2)]));
239          location = (location / 2);
240      }
241
242      if ((edf_heap[location].deadline == edf_heap[(location / 2)].deadline) && (edf_heap[location].task_id < edf_heap[(location / 2)].task_id) && ((location / 2) > 0))
243      {
244          EDFHeapSwap(&(edf_heap[location]), &(edf_heap[(location / 2)]));
245      }
246
247  }
```

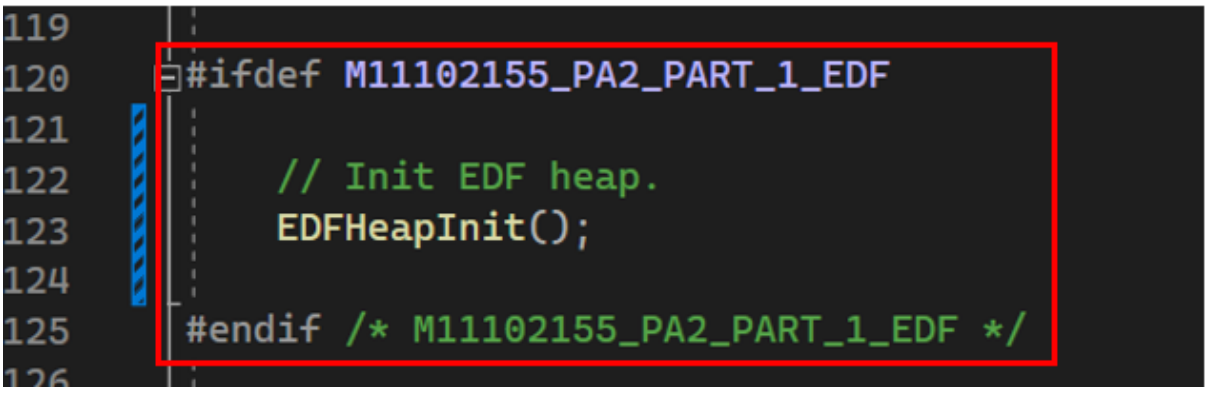
“ EDFHeapInit() “, is using for malloc the EDF minimum heap and initial the value of the nodes in heap, then set the node 0 as the IDLE task, the EDF minimum heap will start as node 1.



```
249 // Initial EDF Heap and set Idle task as the zero position of the heap.
250 void EDFHeapInit()
251 {
252     edf_heap = (TASK_PAIR*)malloc((OS_MAX_TASKS + 1) * sizeof(TASK_PAIR));
253     edf_heap_info = (EDF_HEAP_INFO*)malloc(1 * sizeof(EDF_HEAP_INFO));
254
255     if (edf_heap == NULL | edf_heap_info == NULL)
256     {
257         printf("EDF HEAP malloc failed !!! \n");
258     }
259     else
260     {
261         printf("EDF HEAP malloc success !!! \n");
262     }
263
264     edf_heap_info->num_item = 0;
265     edf_heap_info->size = OS_MAX_TASKS + 1;
266
267     for (int edf_heap_id = 0; edf_heap_id < edf_heap_info->size; edf_heap_id++)
268     {
269         edf_heap[edf_heap_id].task_id = 0;
270         edf_heap[edf_heap_id].deadline = 0;
271     }
272
273     // Assign the heap[0] as Idle task.
274     edf_heap[0].task_id = 63;
275
276
277     printf("EDF HEAP INIT DONE !!! \n");
278
279 }
280
281 #endif /* M11102155_PA2_PART_1_EDF */
```

main.c :

First, we create a EDF minimum heap.



```
119
120 #ifdef M11102155_PA2_PART_1_EDF
121
122     // Init EDF heap.
123     EDFHeapInit();
124
125 #endif /* M11102155_PA2_PART_1_EDF */
126
```

AS we create the task, if the task arrive at beginning, then just insert into EDF minimum heap.

```
184     // Set task ID as priority.
185 #ifdef M11102155_PA2_PART_1_EDF
186
187     for (n = 0; n < TASK_NUMBER; n++)
188     {
189         OSTaskCreateExt(task,
190             &TaskParameter[n],
191             &Task_STK[n][TASK_STAKSIZE - 1],
192             TaskParameter[n].TaskID,
193             TaskParameter[n].TaskID,
194             &Task_STK[n][0],
195             TASK_STAKSIZE,
196             &TaskParameter[0],
197             (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR));
198     }
199
200
201     printf(" after task create :: \n");
202     for (int edf_heap_id = 1; edf_heap_id <= edf_heap_info->num_item; edf_heap_id++)
203     {
204         printf("< %d, %d >\n", edf_heap[edf_heap_id].task_id, edf_heap[edf_heap_id].deadline);
205     }
206
207
208 #endif /* M11102155_PA2_PART_1_EDF */
```

```
235 #if defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA2_PART_1_EDF)
236     printf("Tick \t  Event \t CurrentTask ID \t NextTask ID \t ResponseTime \t PreemptionTime \t OSTimeDly\n");
237 #endif /* M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF */
```

In “ task() ”, first initial the count of the task has executed and the deadline of the first job, then start to periodically do task.

In the inner while loop the task will keep executing until occur interrupt “ Tick ISR ” at the end of each tick, and the count of “ num_recent_execute_time “, means the count of task has executed in current job, will increase by one and consider whether the amount is greater than the total executed time (task finish executing this job) or not. If is, it will break the while loop and do the delay, the delay time will be compute in the OSTimeDly() function. On the other hand, if is not, task will keep doing this job until finish.

Here is a special case, in Task Set 2 at tick 11, the current job of task2 is done but Tick interrupt occur earlier than OSTimeDly(), the task 3 preempt task 2 before task 2 do delay, so task 2 will do delay after task 3 finish it’s job, here i conquer this issue by checking whether this case happen or not in Tick ISR. After adding the condition, this case’s task (e.g. task 2) will be complete (delay) during Tick ISR before preempted by task 3. Futhermore I reset the “ num_recent_execute_time “ before the task be ready, so when task 2 become ready it will still stuck at inner loop and

doing the test, consider this job is done or not, won't repeat doing the " OSTimeDly() ".

```
#if defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA2_PART_1_EDF)

static void task(void* p_arg)
{
    task_para_set* task_date;
    task_date = p_arg;

    while (1)
    {
        // For every task keep executing until finish or preemptive
        while (OSTCBCur->num_recent_execute_time < OSTCBCur->total_execute_time);    // TimeTick interrupt happen before while loop end !!!

        OSTimeDly(0);
    }
}

#endif /* M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF */
```

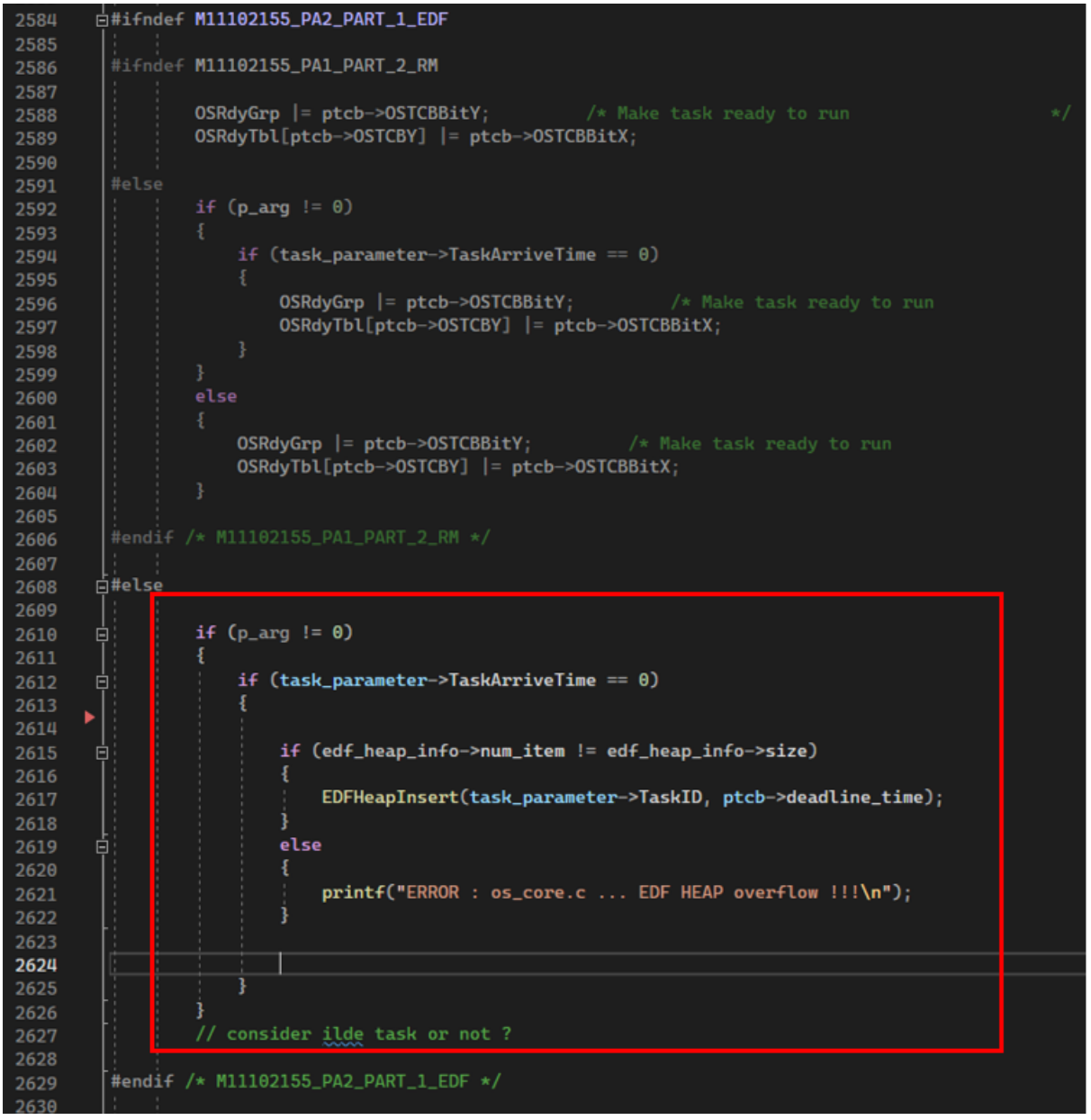
os_time.c :

In " os_time.c ", as the task call " OSTimeDly() ", the task should be delete from EDF minimum heap, so I call EDFHeapDelete() to remove the top item of the heap, thenI cancel the operation of setting the ready table.

