

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 segment.

```

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_EIFO
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 deadline 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 // Delete the task from EDF Heap.
213 void EDFHeapDelete()
214 {
215     if (edf_heap_info->num_item > 0)
216     {
217         //printf("TICK %d :: delete < %d , %d >\n", OSTime, edf_heap[1].task_id, edf_heap[1].deadline);
218         EDFHeapSwap(&(edf_heap[1]), &(edf_heap[edf_heap_info->num_item]));
219         edf_heap_info->num_item--;
220         EDFHeapify(1);
221     }
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 #endif /* M11102155_PA2_PART_1_EDF */
281
```

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_STKSIZE - 1],
192                         TaskParameter[n].TaskID,
193                         TaskParameter[n].TaskID,
194                         &Task_STK[n][0],
195                         TASK_STKSIZE,
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 */
209
210
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 */
238

```

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()

“

```
266 #if defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA2_PART_1_EDF)
267
268 static void task(void* p_arg)
269 {
270     task_para_set* task_date;
271     task_date = p_arg;
272
273     while (1)
274     {
275         // For every task keep executing until finish or preemptive
276         while (OSTCBCur->num_recent_execute_time < OSTCBCur->total_execute_time);    // TimeTick interrupt happen before while loop end !!!
277
278         OSTimeDly(0);
279     }
280 }
281 #endif /* M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF */
282
```

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, then cancel the operation of setting the ready table.

```
54 void OSTimeDly (INT32U ticks)
55 {
56     INT8U y;
57     #if OS_CRITICAL_METHOD == 3u                                /* Allocate storage for CPU status register */
58     OS_CPU_SR cpu_sr = 0u;
59     #endif
60
61     #if defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA2_PART_1_EDF)
62
63         // Compute the delay time before task arrive again.
64         OS_ENTER_CRITICAL();
65         EDFHeapDelete();           // Remove the task from heap.
66         ticks = OSTCBCur->deadline_time - OSTime;
67         OSTCBCur->response_time = OSTime - OSTCBCur->arrive_time;
68         OS_EXIT_CRITICAL();
69     #endif /* M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF */
70
71
72
73     if (OSIntNesting > 0u) {                                /* See if trying to call from an ISR */
74         return;
75     }
76     if (OSLockNesting > 0u) {                                /* See if called with scheduler locked */
77         return;
78     }
79     if (ticks > 0u) {                                       /* 0 means no delay! */
80         OS_ENTER_CRITICAL();
81
82         #ifndef M11102155_PA2_PART_1_EDF
83             y          = OSTCBCur->OSTCBY;                  /* Delay current task */
84             OSRdyTbl[y] &= (OS_PRIO)~OSTCBCur->OSTCBBitX;
85             //OS_TRACE_TASK_SUSPENDED(OSTCBCur);
86             if (OSRdyTbl[y] == 0u) {
87                 OSRdyGrp &= (OS_PRIO)~OSTCBCur->OSTCBBitY;
88             }
89         #endif /* M11102155_PA2_PART_1_EDF */
90
91             OS_TRACE_TASK_SUSPENDED(OSTCBCur);
92             OSTCBCur->OSTCBDly = ticks;                   /* Load ticks in TCB */
93             OS_TRACE_TASK_DLY(ticks);
94             OS_EXIT_CRITICAL();
95             OS_Sched();                                    /* Find next task to run!
96         }
97     }
```

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.

```
2584  #ifndef M11102155_PA2_PART_1_EDF
2585
2586  #ifndef M11102155_PA1_PART_2_RM
2587
2588      OSRdyGrp |= ptcb->OSTCBBitY;           /* Make task ready to run */
2589      OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
2590
2591  #else
2592      if (p_arg != 0)
2593      {
2594          if (task_parameter->TaskArriveTime == 0)
2595          {
2596              OSRdyGrp |= ptcb->OSTCBBitY;           /* Make task ready to run */
2597              OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
2598          }
2599      }
2600      else
2601      {
2602          OSRdyGrp |= ptcb->OSTCBBitY;           /* Make task ready to run */
2603          OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
2604      }
2605
2606  #endif /* M11102155_PA1_PART_2_RM */
2607
2608  #else
2609
2610      if (p_arg != 0)
2611      {
2612          if (task_parameter->TaskArriveTime == 0)
2613          {
2614
2615              if (edf_heap_info->num_item != edf_heap_info->size)
2616              {
2617                  EDFHeapInsert(task_parameter->TaskID, ptcb->deadline_time);
2618              }
2619              else
2620              {
2621                  printf("ERROR : os_core.c ... EDF HEAP overflow !!!\n");
2622              }
2623
2624          }
2625      }
2626
2627      // consider idle task or not ?
2628
2629  #endif /* M11102155_PA2_PART_1_EDF */
2630
```

Second I modify task level context switch, “ OS_Sched() ”, 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.

