Embedded OS Implementation, HW2

Note1: We need to set #define OS_TICKS_PER_SEC to 1u make sure 1 tick/sec in os_cfg.h

Note2: We need to set exit(0) and OSRunning = OS_FALSE when OSTimeGet() > SYSTEM_END_TIME in os_core.c

Note3: Set #define SYSTEM_END_TIME 40

[PART I] EDF Scheduler Implementation

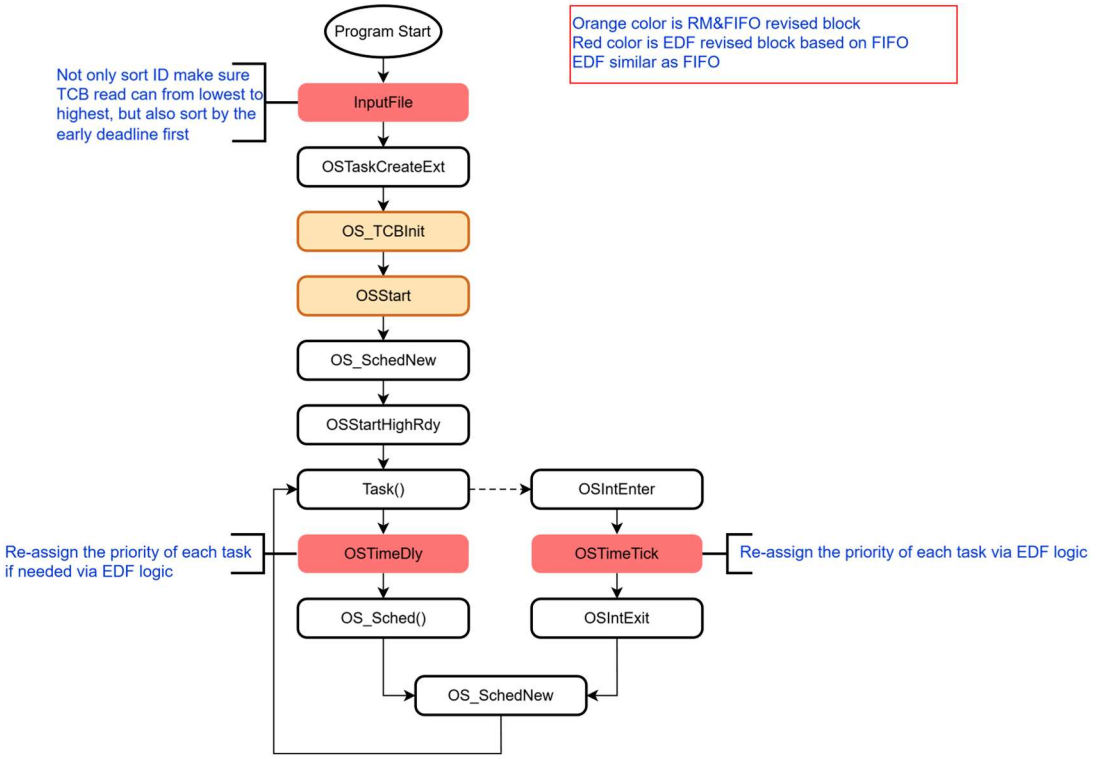
Output for example task set 1 = {τ1 (1, 0, 2, 6), τ2 (2, 0, 5, 9)}

```
 2      Completion      task( 1)( 0)    task( 2)( 0)        2           0           4
 7      Completion      task( 2)( 0)    task( 1)( 1)        7           2           2
 9      Completion      task( 1)( 1)    task( 2)( 1)        3           1           3
12      Preemption      task( 2)( 1)    task( 1)( 2)
14      Completion      task( 1)( 2)    task( 2)( 1)        2           0           4
16      Completion      task( 2)( 1)    task(63)            7           2           2
18      Preemption      task(63)        task( 1)( 3)
20      Completion      task( 1)( 3)    task( 2)( 2)        2           0           4
25      Completion      task( 2)( 2)    task( 1)( 4)        7           2           2
27      Completion      task( 1)( 4)    task( 2)( 3)        3           1           3
30      Preemption      task( 2)( 3)    task( 1)( 5)
32      Completion      task( 1)( 5)    task( 2)( 3)        2           0           4
34      Completion      task( 2)( 3)    task(63)            7           2           2
36      Preemption      task(63)        task( 1)( 6)
38      Completion      task( 1)( 6)    task( 2)( 4)        2           0           4
```