```

void  OSTimeDly (INT32U ticks)
{
    INT8U      y;
#if OS_CRITICAL_METHOD == 3u                 /* Allocate storage for CPU status register           */
    OS_CPU_SR  cpu_sr = 0u;
#endif

#if defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA2_PART_1_EDF)

    // Compute the delay time before task arrive again.
    OS_ENTER_CRITICAL();
    EDFHeapDelete();            // Remove the task from heap.
    ticks = OSTCBCur->deadline_time - OSTime;
    OSTCBCur->response_time = OSTime - OSTCBCur->arrive_time;
    OS_EXIT_CRITICAL();
#endif /* M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF */



    if (OSIntNesting > 0u) {                 /* See if trying to call from an ISR                  */
        return;
    }
    if (OSLockNesting > 0u) {                /* See if called with scheduler locked                */
        return;
    }
    if (ticks > 0u) {                        /* 0 means no delay!                                  */
        OS_ENTER_CRITICAL();

#ifndef M11102155_PA2_PART_1_EDF
        y             =  OSTCBCur->OSTCBY;   /* Delay current task                                 */
        OSRdyTbl[y] &= (OS_PRIO)~OSTCBCur->OSTCBBitX;
        //OS_TRACE_TASK_SUSPENDED(OSTCBCur);
        if (OSRdyTbl[y] == 0u) {
            OSRdyGrp &= (OS_PRIO)~OSTCBCur->OSTCBBitY;
        }
#endif /* M11102155_PA2_PART_1_EDF */

        OS_TRACE_TASK_SUSPENDED(OSTCBCur);
        OSTCBCur->OSTCBDly = ticks;          /* Load ticks in TCB                                  */
        OS_TRACE_TASK_DLY(ticks);
        OS_EXIT_CRITICAL();
        OS_Sched();                          /* Find next task to run!                             */
    }
}
```

os_core.c :

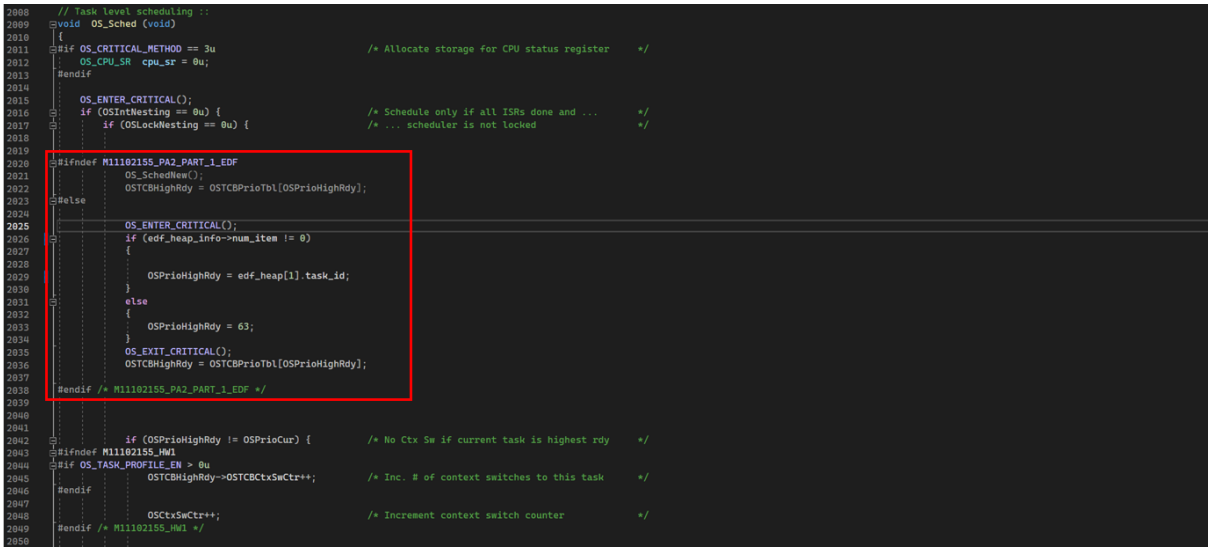
In “ os_core.c ”, I first modify “ OSTCBInit() ”, if a task’s arrive not equals to 0, then insert task into EDF minimum heap.



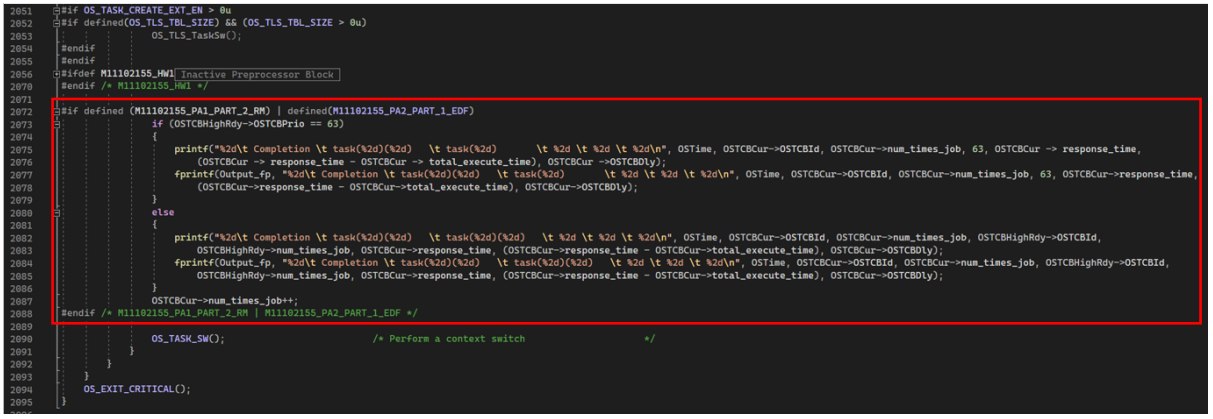
```
#ifndef M11102155_PA2_PART_1_EDF

#ifndef M11102155_PA1_PART_2_RM

    OSRdyGrp |= ptcb->OSTCBBitY;          /* Make task ready to run                */
    OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;

#else
    if (p_arg != 0)
    {
        if (task_parameter->TaskArriveTime == 0)
        {
            OSRdyGrp |= ptcb->OSTCBBitY;          /* Make task ready to run
            OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
        }
    }
    else
    {
        OSRdyGrp |= ptcb->OSTCBBitY;          /* Make task ready to run
        OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
    }

#endif /* M11102155_PA1_PART_2_RM */

#else

    if (p_arg != 0)
    {
        if (task_parameter->TaskArriveTime == 0)
        {

            if (edf_heap_info->num_item != edf_heap_info->size)
            {
                EDFHeapInsert(task_parameter->TaskID, ptcb->deadline_time);
            }
            else
            {
                printf("ERROR : os_core.c ... EDF HEAP overflow !!!\n");
            }

        }
    }
    // consider ilde task or not ?

#endif /* M11102155_PA2_PART_1_EDF */
```

Second I modify task level context switch, “ OS_Scehd() ”, the selected task should has the least deadline, so I just get the task id of the top, first item, of EDF minimum

heap, and set it as the the next executing task. If the heap is empty, just execute Idle task.

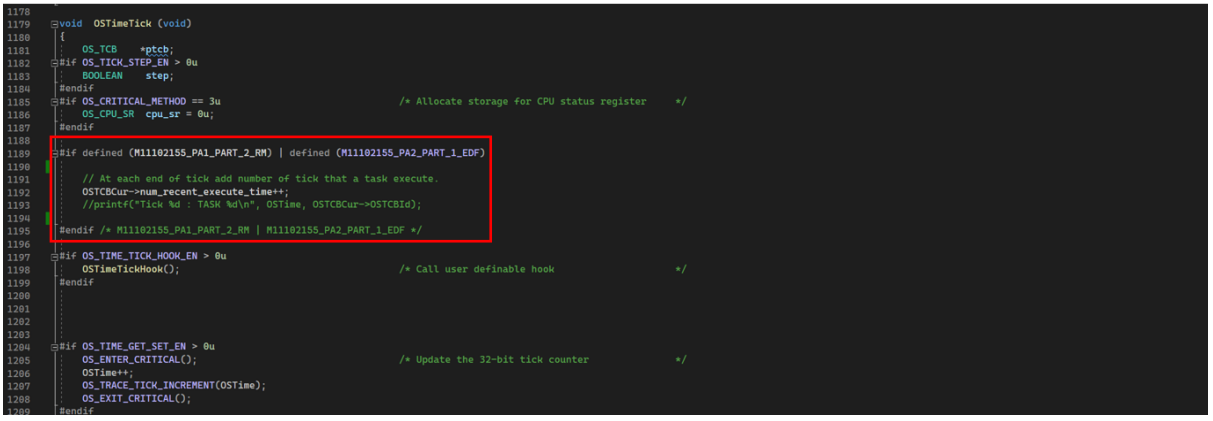


```
// Task level scheduling ::
void  OS_Sched (void)
{
#if OS_CRITICAL_METHOD == 3u                       /* Allocate storage for CPU status register    */
    OS_CPU_SR  cpu_sr = 0u;
#endif

    OS_ENTER_CRITICAL();
    if (OSIntNesting == 0u) {                      /* Schedule only if all ISRs done and ...      */
        if (OSLockNesting == 0u) {                 /* ... scheduler is not locked                 */


#ifndef M11102155_PA2_PART_1_EDF
            OS_SchedNew();
            OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
#else

            OS_ENTER_CRITICAL();
            if (edf_heap_info->num_item != 0)
            {

                OSPrioHighRdy = edf_heap[1].task_id;
            }
            else
            {
                OSPrioHighRdy = 63;
            }
            OS_EXIT_CRITICAL();
            OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];

#endif /* M11102155_PA2_PART_1_EDF */


            if (OSPrioHighRdy != OSPrioCur) {          /* No Ctx Sw if current task is highest rdy    */
#ifndef M11102155_HW1
#if OS_TASK_PROFILE_EN > 0u
                OSTCBHighRdy->OSTCBCtxSwCtr++;         /* Inc. # of context switches to this task     */
#endif

                OSCtxSwCtr++;                          /* Increment context switch counter            */
#endif /* M11102155_HW1 */
```

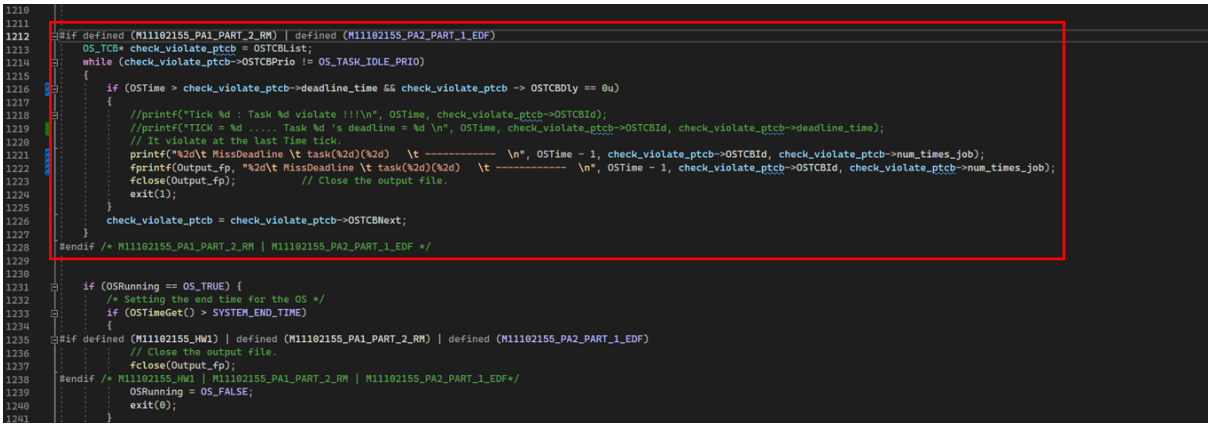