```

2008 // Task level scheduling ::  
2009 void OS_Sched(void)  
2010 {  
2011     if (OS_CRITICAL_METHOD == 3u) /* Allocate storage for CPU status register */  
2012     {  
2013         OS_CPU_SR cpu_sr = 0u;  
2014     }  
2015  
2016     OS_ENTER_CRITICAL();  
2017     if (OSIntNesting == 0u) { /* Schedule only if all ISRs done and ... */  
2018         if (OSLockNesting == 0u) { /* ... scheduler is not locked */  
2019             OS_SCHEDULER_NESTING++;  
2020             #ifndef M11102155_PA2_PART_1_EDF  
2021                 OS_SchedNew();  
2022                 OSTCBHighRdy = OSTCBPriTbl[OSPriority];  
2023             #else  
2024                 OS_ENTER_CRITICAL();  
2025                 if (edf_heap_info->num_item != 0)  
2026                 {  
2027                     OSPriorityHighRdy = edf_heap[1].task_id;  
2028                 }  
2029                 else  
2030                 {  
2031                     OSPriorityHighRdy = 63;  
2032                 }  
2033                 OS_EXIT_CRITICAL();  
2034                 OSTCBHighRdy = OSTCBPriTbl[OSPriorityHighRdy];  
2035             #endif /* M11102155_PA2_PART_1_EDF */  
2036         }  
2037     }  
2038     if (OSPriorityHighRdy != OSPriorityCur) { /* No Ctx Sw if current task is highest rdy */  
2039     #ifndef M11102155_HW1  
2040         if (OS_TASK_PROFILE_EN > 0u)  
2041             OSTCBHighRdy->OSTCBctxSwCtr++; /* Inc. # of context switches to this task */  
2042     #endif  
2043         OSTCBctxSwCtr++; /* Increment context switch counter */  
2044     #endif /* M11102155_HW1 */  
2045 }
```

```

2051 #if OS_TASK_CREATE_EXT_EN == 0u  
2052 #if defined(OS_TLS_TBL_SIZE) && (OS_TLS_TBL_SIZE > 0u)  
2053     OS_TLS_TaskSw();  
2054 #endif  
2055 #endif  
2056 #ifndef M11102155_HW1 /* Inactive Preprocessor Block */  
2057 #endif /* M11102155_HW1 */  
2058  
2059 #if defined (M11102155_PA1_PART_2_RM) | defined(M11102155_PA2_PART_1_EDF)  
2060     if (OSTCBHighRdy == 63)  
2061     {  
2062         printf("%d\t Completion \t task(%d)(%d) \t task(%d)\n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63, OSTCBCur->response_time,  
2063             (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBId);  
2064         fprintf(Output_fp, "%d\t Completion \t task(%d)(%d) \t task(%d)\n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63, OSTCBCur->response_time,  
2065             (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBId);  
2066     }  
2067     else  
2068     {  
2069         printf("%d\t Completion \t task(%d)(%d) \t task(%d)(%d) \t %d\t %d \t %d\n", OSTime, OSTCBHighRdy->num_times_job, OSTCBHighRdy->OSTCBId,  
2070             OSTCBHighRdy->response_time, (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBId);  
2071         fprintf(Output_fp, "%d\t Completion \t task(%d)(%d) \t task(%d)(%d) \t %d\t %d \t %d\n", OSTime, OSTCBHighRdy->num_times_job, OSTCBHighRdy->OSTCBId,  
2072             OSTCBHighRdy->response_time, (OSTCBCur->response_time - OSTCBCur->total_execute_time), OSTCBCur->OSTCBId);  
2073     }  
2074     OSTCBcur->num_ticks++;  
2075 }  
2076 #endif /* M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF */  
2077  
2078     OS_TASK_SW(); /* Perform a context switch */  
2079 }  
2080 }  
2081 }  
2082 }  
2083 }  
2084 OS_EXIT_CRITICAL();  
2085 }
```

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.

```

1178 void OSTimeTick (void)  
1179 {  
1180     OS_TCB *ptcb;  
1181     #if OS_TICK_STEP_EN > 0u  
1182         BOOLEAN step;  
1183     #endif  
1184     #if OS_CRITICAL_METHOD == 3u /* Allocate storage for CPU status register */  
1185         OS_CPU_SR cpu_sr = 0u;  
1186     #endif  
1187  
1188     #if defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA2_PART_1_EDF)  
1189     // At each end of tick add number of tick that a task execute.  
1190     OSTCBcur->num_recent_execute_time++;  
1191     //printf("Tick %d : TASK %d\n", OSTime, OSTCBCur->OSTCBId);  
1192  
1193     #endif /* M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF */  
1194  
1195 #if OS_TIME_TICK_HOOK_EN > 0u  
1196     OSTimeTickHook(); /* Call user definable hook */  
1197 #endif  
1198  
1199  
1200  
1201  
1202  
1203  
1204 #if OS_TRACE_TICK_INCREMENT(OSTime)  
1205     OS_ENTER_CRITICAL();  
1206     OSTime++; /* Update the 32-bit tick counter */  
1207     OS_TRACE_TICK_INCREMENT(OSTime);  
1208     OS_EXIT_CRITICAL();  
1209 #endif
```