Output for example task set 2 = {τ1 (1, 0, 2, 4), τ2 (2, 0, 4, 7)}

```
 2      Completion      task( 1)( 0)    task( 2)( 0)        2           0           2
 6      Completion      task( 2)( 0)    task( 1)( 1)        6           2           1
 8      Completion      task( 1)( 1)    task( 1)( 2)        4           2           0
10      Completion      task( 1)( 2)    task( 2)( 1)        2           0           2
14      Completion      task( 2)( 1)    task( 1)( 3)        7           3           0
16      Completion      task( 1)( 3)    task( 1)( 4)        4           2           0
18      Completion      task( 1)( 4)    task( 2)( 2)        2           0           2
21      MissDeadline    task( 2)( 2)    --------------------
```

Implement EDF scheme



Because the EDF (Earliest Deadline First) implementation is largely similar to the FIFO (First-In-First-Out) practice, I will only highlight the differences between EDF and FIFO. First, as shown in the scheme above, refer to line 185 in the InputFile() function

of app_hook.c. In FIFO, the priority is initialized based on the FIFO order. In contrast, in EDF, the priority must be initialized based on the EDF algorithm.

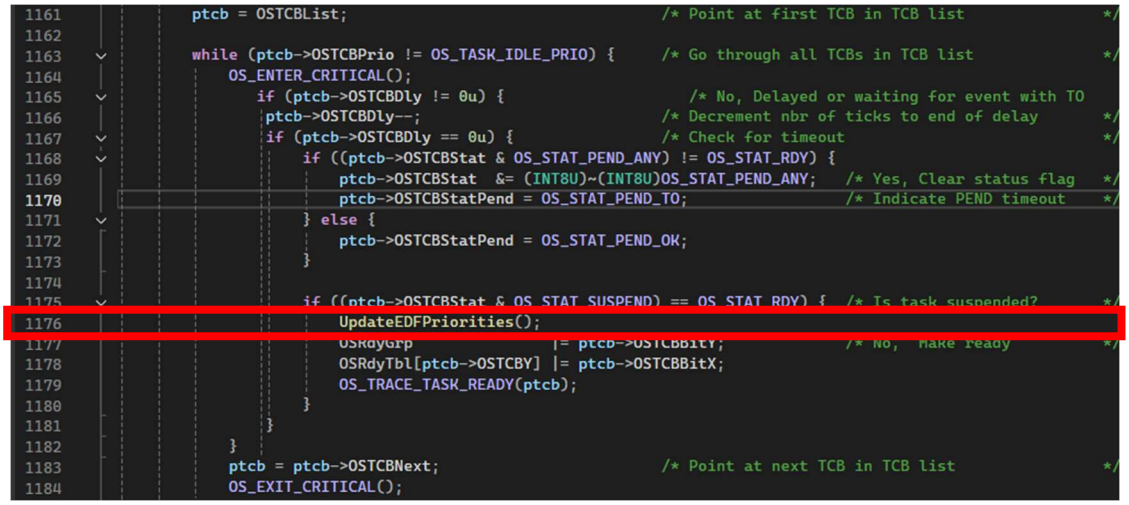
```
171                 else if (i == 2)
172                     TaskParameter[j].TaskExecutionTime = TaskInfo[i];
173                 else if (i == 3)
174                     TaskParameter[j].TaskPeriodic = TaskInfo[i];
175                 i++;
176             }
177
178             // Initial Task Counter
179             TaskParameter[j].TaskNumber = 0;
180             TaskParameter[j].TaskPriority = j;
181             j++;
182         }
183         fclose(fp);
184         // Assign the initial priority for each task
185         PriorityAssignmentInit(TaskParameter, TASK_NUMBER);
186         // Make sure task order is start from lowest in TCB
187         ReverseTaskParameter();
188     }
189
```