```
#if OS_TASK_CREATE_EXT_EN > 0u
#if defined(OS_TLS_TBL_SIZE) && (OS_TLS_TBL_SIZE > 0u)
                OS_TLS_TaskSw();
#endif
#endif
#ifdef M11102155_HW1 Inactive Preprocessor Block
#endif /* M11102155_HW1 */


#if defined (M11102155_PA1_PART_2_RM) | defined(M11102155_PA2_PART_1_EDF)
                if (OSTCBHighRdy->OSTCBPrio == 63)
                {
                    printf("%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)       \t %2d \t %2d \t %2d\n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63, OSTCBCur -> response_time,
                        (OSTCBCur -> response_time - OSTCBCur -> total_execute_time), OSTCBCur ->OSTCBDly);
                    fprintf(Output_fp, "%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)       \t %2d \t %2d \t %2d\n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63, OSTCBCur->response_time,
                        (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBDly);
                }
                else
                {
                    printf("%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)(%2d)   \t %2d \t %2d \t %2d\n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, OSTCBHighRdy->OSTCBId,
                        OSTCBHighRdy->num_times_job, OSTCBCur->response_time, (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBDly);
                    fprintf(Output_fp, "%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)(%2d)   \t %2d \t %2d \t %2d\n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, OSTCBHighRdy->OSTCBId,
                        OSTCBHighRdy->num_times_job, OSTCBCur->response_time, (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBDly);
                }
                OSTCBCur->num_times_job++;
#endif /* M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF */

                OS_TASK_SW();                          /* Perform a context switch                    */
            }
        }
    }
    OS_EXIT_CRITICAL();
}
```

Then I modify “ OSTimeTick() “, that occur at the end in every tick, first increase the current task’s executing tick, then consider whether exist task violate. If exist then end RTOS else keep running.

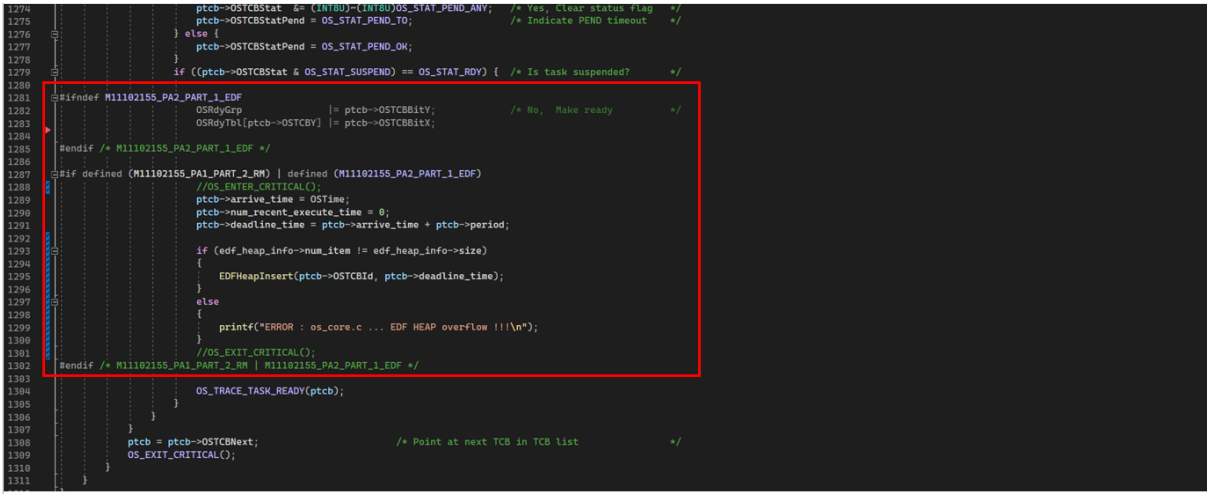


```

void  OSTimeTick (void)
{
    OS_TCB    *ptcb;
#if OS_TICK_STEP_EN > 0u
    BOOLEAN    step;
#endif
#if OS_CRITICAL_METHOD == 3u                       /* Allocate storage for CPU status register    */
    OS_CPU_SR  cpu_sr = 0u;
#endif

#if defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA2_PART_1_EDF)

    // At each end of tick add number of tick that a task execute.
    OSTCBCur->num_recent_execute_time++;
    //printf("Tick %d : TASK %d\n", OSTime, OSTCBCur->OSTCBId);

#endif /* M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF */

#if OS_TIME_TICK_HOOK_EN > 0u
    OSTimeTickHook();                              /* Call user definable hook                    */
#endif




#if OS_TIME_GET_SET_EN > 0u
    OS_ENTER_CRITICAL();                           /* Update the 32-bit tick counter              */
    OSTime++;
    OS_TRACE_TICK_INCREMENT(OSTime);
    OS_EXIT_CRITICAL();
#endif
```



```
#if defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA2_PART_1_EDF)
    OS_TCB* check_violate_ptcb = OSTCBList;
    while (check_violate_ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO)
    {
        if (OSTime > check_violate_ptcb->deadline_time && check_violate_ptcb -> OSTCBDly == 0u)
        {
            //printf("Tick %d : Task %d violate !!!\n", OSTime, check_violate_ptcb->OSTCBId);
            //printf("TICK = %d ..... Task %d 's deadline = %d \n", OSTime, check_violate_ptcb->OSTCBId, check_violate_ptcb->deadline_time);
            // It violate at the last Time tick.
            printf("%2d\t MissDeadline \t task(%2d)(%2d)   \t ------------  \n", OSTime - 1, check_violate_ptcb->OSTCBId, check_violate_ptcb->num_times_job);
            fprintf(Output_fp, "%2d\t MissDeadline \t task(%2d)(%2d)   \t ------------  \n", OSTime - 1, check_violate_ptcb->OSTCBId, check_violate_ptcb->num_times_job);
            fclose(Output_fp);          // Close the output file.
            exit(1);
        }
        check_violate_ptcb = check_violate_ptcb->OSTCBNext;
    }
#endif /* M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF */


    if (OSRunning == OS_TRUE) {
        /* Setting the end time for the OS */
        if (OSTimeGet() > SYSTEM_END_TIME)
        {
#if defined (M11102155_HW1) | defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA2_PART_1_EDF)
            // Close the output file.
            fclose(Output_fp);
#endif /* M11102155_HW1 | M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF*/
            OSRunning = OS_FALSE;
            exit(0);
        }
```

AS, the task finish delay, compute the Informations that record in each task’s TCB, using for each task’s scheduking, then insert the task into EDF heap set it as ready.



```
                    ptcb->OSTCBStat  &= (INT8U)~(INT8U)OS_STAT_PEND_ANY;   /* Yes, Clear status flag   */
                    ptcb->OSTCBStatPend = OS_STAT_PEND_TO;                 /* Indicate PEND timeout    */
                } else {
                    ptcb->OSTCBStatPend = OS_STAT_PEND_OK;
                }
                if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) == OS_STAT_RDY) {  /* Is task suspended?       */

#ifndef M11102155_PA2_PART_1_EDF
                    OSRdyGrp               |= ptcb->OSTCBBitY;             /* No,  Make ready          */
                    OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;

#endif /* M11102155_PA2_PART_1_EDF */

#if defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA2_PART_1_EDF)
                    //OS_ENTER_CRITICAL();
                    ptcb->arrive_time = OSTime;
                    ptcb->num_recent_execute_time = 0;
                    ptcb->deadline_time = ptcb->arrive_time + ptcb->period;

                    if (edf_heap_info->num_item != edf_heap_info->size)
                    {
                        EDFHeapInsert(ptcb->OSTCBId, ptcb->deadline_time);
                    }
                    else
                    {
                        printf("ERROR : os_core.c ... EDF HEAP overflow !!!\n");
                    }
                    //OS_EXIT_CRITICAL();
#endif /* M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF */

                    OS_TRACE_TASK_READY(ptcb);
                }
            }
        }
        ptcb = ptcb->OSTCBNext;                    /* Point at next TCB in TCB list              */
        OS_EXIT_CRITICAL();
    }
}
```

In “ OSIntExit() “, consider the interrupt level context switch, there exist some cases. First the EDF minimum heap is empy, there is no ready task, set the next task into Idle task. Second the current task just complete but arrive immerdiatly, so we must delete the previous task and recompute the recording in TCB later and find the next task with least deadline. Thrid case is the current task complete and isn’t arrive immediately, so the current task will fully execute “ OSTimeDly() “.

```
694   // Interrupt (Kernel) level scheduling ::
695   void  OSIntExit (void)
696   {
697   #if OS_CRITICAL_METHOD == 3u                          /* Allocate storage for CPU status register */
698       OS_CPU_SR  cpu_sr = 0u;
699   #endif
700
701       if (OSRunning == OS_TRUE) {
702           OS_ENTER_CRITICAL();
703           if (OSIntNesting > 0u) {                      /* Prevent OSIntNesting from wrapping       */
704               OSIntNesting--;
705           }
706           if (OSIntNesting == 0u) {                     /* Reschedule only if all ISRs complete ... */
707               if (OSLockNesting == 0u) {                /* ... and not locked.                      */
708
709   #ifndef M11102155_PA2_PART_1_EDF
710                   OS_SchedNew();
711                   OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
712   #else
713                   OS_ENTER_CRITICAL();
714                   if (edf_heap_info -> num_item == 0)         // there is no task in the waiting queue, execute the Idle task
715                   {
716                       if (OSTCBCur->num_recent_execute_time == OSTCBCur->total_execute_time)
717                       {
718                           OSPrioHighRdy = 63;
719                           OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
720                       }
721                   }
722                   else
723                   {
724                       if (OSTCBCur->num_recent_execute_time == OSTCBCur->total_execute_time)
725                       {
726                           EDFHeapDelete();
727                           OSPrioHighRdy = edf_heap[1].task_id;
728                           OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
729                       }
730                       else
731                       {
732                           OSPrioHighRdy = edf_heap[1].task_id;
733                           OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
734                       }
735                   }
736                   OS_EXIT_CRITICAL();
737
738   #endif /* M11102155_PA2_PART_1_EDF */
```

```
737
738   #endif /* M11102155_PA2_PART_1_EDF */
739
740                   if (OSPrioHighRdy != OSPrioCur) {         /* No Ctx Sw if current task is highest rdy */
741
742   #ifndef M11102155_HW1
743   #if OS_TASK_PROFILE_EN > 0u
744                       OSTCBHighRdy->OSTCBCtxSwCtr++;        /* Inc. # of context switches to this task  */
745   #endif
746                       OSCtxSwCtr++;                         /* Keep track of the number of ctx switches */
747   #endif /* M11102155_HW1 */
748
749   #if OS_TASK_CREATE_EXT_EN > 0u
750   #if defined(OS_TLS_TBL_SIZE) && (OS_TLS_TBL_SIZE > 0u)
751                       OS_TLS_TaskSw();
752   #endif
753   #endif
754                       OS_TRACE_ISR_EXIT_TO_SCHEDULER();
755
756   #ifdef M11102155_HW1 Inactive Preprocessor Block
761   #endif /* M11102155_HW1 */
762
763   #ifdef M11102155_PA1_PART_2_RM Inactive Preprocessor Block
822   #endif /* M11102155_PA1_PART_2_RM */
823
```

Here we consider about the second case that we talk about, the current won't execute " OSTimeDly() ", so we must compute the infomations that recording in TCB fisrt and insert the task if the current arrvie immediatley.