```

1210
1211
1212 #endif defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA2_PART_1_EDF)
1213 OS_TCB* check_violate_ptcb = OSTCBList;
1214 while (check_violate_ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO)
1215 {
1216     if (OSTime > check_violate_ptcb->deadline_time && check_violate_ptcb->OSTCBdly == 0u)
1217     {
1218         //printf("Tick %d : Task %d violate !!!\n", OSTime, check_violate_ptcb->OSTCBId);
1219         //printf("TICK = %d .... Task %d 's deadline = %d \n", OSTime, check_violate_ptcb->OSTCBId, check_violate_ptcb->deadline_time);
1220         // It violate at the last Time tick.
1221         printf("%d\t MissDeadline \t task(%d)\t \t ----- \n", OSTime - 1, check_violate_ptcb->OSTCBId, check_violate_ptcb->num_times_job);
1222         fprintf(Output_fp, "%d\t MissDeadline \t task(%d)\t \t ----- \n", OSTime - 1, check_violate_ptcb->OSTCBId, check_violate_ptcb->num_times_job);
1223         fclose(Output_fp); // Close the output file.
1224         exit(1);
1225     }
1226     check_violate_ptcb = check_violate_ptcb->OSTCBNext;
1227 }
1228 #endif /* M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF */
1229
1230
1231 if (OSRunning == OS_TRUE) { /* Setting the end time for the OS */
1232     if (OSTimeGet() > SYSTEM_END_TIME)
1233     {
1234         #if defined (M11102155_HW1) | defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA2_PART_1_EDF)
1235             // Close the output file.
1236             fclose(Output_fp);
1237         #endif /* M11102155_HW1 | M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF */
1238         OSRunning = OS_FALSE;
1239         exit(0);
1240     }
1241 }

```

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

```

1274
1275     ptcb->OSTCBBstat &= (~INITB0)^(~INITB0)OS_STAT_PEND_ANY; /* Yes, Clear status flag */
1276     ptcb->OSTCBBstatPend = OS_STAT_PEND_TO; /* Indicate PEND timeout */
1277 } else {
1278     ptcb->OSTCBBstatPend = OS_STAT_PEND_OK;
1279 }
1280 if ((ptcb->OSTCBBstat & OS_STAT_SUSPEND) == OS_STAT_RDY) { /* Is task suspended? */
1281
1282 #ifndef M11102155_PA2_PART_1_EDF
1283     OSRdyGrp |= ptcb->OSTCBBitY; /* No, Make ready */
1284     OSRdyTbl[ptcb->OSTCBV] |= ptcb->OSTCBBitX;
1285 #endif /* M11102155_PA2_PART_1_EDF */
1286
1287 #if defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA2_PART_1_EDF)
1288     //OS_ENTER_CRITICAL();
1289     ptcb->arrive_time = OSTime;
1290     ptcb->num_recent_execute_time = 0;
1291     ptcb->deadline_time = ptcb->arrive_time + ptcb->period;
1292
1293     if (edf_heap_info->num_item != edf_heap_info->size)
1294     {
1295         EDFHeapInsert(ptcb->OSTCBId, ptcb->deadline_time);
1296     }
1297     else
1298     {
1299         printf("ERROR : ps.core.c ... EDF HEAP overflow !!!\n");
1300     }
1301     //OS_EXIT_CRITICAL();
1302 #endif /* M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF */
1303
1304     OS_TRACE_TASK_READY(ptcb);
1305
1306 }
1307
1308     ptcb = ptcb->OSTCBNext; /* Point at next TCB in TCB list */
1309     OS_EXIT_CRITICAL();
1310 }
1311

```

In “ OSIntExit() ”, consider the interrupt level context switch, there exist some cases. First the EDF minimum heap is empty, there is no ready task, set the next task into Idle task. Second the current task just complete but arrive immediatly, 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 #ifndef 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 (OSInNesting > 0u) { /* Prevent OSIntNesting from wrapping */
704             OSInNesting--;
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_SchedNow();
711             OSTCBHighRdy = OSTCBPrioTbl[OSPriorityHighRdy];
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                     OSPriorityHighRdy = 63;
719                     OSTCBHighRdy = OSTCBPrioTbl[OSPriorityHighRdy];
720                 }
721             }
722             else
723             {
724                 if (OSTCBCur->num_recent_execute_time == OSTCBCur->total_execute_time)
725                 {
726                     EDFHeapDelete();
727                     OSPriorityHighRdy = edf_heap[1].task_id;
728                     OSTCBHighRdy = OSTCBPrioTbl[OSPriorityHighRdy];
729                 }
730                 else
731                 {
732                     OSPriorityHighRdy = edf_heap[1].task_id;
733                     OSTCBHighRdy = OSTCBPrioTbl[OSPriorityHighRdy];
734                 }
735             }
736             OS_EXIT_CRITICAL();
737
738 #endif // M11102155_PA2_PART_1_EDF */
739     }
740 }