```
763   #ifdef M11102155_PA1_PART_2_RM Inactive Preprocessor Block
822   #endif /* M11102155_PA1_PART_2_RM */
823
824   #ifdef M11102155_PA2_PART_1_EDF
825               if (OSTCBCur->num_recent_execute_time == OSTCBCur->total_execute_time)
826               {
827                   // Compute response time.
828                   OSTCBCur->response_time = OSTime - OSTCBCur->arrive_time;
829
830                   // Do delay !!!
831                   INT32U ticks = OSTCBCur->deadline_time - OSTime;
832                   //printf("TICK %d    TASK %d's delay = %d\n", OSTime, OSTCBCur->OSTCBId, ticks);
833                   if (ticks > 0u) {                         /* 0 means no delay!                        */
834                       OS_ENTER_CRITICAL();
835                       OS_TRACE_TASK_SUSPENDED(OSTCBCur);
836                       OSTCBCur->OSTCBDly = ticks;           /* Load ticks in TCB                        */
837                       OS_TRACE_TASK_DLY(ticks);
838                       OS_EXIT_CRITICAL();
839                   }
840                   else
841                   {
842                       // If the task arrive after it complete immediately.
843                       //printf("TICK %d hahaha TASK %d \n", OSTime, OSTCBCur->OSTCBId);
844                       OS_ENTER_CRITICAL();
845                       OSTCBCur->arrive_time = OSTime;
846                       OSTCBCur->num_recent_execute_time = 0;
847                       OSTCBCur->deadline_time = OSTCBCur->arrive_time + OSTCBCur->period;
848
849                       if (edf_heap_info->num_item != edf_heap_info->size)
850                       {
851                           EDFHeapInsert(OSTCBCur->OSTCBId, OSTCBCur->deadline_time);
852                       }
853                       else
854                       {
855                           printf("ERROR : os_core.c ... EDF HEAP overflow !!!\n");
856                       }
857
858                       OS_EXIT_CRITICAL();
859                   }
860
861                   if (OSTCBHighRdy->OSTCBPrio == 63)
862                   {
863                       printf("%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)       \t %2d \t %2d \t %2d  \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63, OSTCBCur->response_time,
864                           (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBDly);
865                       fprintf(Output_fp, "%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)       \t %2d \t %2d \t %2d  \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63, OSTCBCur->response_time,
866                           (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBDly);
867                       OSTCBCur->num_times_job++;
868                   }
869                   else
870                   {
871                       printf("%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)(%2d)       \t %2d \t %2d \t %2d  \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->num_times_job, OSTCBCur->response_time,
872                           (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBDly);
873                       fprintf(Output_fp, "%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)(%2d)       \t %2d \t %2d \t %2d  \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->num_times_job, OSTCBCur->response_time
874                           (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBDly);
875                       OSTCBCur->num_times_job++;
876                   }
877               }
```

[Part 2] CUS Scheduler Implementation

Continuing from the previous answer from EDF, I just keep the EDF scheduler for Part 2 CUS scheduler.

Code Review :

The code I modify will wrapped between

```
#ifdef M11102155_PA2_PART_2_CUS

    # modify code !!!

#endif /* M11102155_PA2_PART_2_CUS*/
```

ucos_ii.h :

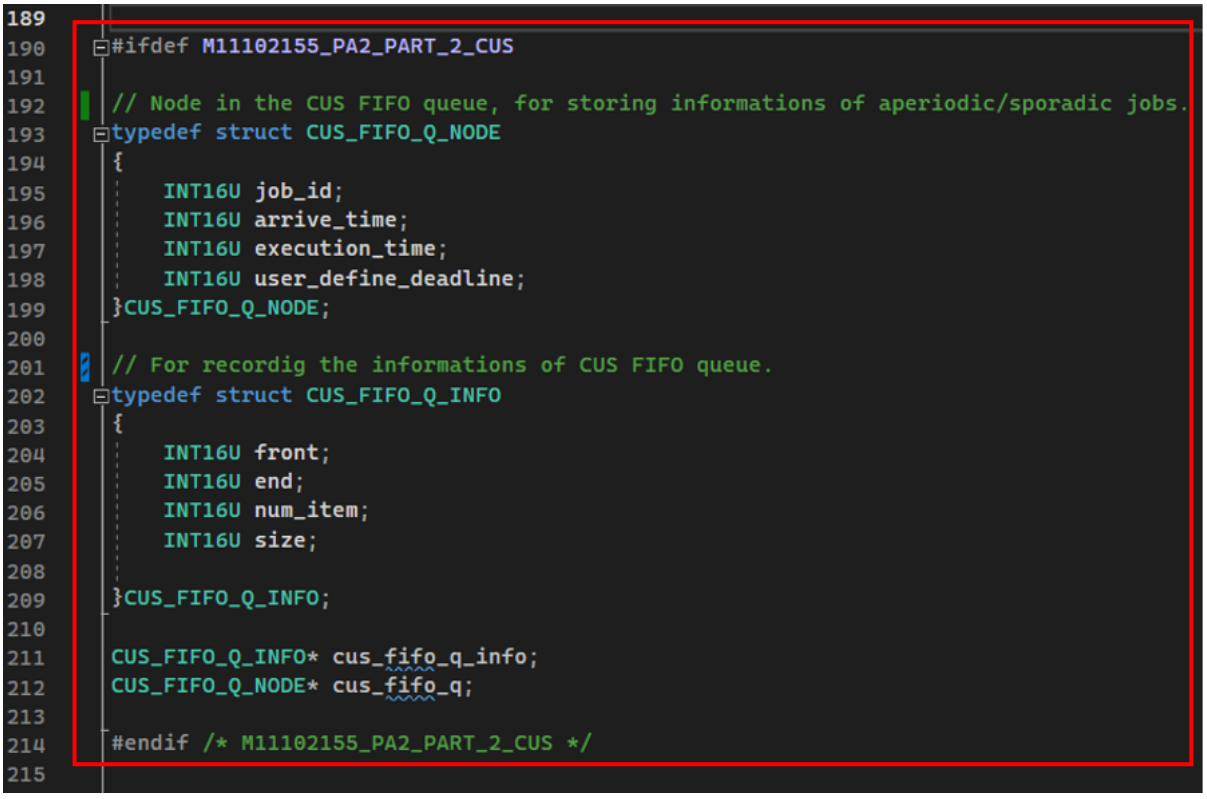
In “ ucos_ii.h ”, I continue using EDF scheduler as I implement in previous part, and modify the code by using M11102155_PA2_PART_2_CUS as I defined.

```
31
32      #ifndef   OS_uCOS_II_H
33      #define   OS_uCOS_II_H
34
35      //#define M11102155_HW1
36      //#define M11102155_PA1_PART_1
37      //#define M11102155_PA1_PART_2_RM
38      //#define M11102155_PA1_PART_3_FIFO
39      #define M11102155_PA2_PART_1_EDF
40      #define M11102155_PA2_PART_2_CUS
41
```

Here I modify the number of input file as two, fisrt is same as previous part, input file including periodic tasks (including CUS), second file is for aperiodic/sporadic jobs.

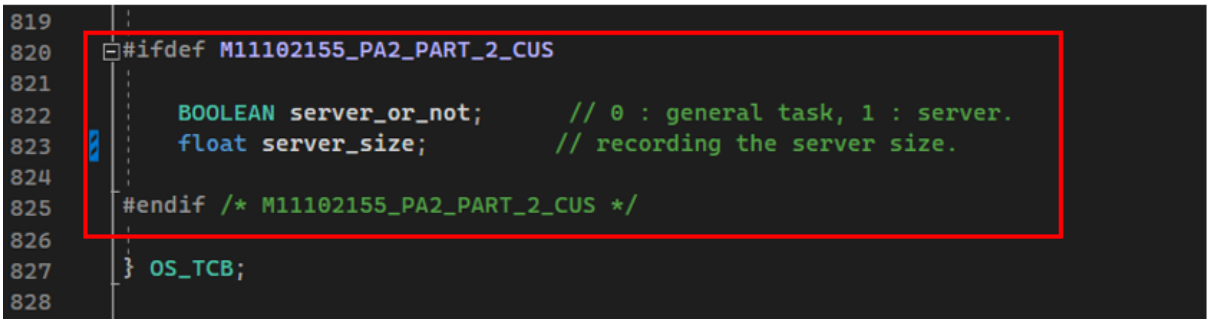
```
96
97      #ifndef M11102155_PA2_PART_2_CUS
98
99      #ifdef M11102155_PA2_PART_1_EDF
100     #define INPUT_FILE_NAME "./EDF/TaskSet1.txt"
101     //#define INPUT_FILE_NAME "./TaskSet.txt"
102     #endif /* M11102155_PA2_PART_1_EDF */
103     #else
104
105     #define CUS_INTPUT_FILE_NAME "./CUS/AperiodicJobs.txt"
106     #define INPUT_FILE_NAME "./CUS/TaskSet.txt"
107
108     #endif /* M11102155_PA2_PART_2_CUS */
109
```

I create a FIFO queue for storing aperiodic/spradic job, this queue will be used for CUS



```
#ifdef M11102155_PA2_PART_2_CUS

// Node in the CUS FIFO queue, for storing informations of aperiodic/sporadic jobs.
typedef struct CUS_FIFO_Q_NODE
{
    INT16U job_id;
    INT16U arrive_time;
    INT16U execution_time;
    INT16U user_define_deadline;
}CUS_FIFO_Q_NODE;

// For recordig the informations of CUS FIFO queue.
typedef struct CUS_FIFO_Q_INFO
{
    INT16U front;
    INT16U end;
    INT16U num_item;
    INT16U size;

}CUS_FIFO_Q_INFO;

CUS_FIFO_Q_INFO* cus_fifo_q_info;
CUS_FIFO_Q_NODE* cus_fifo_q;