```

```

737 #endif /* M11102155_PA2_PART_1_EDF */
738
739     if (OSPriorityHighRdy != OSPriorityCur) { /* No Ctx Sw if current task is highest rdy */
740
741 #ifndef M11102155_HW1
742 #if OS_TASK_PROFILE_EN > 0u
743         OSTCBHighRdy->OSTCBxCtxSwCtr++;
744 #endif
745         OSCTxCtxSwCtr++;
746 #endif /* M11102155_HW1 */
747
748 #if OS_TASK_CREATE_EXT_EN > 0u
749 #if defined(OS_TLS_TBL_SIZE) && (OS_TLS_TBL_SIZE > 0u)
750     OS_TLS_TaskSw();
751 #endif
752 #endif
753
754     OS_TRACE_ISR_EXIT_TO_SCHEDULER();
755
756 #ifndef M11102155_HW1
757 #endif /* M11102155_HW1 */
758
759 #ifndef M11102155_PA1_PART_2_RM
760     Inactive Preprocessor Block
761 #endif /* M11102155_PA1_PART_2_RM */
762
763 #ifndef M11102155_PA1_PART_2_RM
764     Inactive Preprocessor Block
765 #endif /* M11102155_PA1_PART_2_RM */
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
```

```

876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
endif /* M11102155_PA2_PART_1_EDF */

918
919     OSIntCtxSw();           /* Perform interrupt level ctx switch */
920
921 } else {
922     OS_TRACE_ISR_EXIT();
923 }
924
925 } else {
926     OS_TRACE_ISR_EXIT();
927 }
928
929 OS_EXIT_CRITICAL();

```

[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

```

#ifndef 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, first 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
104    else
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

```

189 #ifndef M11102155_PA2_PART_2_CUS
190
191 // Node in the CUS FIFO queue, for storing informations of aperiodic/sporadic jobs.
192 typedef struct CUS_FIFO_Q_NODE
193 {
194     INT16U job_id;
195     INT16U arrive_time;
196     INT16U execution_time;
197     INT16U user_define_deadline;
198 }CUS_FIFO_Q_NODE;
199
200
201 // For recording the informations of CUS FIFO queue.
202 typedef struct CUS_FIFO_Q_INFO
203 {
204     INT16U front;
205     INT16U end;
206     INT16U num_item;
207     INT16U size;
208
209 }CUS_FIFO_Q_INFO;
210
211 CUS_FIFO_Q_INFO* cus_fifo_q_info;
212 CUS_FIFO_Q_NODE* cus_fifo_q;
213
214 #endif /* M11102155_PA2_PART_2_CUS */
215

```

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 server task(CUS).

```

819 #ifndef M11102155_PA2_PART_2_CUS
820
821     BOOLEAN server_or_not;      // 0 : general task, 1 : server.
822     float server_size;         // recording the server size.
823
824 #endif /* M11102155_PA2_PART_2_CUS */
825
826
827 } OS_TCB;
828

```

app_hooks.c :

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

```

112
113 #ifndef M11102155_PA2_PART_2_CUS Inactive Preprocessor Block
179
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 cus_job_number = 0;
248 i, j = 0;
249 while (!feof(fp_cus))
{
250     i = 0;
251     memset(str, 0, sizeof(str));
252     fgets(str, sizeof(str) - 1, fp_cus);
253     ptr = strtok_s(str, " ", &pTmp);
254     while (ptr != NULL)
255     {
256         TaskInfo[i] = atoi(ptr);
257         ptr = strtok_s(NULL, " ", &pTmp);
258         //printf("Info : %d\n", TaskInfo[i]);
259         if (i == 0)
260         {
261             cus_job_parameter[j].TaskID = cus_job_number;
262             cus_job_number++;
263             //printf(" cus_job_number %d\n ", cus_job_number);
264         }
265         else if (i == 1)
266         {
267             cus_job_parameter[j].TaskArriveTime = TaskInfo[i];
268         }
269         else if (i == 2)
270         {
271             cus_job_parameter[j].TaskExecutionTime = TaskInfo[i];
272         }
273         else if (i == 3)
274         {
275             cus_job_parameter[j].JobDeadline = TaskInfo[i];
276         }
277         i++;
278     }
279     /* Initial Priority */
280     cus_job_parameter[j].TaskPriority = j;
281     j++;
282 }
283

284 fclose(fp);
285 fclose(fp_cus);
286 /* read file */
287 }

288 #endif /* M11102155_PA2_PART_2_CUS */

```

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