#endif /* M11102155_PA2_PART_2_CUS */
```

In struct “ OS_TCB ”, I add a flag to consider whether this task is server task(CUS) or general periodic tasks. As you can see, when server_or_not equals to 1, means it is server task(CUS), otherwise it is general periodic tasks. The server_size records the server size of the sercer task(CUS).

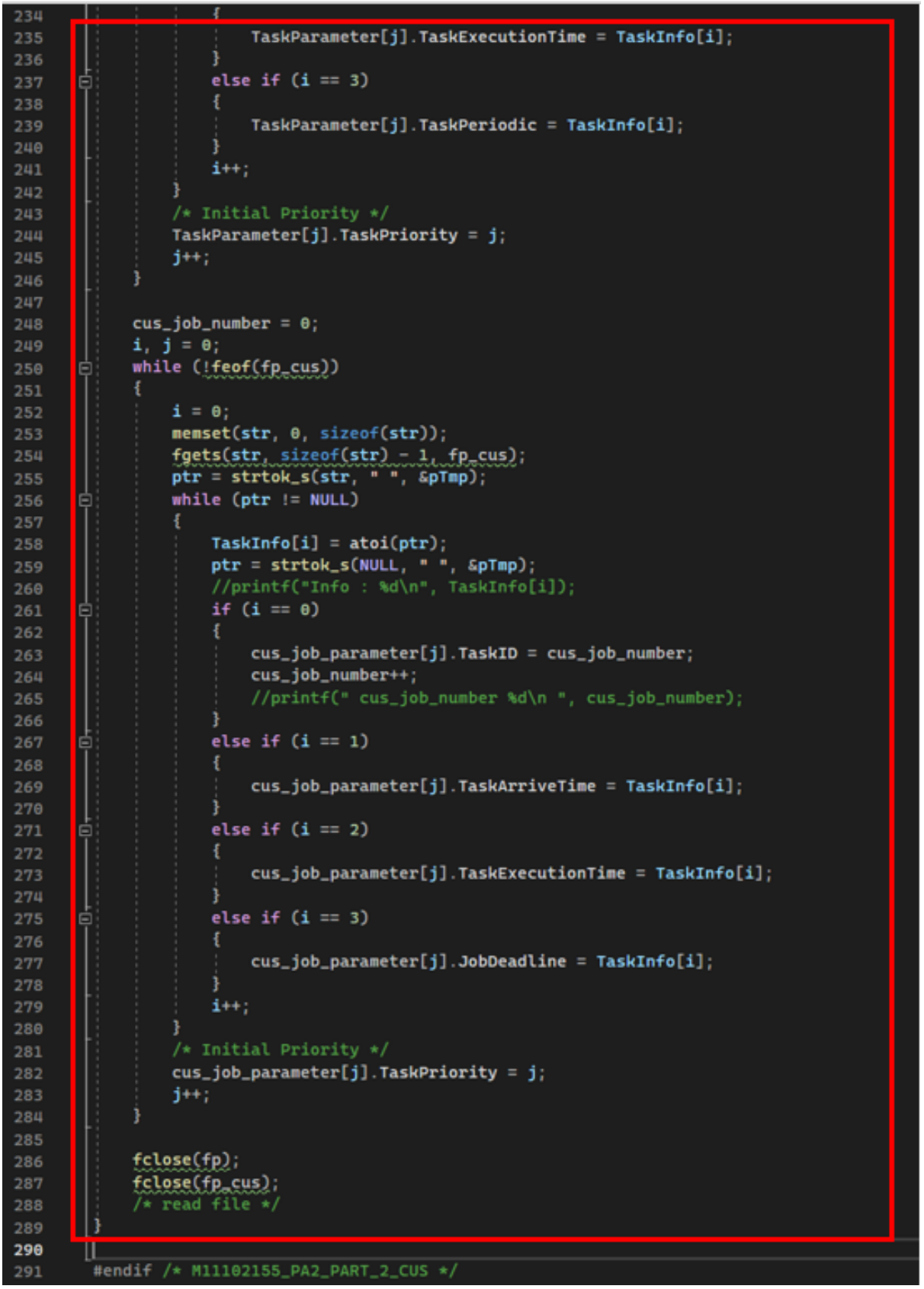


```
#ifdef M11102155_PA2_PART_2_CUS

    BOOLEAN server_or_not;      // 0 : general task, 1 : server.
    float server_size;          // recording the server size.

#endif /* M11102155_PA2_PART_2_CUS */

} OS_TCB;
```

app_hooks.c :

In “ app_hook.c ”, I cancel InputFile() in previous part, I rewrite InputFile(), for reading two intput file, including periodic and aperiodic.

```
112
113 #ifndef M11102155_PA2_PART_2_CUS Inactive Preprocessor Block
179 #else
180 void InputFile()
181 {
182     /*
183     *  Read file
184     *  Task Information :
185     *  Task_ID ArriveTime ExecutionTime Periodic
186     */
187
188     errno_t err;
189     if ((err = fopen_s(&fp, INPUT_FILE_NAME, "r")) == 0)
190     {
191         printf("The file 'TaskSet.txt' was opened\n");
192     }
193     else
194     {
195         printf("The file 'TaskSet.txt' was not opened\n");
196     }
197
198     if ((err = fopen_s(&fp_cus, CUS_INTPUT_FILE_NAME, "r")) == 0)
199     {
200         printf("The file 'AperiodicJobs.txt' was opened\n");
201     }
202     else
203     {
204         printf("The file 'AperiodicJobs.txt' was not opened\n");
205     }
206
207     char str[MAX];
208     char* ptr;
209     char* pTmp = NULL;
210     int TaskInfo[INFO], i, j = 0;
211     TASK_NUMBER = 0;
212
213     while (!feof(fp))
214     {
215         i = 0;
216         memset(str, 0, sizeof(str));
217         fgets(str, sizeof(str) - 1, fp);
218         ptr = strtok_s(str, " ", &pTmp);
219         while (ptr != NULL)
220         {
221             TaskInfo[i] = atoi(ptr);
222             ptr = strtok_s(NULL, " ", &pTmp);
223             //printf("Info : %d\n", TaskInfo[i]);
224             if (i == 0)
225             {
226                 TASK_NUMBER++;
227                 TaskParameter[j].TaskID = TASK_NUMBER;
228             }
229             else if (i == 1)
230             {
231                 TaskParameter[j].TaskArriveTime = TaskInfo[i];
232             }
233             else if (i == 2)
234             {
```



```
234                 {
235                     TaskParameter[j].TaskExecutionTime = TaskInfo[i];
236                 }
237                 else if (i == 3)
238                 {
239                     TaskParameter[j].TaskPeriodic = TaskInfo[i];
240                 }
241                 i++;
242             }
243             /* Initial Priority */
244             TaskParameter[j].TaskPriority = j;
245             j++;
246         }
247 
248         cus_job_number = 0;
249         i, j = 0;
250         while (!feof(fp_cus))
251         {
252             i = 0;
253             memset(str, 0, sizeof(str));
254             fgets(str, sizeof(str) - 1, fp_cus);
255             ptr = strtok_s(str, " ", &pTmp);
256             while (ptr != NULL)
257             {
258                 TaskInfo[i] = atoi(ptr);
259                 ptr = strtok_s(NULL, " ", &pTmp);
260                 //printf("Info : %d\n", TaskInfo[i]);
261                 if (i == 0)
262                 {
263                     cus_job_parameter[j].TaskID = cus_job_number;
264                     cus_job_number++;
265                     //printf(" cus_job_number %d\n ", cus_job_number);
266                 }
267                 else if (i == 1)
268                 {
269                     cus_job_parameter[j].TaskArriveTime = TaskInfo[i];
270                 }
271                 else if (i == 2)
272                 {
273                     cus_job_parameter[j].TaskExecutionTime = TaskInfo[i];
274                 }
275                 else if (i == 3)
276                 {
277                     cus_job_parameter[j].JobDeadline = TaskInfo[i];
278                 }
279                 i++;
280             }
281             /* Initial Priority */
282             cus_job_parameter[j].TaskPriority = j;
283             j++;
284         }
285 
286         fclose(fp);
287         fclose(fp_cus);
288         /* read file */
289     }
290 
291 #endif /* M11102155_PA2_PART_2_CUS */
```

Here I write two function for CUS FIFO queue, Initialization and Insert aperiodic/sporadic jobs.

```c
#ifdef M11102155_PA2_PART_2_CUS

// Insert aperiodic/sporadic job into CUS FIFO queue.
void CUSQInsert(INT16U job_id, INT16U arrive_time, INT16U execution_time, INT16U deadline)
{
    printf("......... %d %d %d %d\n", job_id, arrive_time, execution_time, deadline);
    if (cus_fifo_q_info->num_item != cus_fifo_q_info->size)
    {
        cus_fifo_q[cus_fifo_q_info->front].job_id = job_id;
        cus_fifo_q[cus_fifo_q_info->front].arrive_time = arrive_time;
        cus_fifo_q[cus_fifo_q_info->front].execution_time = execution_time;
        cus_fifo_q[cus_fifo_q_info->front].user_define_deadline = deadline;
        cus_fifo_q_info->front++;
        cus_fifo_q_info->num_item++;
    }
    else
    {
        printf("ERROR : os_core.c ... FIFO QUEUE overflow !!!\n");
    }
}

// Init the CUS FIFO queue.
void CUSQInit()
{
    cus_fifo_q = (CUS_FIFO_Q_NODE*)malloc((OS_MAX_TASKS + 1) * sizeof(CUS_FIFO_Q_NODE));
    cus_fifo_q_info = (CUS_FIFO_Q_INFO*)malloc(1 * sizeof(CUS_FIFO_Q_INFO));

    if ((cus_fifo_q == NULL) | (cus_fifo_q_info == NULL))
    {
        printf("CUS FIFO Q malloc failed !!! \n");
    }
    else
    {
        printf("CUS FIFO Q success !!! \n");
    }

    cus_fifo_q_info->front = 0;
    cus_fifo_q_info->end = 0;
    cus_fifo_q_info->num_item = 0;
    cus_fifo_q_info->size = (OS_MAX_TASKS + 1);

    printf("END OF FIFO QUEUE INIT !!!\n");
}

#endif /* M11102155_PA2_PART_2_CUS */
```

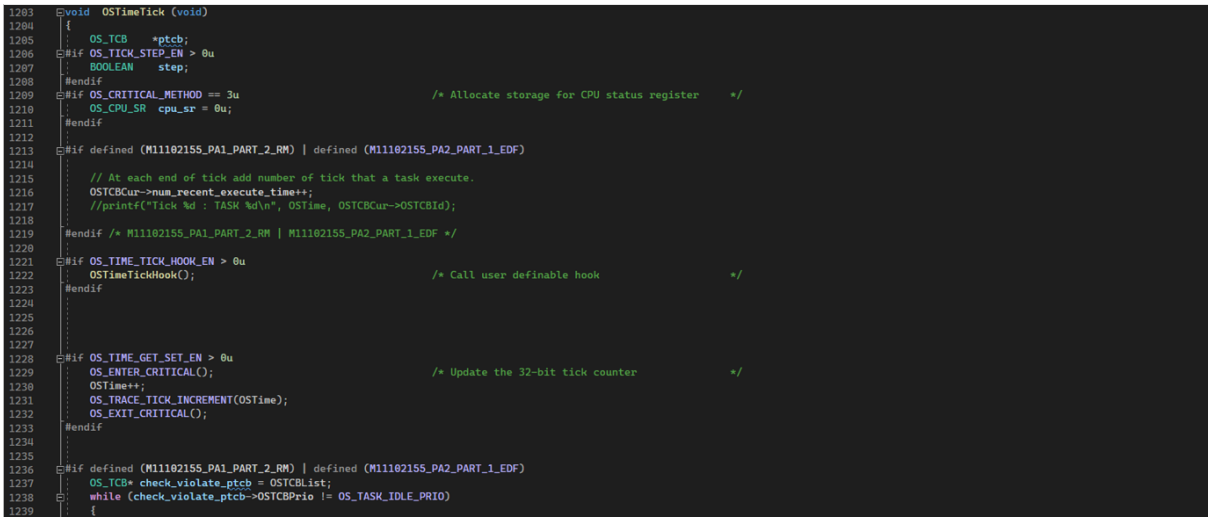
main.c :

In “ main.c ”, I create CUS FIFO queue fisrt, then start to insert aperiodic/sporadic jobs into queue.