```

397 // #ifndef M11102155_PA2_PART_2_CUS
398 // Insert aperiodic/sporadic job into CUS FIFO queue.
399 void CUSQInsert(INT16U job_id, INT16U arrive_time, INT16U execution_time, INT16U deadline)
400 {
401     printf("..... %d %d %d\n", job_id, arrive_time, execution_time, deadline);
402     if (cus_fifo_q_info->num_item != cus_fifo_q_info->size)
403     {
404         cus_fifo_q[cus_fifo_q_info->front].job_id = job_id;
405         cus_fifo_q[cus_fifo_q_info->front].arrive_time = arrive_time;
406         cus_fifo_q[cus_fifo_q_info->front].execution_time = execution_time;
407         cus_fifo_q[cus_fifo_q_info->front].user_define_deadline = deadline;
408         cus_fifo_q_info->front++;
409         cus_fifo_q_info->num_item++;
410     }
411     else
412     {
413         printf("ERROR : os_core.c ... FIFO QUEUE overflow !!!\n");
414     }
415 }
416
417
418 // Init the CUS FIFO queue.
419 void CUSQInit()
420 {
421     cus_fifo_q = (CUS_FIFO_Q_NODE*)malloc((OS_MAX_TASKS + 1) * sizeof(CUS_FIFO_Q_NODE));
422     cus_fifo_q_info = (CUS_FIFO_Q_INFO*)malloc(1 * sizeof(CUS_FIFO_Q_INFO));
423
424     if ((cus_fifo_q == NULL) | (cus_fifo_q_info == NULL))
425     {
426         printf("CUS FIFO Q malloc failed !!! \n");
427     }
428     else
429     {
430         printf("CUS FIFO Q success !!! \n");
431     }
432
433     cus_fifo_q_info->front = 0;
434     cus_fifo_q_info->end = 0;
435     cus_fifo_q_info->num_item = 0;
436     cus_fifo_q_info->size = (OS_MAX_TASKS + 1);
437
438     printf("END OF FIFO QUEUE INIT !!!\n");
439 }
440
441
442 #endif /* M11102155_PA2_PART_2_CUS */
443

```

main.c :

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

```

126 // #ifndef M11102155_PA2_PART_2_CUS
127 // Init CUS FIFO queue.
128 CUSQInit();
129
130 // Insert the aperiodic job into CUS FIFO queue.
131 int job_id;
132 for (job_id = 0; job_id < cus_job_number; job_id++)
133 {
134     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);
135 }
136
137 printf("Done insert CUS queue!!!\n");
138
139
140 #endif /* M11102155_PA2_PART_2_CUS */
141

```

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->n_ticks_executed++;
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     #if OS_TIME_GET_SET_EN > 0u
1228         OS_ENTER_CRITICAL();                                /* Update the 32-bit tick counter */
1229         OSTime++;
1230         OS_TRACE_TICK_INCREMENT(OSTime);
1231         OS_EXIT_CRITICAL();
1232     #endif
1233
1234
1235     #if defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA2_PART_1_EDF)
1236         OS_TCB* check_violate_ptcb = OSTCBList;
1237         while (check_violate_ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO)
1238     {
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.

```

1240 #ifndef M11102155_PA2_PART_2_CUS
1241     if (!check_violate_ptcb->server_or_not) // Consider general task missing deadline or not.
1242     {
1243         if (OSTime > check_violate_ptcb->deadline_time && check_violate_ptcb->OSTCBId == 0u)
1244         {
1245             //printf("Tick %d : Task %d violate !!!\n", OSTime, check_violate_ptcb->OSTCBId);
1246             //printf("TICK = %d ... Task %d's deadline = %d \n", OSTime, check_violate_ptcb->OSTCBId, check_violate_ptcb->deadline_time);
1247             // It violate at the last Time tick.
1248             printf("%2dt MissDeadline \t task(%2d) \t ----- \n", OSTime - 1, check_violate_ptcb->OSTCBId, check_violate_ptcb->num_times_job);
1249             fprintf(Output_fp, "%2dt MissDeadline \t task(%2d) \t ----- \n", OSTime - 1, check_violate_ptcb->OSTCBId, check_violate_ptcb->num_times_job);
1250             fclose(Output_fp); // Close the output file.
1251             exit(1);
1252         }
1253     }
1254 #endif M11102155_PA2_PART_2_CUS
1255     else // Consider about the CUS server side.
1256     {
1257         // SERVER DEADLINE :::
1258         // Consider whether the running aperiodic/sporadic job miss it's server deadline (consider the server task)
1259         if ((check_violate_ptcb -> total_execute_time != 0) && (check_violate_ptcb -> arrive_time != 0)) // Means there exist a running aperiodic/sporadic job.
1260         {
1261             if (OSTime > check_violate_ptcb->deadline_time)
1262             {
1263                 printf("%2dt MissDeadline \t task(%2d) \t ----- \n", OSTime - 1, check_violate_ptcb->OSTCBId, check_violate_ptcb->num_times_job);
1264                 //printf("%2dt Aperiodic job(%d) MissDeadline \t (Server deadline) \t ----- \n", OSTime - 1, cus_fifo_q[cus_fifo_q_info->end].job_id);
1265                 fprintf(Output_fp, "%2dt MissDeadline \t task(%2d) \t ----- \n", OSTime - 1, check_violate_ptcb->OSTCBId, check_violate_ptcb->num_times_job);
1266                 //printf(Output_fp, "%2dt Aperiodic job(%d) MissDeadline \t (Server deadline) \t ----- \n", OSTime - 1, cus_fifo_q[cus_fifo_q_info->end].job_id);
1267             }
1268         }
1269     }
1270     // Consider whether aperiodic/sporadic job arrive.
1271     if (OSTime >= cus_fifo_q[cus_fifo_q_info->end].arrive_time) && (OSTime >= check_violate_ptcb->deadline_time)
1272     {
1273         // Compute CUS Server deadline.
1274         // Find the maximum start time (previous server deadline v.s. aperiodic/sporadic job's arrive time)
1275         INT16U maximum_start_time;
1276         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.
1277         {
1278             maximum_start_time = check_violate_ptcb->deadline_time;
1279         }
1280         else
1281         {
1282             maximum_start_time = cus_fifo_q[cus_fifo_q_info->end].arrive_time;
1283         }
1284     }
1285     // Compute server deadline of executing aperiodic/sporadic job.
1286     float append_time = ((float)cus_fifo_q[cus_fifo_q_info->end].execution_time) / (check_violate_ptcb->server_size);
1287     check_violate_ptcb->deadline_time = maximum_start_time + (INT16U)append_time;
1288
1289     // Context switch aperiodic/sporadic job and server task.
1290     check_violate_ptcb->arrive_time = cus_fifo_q[cus_fifo_q_info->end].arrive_time;
1291     check_violate_ptcb->num_recent_execute_time = 0;
1292     check_violate_ptcb->total_execute_time = cus_fifo_q[cus_fifo_q_info->end].execution_time;
1293
1294     // Add task 3 into EDF minimum heap.
1295     EDFHeapInsert(check_violate_ptcb->OSTCBId, check_violate_ptcb->deadline_time);
1296
1297 }

```