```c
#ifdef M11102155_PA2_PART_2_CUS

    // Init CUS FIFO queue.
    CUSQInit();

    // Insert the aperiodic job into CUS FIFO queue.
    int job_id;
    for (job_id = 0; job_id < cus_job_number; job_id++)
    {
        CUSQInsert(cus_job_parameter[job_id].TaskID, cus_job_parameter[job_id].TaskArriveTime, cus_job_parameter[job_id].TaskExecutionTime, cus_job_parameter[job_id].JobDeadline);
    }
    printf("Done insert CUS queue!!!\n");

#endif /* M11102155_PA2_PART_2_CUS */
```

os_core.c :

In “ os_core.c ”, I consider server task(CUS) when occur OSTimeTick() interrupt and OSIntExit() that exist interrupt.

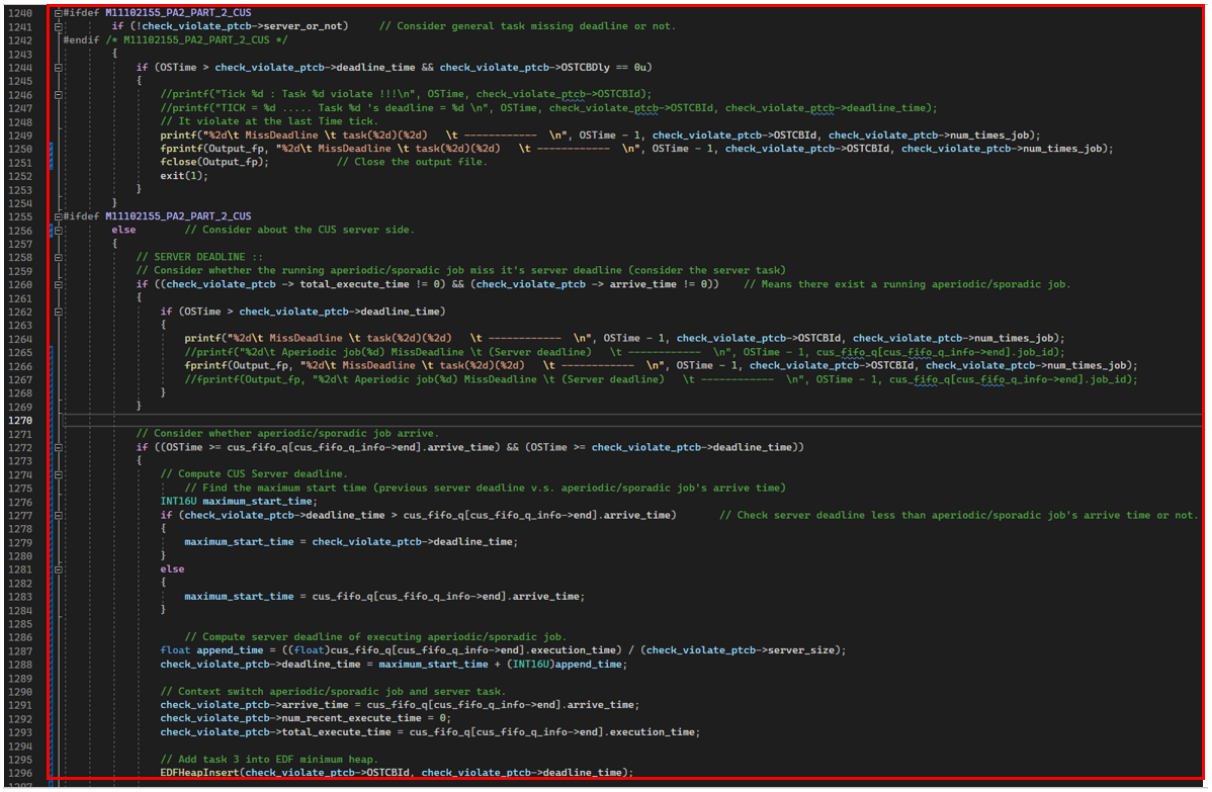


```
1203  void  OSTimeTick (void)
1204  {
1205      OS_TCB     *ptcb;
1206  #if OS_TICK_STEP_EN > 0u
1207      BOOLEAN    step;
1208  #endif
1209  #if OS_CRITICAL_METHOD == 3u                          /* Allocate storage for CPU status register     */
1210      OS_CPU_SR  cpu_sr = 0u;
1211  #endif
1212
1213  #if defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA2_PART_1_EDF)
1214
1215      // At each end of tick add number of tick that a task execute.
1216      OSTCBCur->num_recent_execute_time++;
1217      //printf("Tick %d : TASK %d\n", OSTime, OSTCBCur->OSTCBId);
1218
1219  #endif /* M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF */
1220
1221  #if OS_TIME_TICK_HOOK_EN > 0u
1222      OSTimeTickHook();                                 /* Call user definable hook                     */
1223  #endif
1224
1225
1226
1227
1228  #if OS_TIME_GET_SET_EN > 0u
1229      OS_ENTER_CRITICAL();                              /* Update the 32-bit tick counter               */
1230      OSTime++;
1231      OS_TRACE_TICK_INCREMENT(OSTime);
1232      OS_EXIT_CRITICAL();
1233  #endif
1234
1235
1236  #if defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA2_PART_1_EDF)
1237      OS_TCB* check_violate_ptcb = OSTCBList;
1238      while (check_violate_ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO)
1239      {
```

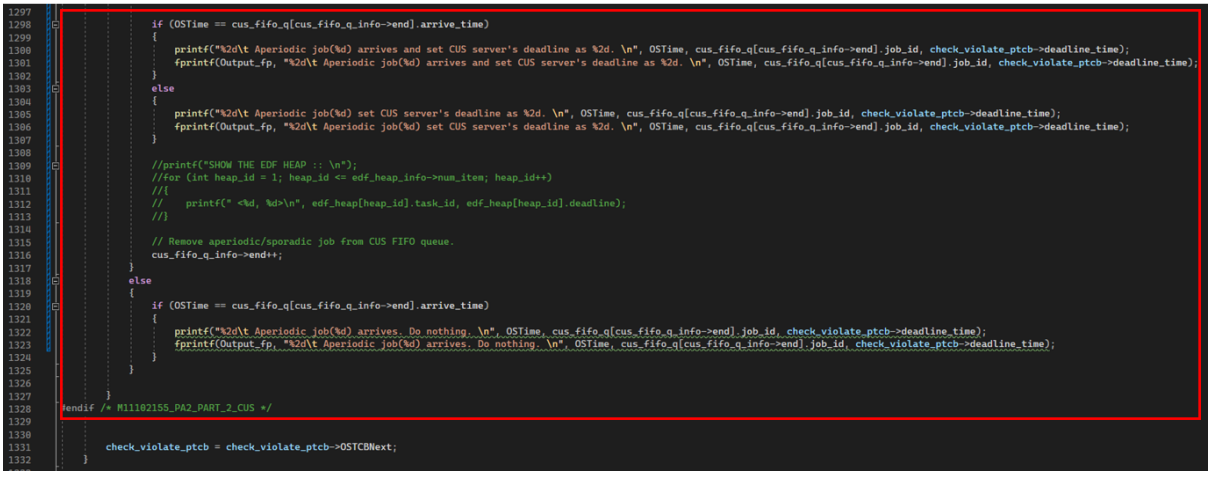
In “ OSTimeTick() ”, when checking whether a task miss it’s deadline, if TCB is general periodic task, then check deadline as previous, but if TCB is server task(CUS), then I will do extra things.

The following things I do when it is a server task(CUS) :

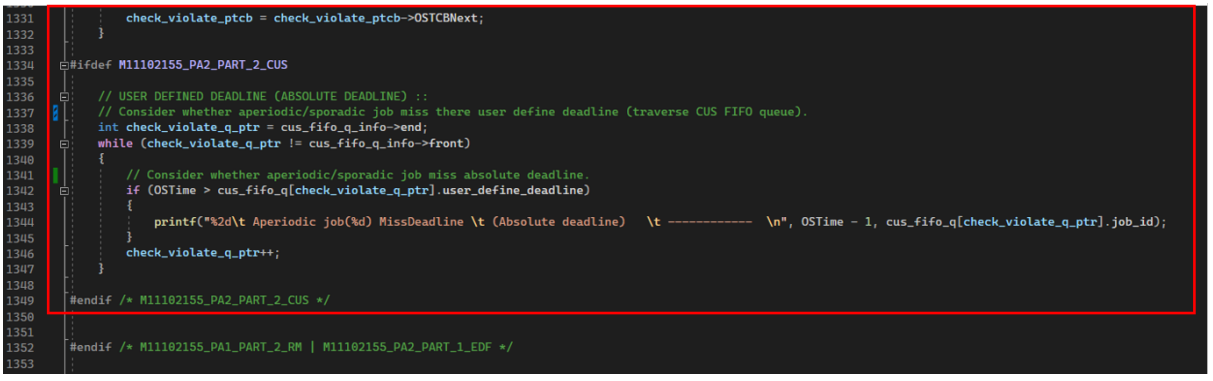
- First, check whether aperiodic/sporadic job that server task(CUS) executed miss server deadline.
- Second, I’ll consider whether exist aperiodic/sporadic job reside in CUS FIFO queue arrives, if so then just let server task(CUS) start to executing this job, and update server task(CUS) deadline.
- Third, check whether exist job arrive but server task(CUS) is executing other aperiodic/sporadic job, if so then just print aperiodic/sporadic job arrive and do nothing.



```
#ifdef M11102155_PA2_PART_2_CUS
        if (!check_violate_ptcb->server_or_not)     // Consider general task missing deadline or not.
#endif /* M11102155_PA2_PART_2_CUS */
        {
            if (OSTime > check_violate_ptcb->deadline_time && check_violate_ptcb->OSTCBDly == 0u)
            {
                //printf("Tick %d : Task %d violate !!!\n", OSTime, check_violate_ptcb->OSTCBId);
                //printf("TICK = %d ..... Task %d 's deadline = %d \n", OSTime, check_violate_ptcb->OSTCBId, check_violate_ptcb->deadline_time);
                // It violate at the last Time tick.
                printf("%2d\t MissDeadline \t task(%2d)(%2d)   \t ------------  \n", OSTime - 1, check_violate_ptcb->OSTCBId, check_violate_ptcb->num_times_job);
                fprintf(Output_fp, "%2d\t MissDeadline \t task(%2d)(%2d)   \t ------------  \n", OSTime - 1, check_violate_ptcb->OSTCBId, check_violate_ptcb->num_times_job);
                fclose(Output_fp);          // Close the output file.
                exit(1);
            }
        }
#ifdef M11102155_PA2_PART_2_CUS
        else        // Consider about the CUS server side.
        {
            // SERVER DEADLINE ::
            // Consider whether the running aperiodic/sporadic job miss it's server deadline (consider the server task)
            if ((check_violate_ptcb -> total_execute_time != 0) && (check_violate_ptcb -> arrive_time != 0))    // Means there exist a running aperiodic/sporadic job.
            {
                if (OSTime > check_violate_ptcb->deadline_time)
                {
                    printf("%2d\t MissDeadline \t task(%2d)(%2d)   \t ------------  \n", OSTime - 1, check_violate_ptcb->OSTCBId, check_violate_ptcb->num_times_job);
                    //printf("%2d\t Aperiodic job(%d) MissDeadline \t (Server deadline)   \t ------------  \n", OSTime - 1, cus_fifo_q[cus_fifo_q_info->end].job_id);
                    fprintf(Output_fp, "%2d\t MissDeadline \t task(%2d)(%2d)   \t ------------  \n", OSTime - 1, check_violate_ptcb->OSTCBId, check_violate_ptcb->num_times_job);
                    //fprintf(Output_fp, "%2d\t Aperiodic job(%d) MissDeadline \t (Server deadline)   \t ------------  \n", OSTime - 1, cus_fifo_q[cus_fifo_q_info->end].job_id);
                }
            }

            // Consider whether aperiodic/sporadic job arrive.
            if ((OSTime >= cus_fifo_q[cus_fifo_q_info->end].arrive_time) && (OSTime >= check_violate_ptcb->deadline_time))
            {
                // Compute CUS Server deadline.
                    // Find the maximum start time (previous server deadline v.s. aperiodic/sporadic job's arrive time)
                INT16U maximum_start_time;
                if (check_violate_ptcb->deadline_time > cus_fifo_q[cus_fifo_q_info->end].arrive_time)       // Check server deadline less than aperiodic/sporadic job's arrive time or not.
                {
                    maximum_start_time = check_violate_ptcb->deadline_time;
                }
                else
                {
                    maximum_start_time = cus_fifo_q[cus_fifo_q_info->end].arrive_time;
                }

                    // Compute server deadline of executing aperiodic/sporadic job.
                float append_time = ((float)cus_fifo_q[cus_fifo_q_info->end].execution_time) / (check_violate_ptcb->server_size);
                check_violate_ptcb->deadline_time = maximum_start_time + (INT16U)append_time;

                // Context switch aperiodic/sporadic job and server task.
                check_violate_ptcb->arrive_time = cus_fifo_q[cus_fifo_q_info->end].arrive_time;
                check_violate_ptcb->num_recent_execute_time = 0;
                check_violate_ptcb->total_execute_time = cus_fifo_q[cus_fifo_q_info->end].execution_time;

                // Add task 3 into EDF minimum heap.
                EDFHeapInsert(check_violate_ptcb->OSTCBId, check_violate_ptcb->deadline_time);
```



```
                if (OSTime == cus_fifo_q[cus_fifo_q_info->end].arrive_time)
                {
                    printf("%2d\t Aperiodic job(%d) arrives and set CUS server's deadline as %2d. \n", OSTime, cus_fifo_q[cus_fifo_q_info->end].job_id, check_violate_ptcb->deadline_time);
                    fprintf(Output_fp, "%2d\t Aperiodic job(%d) arrives and set CUS server's deadline as %2d. \n", OSTime, cus_fifo_q[cus_fifo_q_info->end].job_id, check_violate_ptcb->deadline_time);
                }
                else
                {
                    printf("%2d\t Aperiodic job(%d) set CUS server's deadline as %2d. \n", OSTime, cus_fifo_q[cus_fifo_q_info->end].job_id, check_violate_ptcb->deadline_time);
                    fprintf(Output_fp, "%2d\t Aperiodic job(%d) set CUS server's deadline as %2d. \n", OSTime, cus_fifo_q[cus_fifo_q_info->end].job_id, check_violate_ptcb->deadline_time);
                }

                //printf("SHOW THE EDF HEAP :: \n");
                //for (int heap_id = 1; heap_id <= edf_heap_info->num_item; heap_id++)
                //{
                //    printf(" <%d, %d>\n", edf_heap[heap_id].task_id, edf_heap[heap_id].deadline);
                //}

                // Remove aperiodic/sporadic job from CUS FIFO queue.
                cus_fifo_q_info->end++;
            }
            else
            {
                if (OSTime == cus_fifo_q[cus_fifo_q_info->end].arrive_time)
                {
                    printf("%2d\t Aperiodic job(%d) arrives. Do nothing. \n", OSTime, cus_fifo_q[cus_fifo_q_info->end].job_id, check_violate_ptcb->deadline_time);
                    fprintf(Output_fp, "%2d\t Aperiodic job(%d) arrives. Do nothing. \n", OSTime, cus_fifo_q[cus_fifo_q_info->end].job_id, check_violate_ptcb->deadline_time);
                }
            }
        }
#endif /* M11102155_PA2_PART_2_CUS */

        check_violate_ptcb = check_violate_ptcb->OSTCBNext;
    }
```



```
        check_violate_ptcb = check_violate_ptcb->OSTCBNext;
    }

#ifdef M11102155_PA2_PART_2_CUS

    // USER DEFINED DEADLINE (ABSOLUTE DEADLINE) ::
    // Consider whether aperiodic/sporadic job miss there user define deadline (traverse CUS FIFO queue).
    int check_violate_q_ptr = cus_fifo_q_info->end;
    while (check_violate_q_ptr != cus_fifo_q_info->front)
    {
        // Consider whether aperiodic/sporadic job miss absolute deadline.
        if (OSTime > cus_fifo_q[check_violate_q_ptr].user_define_deadline)
        {
            printf("%2d\t Aperiodic job(%d) MissDeadline \t (Absolute deadline)   \t ------------  \n", OSTime - 1, cus_fifo_q[check_violate_q_ptr].job_id);
        }
        check_violate_q_ptr++;
    }

#endif /* M11102155_PA2_PART_2_CUS */


#endif /* M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF */
```

At the end of OSTimeTick(), waking up general periodic task only.

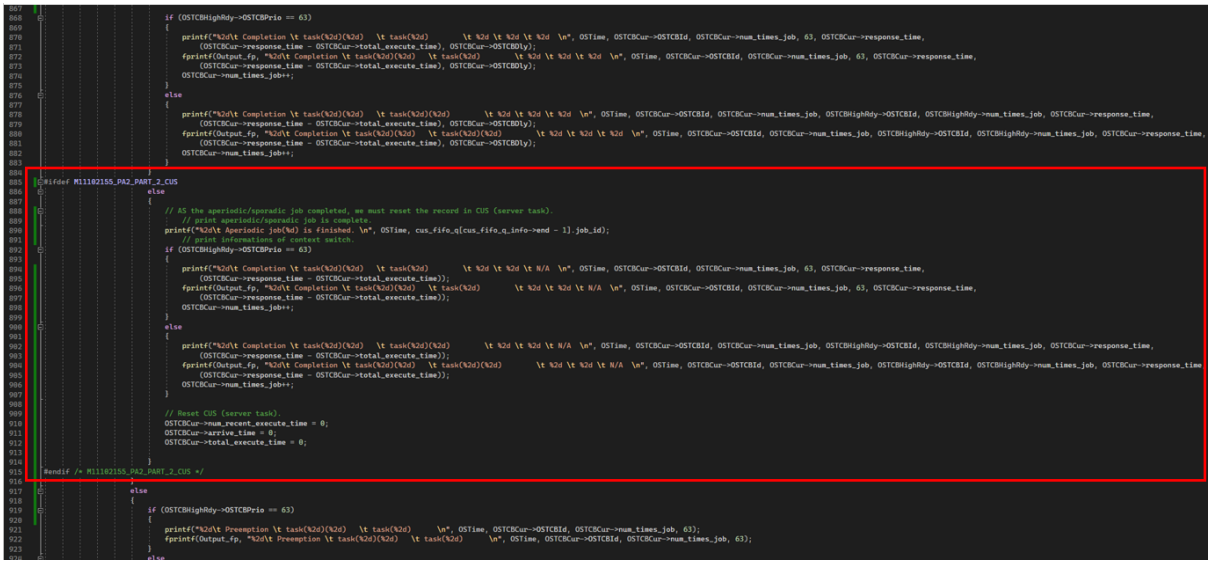
```
    if (OSRunning == OS_TRUE) {
        /* Setting the end time for the OS */
        if (OSTimeGet() > SYSTEM_END_TIME)
        {
#if defined (M11102155_HW1) | defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA2_PART_1_EDF)
            // Close the output file.
            fclose(Output_fp);
#endif /* M11102155_HW1 | M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF*/
            OSRunning = OS_FALSE;
            exit(0);
        }
        /* Setting the end time for the OS */
#if OS_TICK_STEP_EN > 0u
        switch (OSTickStepState) {                          /* Determine whether we need to process a tick  */
            case OS_TICK_STEP_DIS:                          /* Yes, stepping is disabled                    */
                 step = OS_TRUE;
                 break;

            case OS_TICK_STEP_WAIT:                         /* No,  waiting for uC/OS-View to set ...       */
                 step = OS_FALSE;                           /*      .. OSTickStepState to OS_TICK_STEP_ONCE */
                 break;

            case OS_TICK_STEP_ONCE:                         /* Yes, process tick once and wait for next ... */
                 step           = OS_TRUE;                  /*      ... step command from uC/OS-View        */
                 OSTickStepState = OS_TICK_STEP_WAIT;
                 break;

            default:                                        /* Invalid case, correct situation              */
                 step           = OS_TRUE;
                 OSTickStepState = OS_TICK_STEP_DIS;
                 break;
        }
        if (step == OS_FALSE) {                             /* Return if waiting for step command           */
            return;
        }
#endif
        ptcb = OSTCBList;                                   /* Point at first TCB in TCB list               */
        while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {      /* Go through all TCBs in TCB list              */
            OS_ENTER_CRITICAL();
            if (ptcb->OSTCBDly != 0u) {                     /* No, Delayed or waiting for event with TO     */
                ptcb->OSTCBDly--;                           /* Decrement nbr of ticks to end of delay       */
                if (ptcb->OSTCBDly == 0u) {                 /* Check for timeout                            */
                    if ((ptcb->OSTCBStat & OS_STAT_PEND_ANY) != OS_STAT_RDY) {
                        ptcb->OSTCBStat  &= (INT8U)~(INT8U)OS_STAT_PEND_ANY;   /* Yes, Clear status flag   */
                        ptcb->OSTCBStatPend = OS_STAT_PEND_TO;                 /* Indicate PEND timeout    */
                    } else {
                        ptcb->OSTCBStatPend = OS_STAT_PEND_OK;
                    }
                    if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) == OS_STAT_RDY) {  /* Is task suspended?       */

#ifndef M11102155_PA2_PART_1_EDF
                        OSRdyGrp               |= ptcb->OSTCBBitY;             /* No,  Make ready          */
                        OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;

#endif /* M11102155_PA2_PART_1_EDF */
```

```
#if defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA2_PART_1_EDF)

#ifdef M11102155_PA2_PART_2_CUS
                        if (!ptcb->server_or_not)
#endif /* M11102155_PA2_PART_2_CUS */
                        {
                            //OS_ENTER_CRITICAL();
                            ptcb->arrive_time = OSTime;
                            ptcb->num_recent_execute_time = 0;
                            ptcb->deadline_time = ptcb->arrive_time + ptcb->period;