```

1297
1298     if (OSTime == cus_fifo_q[cus_fifo_q_info->end].arrive_time)
1299     {
1300         printf("%2dt 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);
1301         fprintf(Output_fp, "%2dt 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);
1302     }
1303     else
1304     {
1305         printf("%2dt 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);
1306         fprintf(Output_fp, "%2dt 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);
1307     }
1308     //printf("SHOW THE EDF HEAP :: \n");
1309     //for (int heap_id = 1; heap_id <= edf_heap_info->num_item; heap_id++)
1310     //{
1311     //    printf("%d<|d\n", heap_id, edf_heap[heap_id].task_id, edf_heap[heap_id].deadline);
1312     //}
1313
1314     // Remove aperiodic/sporadic job from CUS FIFO queue.
1315     cus_fifo_q_info->end++;
1316
1317     else
1318     {
1319         if (OSTime == cus_fifo_q[cus_fifo_q_info->end].arrive_time)
1320         {
1321             printf("%2dt Aperiodic job(%d) arrives. Do nothing. \n", OSTime, cus_fifo_q[cus_fifo_q_info->end].job_id, check_violate_ptcb->deadline_time);
1322             fprintf(Output_fp, "%2dt Aperiodic job(%d) arrives. Do nothing. \n", OSTime, cus_fifo_q[cus_fifo_q_info->end].job_id, check_violate_ptcb->deadline_time);
1323         }
1324     }
1325
1326 }
1327
1328 #endif /* M11102155_PA2_PART_2_CUS */
1329
1330     check_violate_ptcb = check_violate_ptcb->OSTCBNext;
1331 }

```

```

1331     check_violate_ptcb = check_violate_ptcb->OSTCBNext;
1332 }
1333
1334 #ifdef M11102155_PA2_PART_2_CUS
1335
1336     // USER DEFINED DEADLINE (ABSOLUTE DEADLINE) :::
1337     // Consider whether aperiodic/sporadic job miss there user define deadline (traverse CUS FIFO queue).
1338     int check_violate_q_ptr = cus_fifo_q[cus_fifo_q_info->end];
1339     while (check_violate_q_ptr != cus_fifo_q_info->front)
1340     {
1341         // Consider whether aperiodic/sporadic job miss absolute deadline.
1342         if (OSTime > cus_fifo_q[check_violate_q_ptr].user_define_deadline)
1343         {
1344             printf("%2dt Aperiodic job(%d) MissDeadline \t (Absolute deadline) \t ----- \n", OSTime - 1, cus_fifo_q[check_violate_q_ptr].job_id);
1345             check_violate_q_ptr++;
1346         }
1347     }
1348
1349 #endif /* M11102155_PA2_PART_2_CUS */
1350
1351
1352 #endif /* M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF */
1353

```

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