                            if (edf_heap_info->num_item != edf_heap_info->size)
                            {
                                EDFHeapInsert(ptcb->OSTCBId, ptcb->deadline_time);
                            }
                            else
                            {
                                printf("ERROR : os_core.c ... EDF HEAP overflow !!!\n");
                            }
                            //OS_EXIT_CRITICAL();
                        }
#endif /* M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF */

                        OS_TRACE_TASK_READY(ptcb);
                    }
                }
            }
            ptcb = ptcb->OSTCBNext;                         /* Point at next TCB in TCB list                */
            OS_EXIT_CRITICAL();
        }
    }
}
```

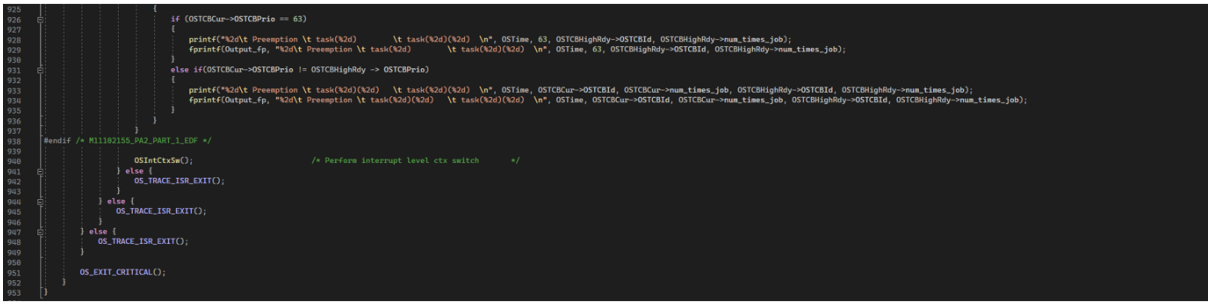
In “ OSIntExit() ”, I just add a condition to check whether current task is server task(CUS) or not, if so then print aperiodic/sporadic job is done, else it is general periodic task, same as previous EDF scheduler.

```
694    // Interrupt (Kernel) level scheduling ::
695   void  OSIntExit (void)
696   {
697   #if OS_CRITICAL_METHOD == 3u                          /* Allocate storage for CPU status register */
698       OS_CPU_SR  cpu_sr = 0u;
699   #endif
700
701       if (OSRunning == OS_TRUE) {
702           OS_ENTER_CRITICAL();
703           if (OSIntNesting > 0u) {                      /* Prevent OSIntNesting from wrapping       */
704               OSIntNesting--;
705           }
706           if (OSIntNesting == 0u) {                     /* Reschedule only if all ISRs complete ... */
707               if (OSLockNesting == 0u) {                /* ... and not locked.                      */
708
709   #ifndef M11102155_PA2_PART_1_EDF
710                   OS_SchedNew();
711                   OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
712   #else
713                   OS_ENTER_CRITICAL();
714                   if (edf_heap_info -> num_item == 0)         // there is no task in the waiting queue, execute the Idle task
715                   {
716                       if (OSTCBCur->num_recent_execute_time == OSTCBCur->total_execute_time)
717                       {
718                           OSPrioHighRdy = 63;
719                           OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
720                       }
721                   }
722                   else
723                   {
724                       if (OSTCBCur->num_recent_execute_time == OSTCBCur->total_execute_time)
725                       {
726                           EDFHeapDelete();
727                           OSPrioHighRdy = edf_heap[1].task_id;
728                           OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
729                       }
730                       else
731                       {
732                           OSPrioHighRdy = edf_heap[1].task_id;
733                           OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
734                       }
735                   }
736                   OS_EXIT_CRITICAL();
737
738   #endif /* M11102155_PA2_PART_1_EDF */
739
740                   if (OSPrioHighRdy != OSPrioCur) {         /* No Ctx Sw if current task is highest rdy */
741
742   #ifndef M11102155_HW1
743   #if OS_TASK_PROFILE_EN > 0u
744                       OSTCBHighRdy->OSTCBCtxSwCtr++;        /* Inc. # of context switches to this task  */
745   #endif
746                       OSCtxSwCtr++;                         /* Keep track of the number of ctx switches */
747   #endif /* M11102155_HW1 */
```

```
749   #if OS_TASK_CREATE_EXT_EN > 0u
750   #if defined(OS_TLS_TBL_SIZE) && (OS_TLS_TBL_SIZE > 0u)
751                       OS_TLS_TaskSw();
752   #endif
753   #endif
754                       OS_TRACE_ISR_EXIT_TO_SCHEDULER();
755
756   #ifdef M11102155_HW1 Inactive Preprocessor Block
761   #endif /* M11102155_HW1 */
762
763   #ifdef M11102155_PA1_PART_2_RM Inactive Preprocessor Block
822   #endif /* M11102155_PA1_PART_2_RM */
823
824   #ifdef M11102155_PA2_PART_1_EDF
825                       //printf("TASK %d num_exe_time = %d, total = %d\n", OSTCBCur->OSTCBId, OSTCBCur->num_recent_execute_time, OSTCBCur->total_execute_time);
826                       if (OSTCBCur->num_recent_execute_time == OSTCBCur->total_execute_time)
827                       {
828                           // Compute response time.
829                           OSTCBCur->response_time = OSTime - OSTCBCur->arrive_time;
830
831                           // Do delay !!!
832                           INT32U ticks = OSTCBCur->deadline_time - OSTime;
833                           //printf("TICK %d    TASK %d's delay = %d\n", OSTime, OSTCBCur->OSTCBId, ticks);
834                           if (ticks > 0u) {                         /* 0 means no delay!                        */
835                               OS_ENTER_CRITICAL();
836                               OS_TRACE_TASK_SUSPENDED(OSTCBCur);
837                               OSTCBCur->OSTCBDly = ticks;           /* Load ticks in TCB                        */
838                               OS_TRACE_TASK_DLY(ticks);
839                               OS_EXIT_CRITICAL();
840                           }
841                           else
842                           {
843                               // If the task arrive after it complete immediately.
844                               //printf("TICK %d hahaha TASK %d \n", OSTime, OSTCBCur->OSTCBId);
845                               OS_ENTER_CRITICAL();
846                               OSTCBCur->arrive_time = OSTime;
847                               OSTCBCur->num_recent_execute_time = 0;
848                               OSTCBCur->deadline_time = OSTCBCur->arrive_time + OSTCBCur->period;
849
850                               if (edf_heap_info->num_item != edf_heap_info->size)
851                               {
852                                   EDFHeapInsert(OSTCBCur->OSTCBId, OSTCBCur->deadline_time);
853                               }
854                               else
855                               {
856                                   printf("ERROR : os_core.c ... EDF HEAP overflow !!!\n");
857                               }
858
859                               OS_EXIT_CRITICAL();
860                           }
861
862   #ifdef M11102155_PA2_PART_2_CUS
863
864                       if (!OSTCBCur->server_or_not)
865                       {
866   #endif /* M11102155_PA2_PART_2_CUS */
867
```



```
                        if (OSTCBHighRdy->OSTCBPrio == 63)
                        {
                            printf("%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)       \t %2d \t %2d \t %2d  \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63, OSTCBCur->response_time,
                                (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBDly);
                            fprintf(Output_fp, "%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)       \t %2d \t %2d \t %2d  \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63, OSTCBCur->response_time,
                                (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBDly);
                            OSTCBCur->num_times_job++;
                        }
                        else
                        {
                            printf("%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)(%2d)       \t %2d \t %2d \t %2d  \n", OSTime, OSTCBCur->OSTCBId, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->num_times_job, OSTCBCur->response_time,
                                (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBDly);
                            fprintf(Output_fp, "%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)(%2d)       \t %2d \t %2d \t %2d  \n", OSTime, OSTCBCur->OSTCBId, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->num_times_job, OSTCBCur->response_time,
                                (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBDly);
                            OSTCBCur->num_times_job++;
                        }
                    }
#ifdef M11102155_PA2_PART_2_CUS
                    else
                    {
                        // AS the aperiodic/sporadic job completed, we must reset the record in CUS (server task).
                        // print aperiodic/sporadic job is complete.
                        printf("%2d\t Aperiodic job(%d) is finished. \n", OSTime, cus_fifo_q[cus_fifo_q_info->end - 1].job_id);
                        // print informations of context switch.
                        if (OSTCBHighRdy->OSTCBPrio == 63)
                        {
                            printf("%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)       \t %2d \t %2d \t N/A  \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63, OSTCBCur->response_time,
                                (OSTCBCur->response_time - OSTCBCur->total_execute_time));
                            fprintf(Output_fp, "%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)       \t %2d \t %2d \t N/A  \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63, OSTCBCur->response_time,
                                (OSTCBCur->response_time - OSTCBCur->total_execute_time));
                            OSTCBCur->num_times_job++;
                        }
                        else
                        {
                            printf("%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)(%2d)       \t %2d \t %2d \t N/A  \n", OSTime, OSTCBCur->OSTCBId, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->num_times_job, OSTCBCur->response_time,
                                (OSTCBCur->response_time - OSTCBCur->total_execute_time));
                            fprintf(Output_fp, "%2d\t Completion \t task(%2d)(%2d)   \t task(%2d)(%2d)       \t %2d \t %2d \t N/A  \n", OSTime, OSTCBCur->OSTCBId, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->num_times_job, OSTCBCur->response_time
                                (OSTCBCur->response_time - OSTCBCur->total_execute_time));
                            OSTCBCur->num_times_job++;
                        }

                        // Reset CUS (server task).
                        OSTCBCur->num_recent_execute_time = 0;
                        OSTCBCur->arrive_time = 0;
                        OSTCBCur->total_execute_time = 0;

                    }
#endif /* M11102155_PA2_PART_2_CUS */
                }
                else
                {
                    if (OSTCBHighRdy->OSTCBPrio == 63)
                    {
                        printf("%2d\t Preemption \t task(%2d)(%2d)   \t task(%2d)      \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63);
                        fprintf(Output_fp, "%2d\t Preemption \t task(%2d)(%2d)   \t task(%2d)      \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63);
                    }
                    else
```



```
                    {
                        if (OSTCBCur->OSTCBPrio == 63)
                        {
                            printf("%2d\t Preemption \t task(%2d)       \t task(%2d)(%2d)  \n", OSTime, 63, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->num_times_job);
                            fprintf(Output_fp, "%2d\t Preemption \t task(%2d)       \t task(%2d)(%2d)  \n", OSTime, 63, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->num_times_job);
                        }
                        else if(OSTCBCur->OSTCBPrio != OSTCBHighRdy -> OSTCBPrio)
                        {
                            printf("%2d\t Preemption \t task(%2d)(%2d)   \t task(%2d)(%2d)  \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->num_times_job);
                            fprintf(Output_fp, "%2d\t Preemption \t task(%2d)(%2d)   \t task(%2d)(%2d)  \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, OSTCBHighRdy->OSTCBId, OSTCBHighRdy->num_times_job);
                        }
                    }
                }
#endif /* M11102155_PA2_PART_1_EDF */

                OSIntCtxSw();                       /* Perform interrupt level ctx switch      */
            } else {
                OS_TRACE_ISR_EXIT();
            }
        } else {
            OS_TRACE_ISR_EXIT();
        }
    } else {
        OS_TRACE_ISR_EXIT();
    }

    OS_EXIT_CRITICAL();
}
```