```

1355     if (OSRunning == OS_TRUE) {
1356         /* Setting the end time for the OS */
1357         if (OSTimeGet() > SYSTEM_END_TIME)
1358             {
1359 #ifndef M11102155_HW1
1360             // Close the output file.
1361             fclose(Output_fp);
1362 #endif /* M11102155_HW1 | M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF */
1363             OSRunning = OS_FALSE;
1364             exit(0);
1365         }
1366         /* Setting the end time for the OS */
1367 #ifndef OS_TICK_STEP_EN > 0u
1368         switch (OSTickStepState) {
1369             case OS_TICK_STEP_DIS:                                /* Determine whether we need to process a tick */
1370                 step = OS_TRUE;                                /* Yes, stepping is disabled */
1371                 break;
1372
1373             case OS_TICK_STEP_WAIT:                            /* No, waiting for uC/OS-View to set ... */
1374                 step = OS_FALSE;                            /* ... OSTickStepState to OS_TICK_STEP_ONCE */
1375                 break;
1376
1377             case OS_TICK_STEP_ONCE:                           /* Yes, process tick once and wait for next ... */
1378                 step = OS_TRUE;                            /* ... step command from uC/OS-View */
1379                 OSTickStepState = OS_TICK_STEP_WAIT;
1380                 break;
1381
1382             default:                                         /* Invalid case, correct situation */
1383                 step = OS_TRUE;
1384                 OSTickStepState = OS_TICK_STEP_DIS;
1385                 break;
1386         }
1387         if (step == OS_FALSE) {                                /* Return if waiting for step command */
1388             return;
1389         }
1390 #endif
1391         ptcb = OSTCBList;
1392         while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {        /* Point at first TCB in TCB list */
1393             /* Go through all TCBs in TCB list */
1394             OS_ENTER_CRITICAL();
1395             if (ptcb->OSTCBDly != 0u) {                         /* No, Delayed or waiting for event with TO */
1396                 ptcb->OSTCBDly--;                            /* Decrement nbr of ticks to end of delay */
1397                 if (ptcb->OSTCBTmo == 0u) {                      /* Check for timeout */
1398                     if ((ptcb->OSTCBStat & OS_STAT_PEND_ANY) != OS_STAT_RDY) {
1399                         ptcb->OSTCBStat &= ~INTUBUS_STAT_PEND_ANY;    /* Yes, Clear status flag */
1400                         ptcb->OSTCBstatPend = OS_STAT_PEND_TO;      /* Indicate PEND timeout */
1401                     } else {
1402                         ptcb->OSTCBstatPend = OS_STAT_PEND_OK;
1403                     }
1404                     if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) == OS_STAT_RDY) { /* Is task suspended? */
1405                         OSRdyGrp |= ptcb->OSTCBBitY;                  /* No, Make ready */
1406                         OSRdyTbl[ptcb->OSTCBY] = ptcb->OSTCBBitX;
1407 #endif /* M11102155_PA2_PART_1_EDF */
1408         }

```

```
1410
1411     #ifndef defined (M11102155_PA1_PART_2_RM) | defined (M11102155_PA2_PART_1_EDF)
1412
1413     #ifndef M11102155_PA2_PART_2_CUS
1414         if (!ptcb->server_or_not)
1415             #endif /* M11102155_PA2_PART_2_CUS */
1416
1417             //OS_ENTER_CRITICAL();
1418             ptcb->arrive_time = OSTime;
1419             ptcb->num_recent_execute_time = 0;
1420             ptcb->deadline_time = ptcb->arrive_time + ptcb->period;
1421
1422             if (edf_heap_info->num_item != edf_heap_info->size)
1423             {
1424                 EDFHeapInsert(ptcb->OSTCBid, ptcb->deadline_time);
1425             }
1426             else
1427             {
1428                 printf("ERROR : os_core.c ... EDF HEAP overflow !!!\n");
1429             }
1430             //OS_EXIT_CRITICAL();
1431
1432     #endif /* M11102155_PA1_PART_2_RM | M11102155_PA2_PART_1_EDF */
1433
1434         OS_TRACE_TASK_READY(ptcb);
1435
1436     }
1437
1438     ptcb = ptcb->OSTCBNext;
1439     OS_EXIT_CRITICAL();
1440 }
1441
1442 }
```

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
698     OS_CPU_SR cpu_sr = 0u;
699 #endif
700
701     if (OSRunning == OS_TRUE) {
702         OS_ENTER_CRITICAL(); /* Allocate storage for CPU status register */
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                 OS_SCHED_NESTING();
709                 OS_SchedNew();
710                 OSTCBHighRdy = OSTCBPrioTbl[OSPriorHighRdy];
711             }
712         }
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 (OSTCFCur->num_recent_execute_time == OSTCFCur->total_execute_time)
717             {
718                 OSPrioHighRdy = 63;
719                 OSTCBHighRdy = OSTCBPrioTbl[OSPriorHighRdy];
720             }
721         }
722     }
723     else
724     {
725         if (OSTCFCur->num_recent_execute_time == OSTCFCur->total_execute_time)
726         {
727             EDFHeapDelete();
728             OSPrioHighRdy = edf_heap[1].task_id;
729             OSTCBHighRdy = OSTCBPrioTbl[OSPriorHighRdy];
730         }
731         else
732         {
733             OSPrioHighRdy = edf_heap[1].task_id;
734             OSTCBHighRdy = OSTCBPrioTbl[OSPriorHighRdy];
735         }
736     }
737     OS_EXIT_CRITICAL();
738 }
739 #endif /* M11102155_PA2_PART_1_EDF */
740
741     if (OSPriorHighRdy != OSPrioCur) /* No Ctx Sw if current task is highest rdy */
742 #ifndef M11102155_HW1
743 #if OS_TASK_PROFILE_EN > 0u
744     OSTCBHighRdy->OSTCB_CtxSwCtr++;
745 #endif
746     OSTCBHighRdy++; /* Keep track of the number of ctx switches */
747 }

748 #if OS_TASK_CREATE_EXT_EN > 0u
749 #if defined(OS_TLS_TBL_SIZE) && (OS_TLS_TBL_SIZE > 0u)
750     OS_TLS_TaskSw();
751 #endif
752 #endif
753
754     OS_TRACE_ISR_EXIT_TO_SCHEDULER();
755
756 #ifndef M11102155_HW1
757 #if OS_TASK_PROFILE_EN > 0u
758     OSTCBHighRdy->OSTCB_CtxSwCtr++;
759 #endif
760     OSTCBHighRdy++; /* Keep track of the number of ctx switches */
761 }

762 #ifndef M11102155_PA1_PART_2_RM
763 #endif /* M11102155_PA1_PART_2_RM */

764 #ifndef M11102155_PA2_PART_1_EDF
765 //printf("TASK %d num_exe_time = %d, total = %d\n", OSTCBid, OSTCBCur->num_recent_execute_time, OSTCBCur->total_execute_time);
766     if (OSTCFCur->num_recent_execute_time == OSTCFCur->total_execute_time)
767     {
768         // Compute response time.
769         OSTCBCur->response_time = OSTime - OSTCBCur->arrive_time;
770
771         // Do delay !!!
772         INT32U ticks = OSTCBCur->deadline_time - OSTime;
773         //printf("TICK %d TASK %d's delay = %d\n", OSTime, OSTCBCur->OSTCBid, ticks);
774         if (ticks > 0u) {
775             // 0 means no delay!
776             OS_ENTER_CRITICAL();
777             OS_TRACE_TASK_SUSPENDED(OSTCBCur);
778             OSTCBCur->OSTCB_DLY_ticks = ticks; /* Load ticks in TCB */
779             OS_TRACE_TASK_DLY_ticks();
780             OS_EXIT_CRITICAL();
781         }
782         else
783         {
784             // If the task arrive after it complete immediately.
785             //printf("TICK %dahaha TASK %d\n", OSTime, OSTCBCur->OSTCBid);
786             OS_ENTER_CRITICAL();
787             OSTCBCur->arrive_time = OSTime;
788             OSTCBCur->num_recent_execute_time = 0;
789             OSTCBCur->deadline_time = OSTCBCur->arrive_time + OSTCBCur->period;
790
791             if (edf_heap_info->num_item != edf_heap_info->size)
792             {
793                 EDFHeapInsert(OSTCBCur->OSTCBid, OSTCBCur->deadline_time);
794             }
795             else
796             {
797                 printf("ERROR : os_core.c .... EDF HEAP overflow !!!\n");
798             }
799         }
800     }
801     OS_EXIT_CRITICAL();
802 }
803 #ifndef M11102155_PA2_PART_2_CUS
804     if (!OSTCBCur->server_or_not)
805     {
806 #endif /* M11102155_PA2_PART_2_CUS */

```

```

860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953

```

M11102155_PA2_PART_2_CUS

```

if (OSTCBHighRdy->OSTCBPrio == 63)
{
    printf("%2d\t Completion \t task(%2d)\n%2d \t task(%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->OSTCBId);
    fprintf(Output_fp, "%2d\t Completion \t task(%2d)\n%2d \t task(%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->OSTCBId);
    OSTCBCur->num_times_job++;
}
else
{
    printf("%2d\t Completion \t task(%2d)\n%2d \t task(%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->OSTCBId);
    fprintf(Output_fp, "%2d\t Completion \t task(%2d)\n%2d \t task(%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->OSTCBId);
    OSTCBCur->num_times_job++;
}

// AS the aperiodic/periodic job completed, we must reset the record in CUS (server task).
// AS the aperiodic/periodic job completed, we must reset the record in CUS (server task).
// print informations of context switch.
if (OSTCBHighRdy->OSTCBPrio == 63)

{
    printf("%2d\t Completion \t task(%2d)\n%2d \t task(%2d) \t %2d \t %2d \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)\n%2d \t task(%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->num_times_job++;
}
else

{
    printf("%2d\t Completion \t task(%2d)\n%2d \t task(%2d) \t %2d \t %2d \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)\n%2d \t task(%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->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)\n%2d \t task(%2d) \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63);
        fprintf(Output_fp, "%2d\t Preemption \t task(%2d)\n%2d \t task(%2d) \n", OSTime, OSTCBCur->OSTCBId, OSTCBCur->num_times_job, 63);
    }
}

```

```

926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953

```

```

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

endif /* M11102155_PA2_PART_1_EDF */

OSIntCtxSw();
} else {
    OS_TRACE_ISR_EXIT();
}
} else {
    OS_TRACE_ISR_EXIT();
}
} else {
    OS_TRACE_ISR_EXIT();
}
OS_EXIT_CRITICAL();

```