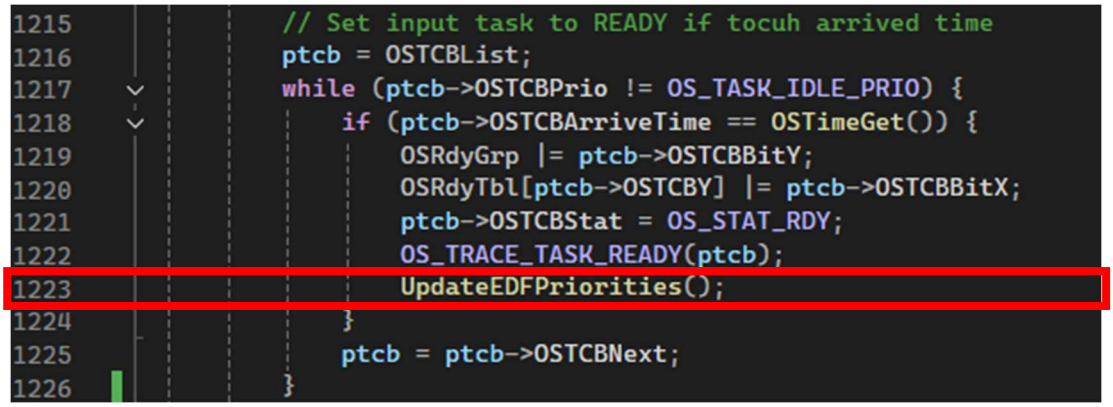
From lines 123 to 131, I initialize each task's priority using a quicksort algorithm, with the compare function defined in lines 113 to 120. The compare function sorts tasks based on their deadlines. Additionally, if tasks have the same priority, I compare task IDs to ensure that the task with the smallest ID is prioritized during initialization use bubble sort from lines 134 to 143.

```
113     // Comparison function, used for qsort sorting
114     int compareTasksInit(const void* a, const void* b) {
115         task_para_set* taskA = (task_para_set*)a;
116         task_para_set* taskB = (task_para_set*)b;
117         int deadlineA = taskA->TaskArriveTime + taskA->TaskPeriodic;
118         int deadlineB = taskB->TaskArriveTime + taskB->TaskPeriodic;
119         // Sort by deadline (smaller deadlines first)
120         return deadlineA - deadlineB;
121     }
122
123     void PriorityAssignmentInit(task_para_set* TaskParameter, int TASK_NUMBER) {
124         // Sort by deadline (EDF)
125         qsort(TaskParameter, TASK_NUMBER, sizeof(task_para_set), compareTasksInit);
126
127         // Assign a unique priority (0 to TASK_NUMBER)
128         for (int i = 0; i < TASK_NUMBER; i++) {
129             // Priority starts at 0, with 0 being the highest priority.
130             TaskParameter[i].TaskPriority = i;
131         }
132
133         // Use bubble sort to order structure by TaskID from smallest to bigger
134         task_para_set temp;
135         for (int i = 0; i < TASK_NUMBER - 1; i++) {
136             for (int j = i + 1; j < TASK_NUMBER; j++) {
137                 if (TaskParameter[i].TaskID > TaskParameter[j].TaskID) {
138                     temp = TaskParameter[i];
139                     TaskParameter[i] = TaskParameter[j];
140                     TaskParameter[j] = temp;
141                 }
142             }
143         }
144     }
145
```

Once the system begins scheduling, it is fully operational. Next, we need to consider

assigning EDF priorities for each task during runtime, similar to the previous FIFO report. There are three locations where we must assign new priorities to each task. The three locations for assigning new EDF priorities are as follows. First, when a task is ready to execute after being delayed. Second, when a task arrives after the system has started running. Third, when a task is completed and does not require further delay. Focusing on the first two locations, similar to the FIFO report, we update task priorities in OSTimeTick() at line 1176 and line 1223 in os_core.c. Instead of using a simple approach to update task priorities, I created a new function to update each task's priority based on EDF logic.





Then I implemented the UpdateEDFPriorities() function in os_core.c from lines 1023 to 1065. This function enables the OS to evaluate each task's deadline and assign new priorities to tasks during runtime. I not only need to clear priority but also assign new priority for each task by using this function.

```
1023  // Update all task's priority via EDF
1024  void UpdateEDFPriorities(void) {
1025      OS_TCB* ptcb;
1026      EDFEntry  EDF_deadline_order[MAX];
1027      int current_task = OSTCBCur->OSTCBId;
1028      int count = 0;
1029      // Create EDF list
1030      ptcb = OSTCBList;
1031      while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {
1032          EDFEntry entry;
1033          // Calculate deadline
1034          entry.id = ptcb->OSTCBId;
1035          entry.deadline = ptcb->OSTCBArriveTime + ptcb->OSTCBPeriodic * ptcb->OSTCBNumber + ptcb->OSTCBPeriodic;
1036          EDF_deadline_order[count++] = entry;
1037          ptcb = ptcb->OSTCBNext;
1038      }
1039      // First sort by deadline from small to large. If the deadlines are the same, sort by id from small to large.
1040      qsort(EDF_deadline_order, count, sizeof(EDFEntry), compare_tasks_via_deadline);
1041
1042      // Update the priority of each task
1043      for (int i = 0; i < count; i++) {
1044          EDF_deadline_order[i].newpriority = i;
1045      }
1046
1047      // Reset all task's priority
1048      ClearAllTaskPriority();
1049      // Assign the priority of each task
1050      OS_TCB* taskList[MAX];
1051      ptcb = OSTCBList;
1052      int numTasks = 0;
1053      int i = 0;
1054
1055      while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {
1056          taskList[numTasks++] = ptcb;
1057          ptcb = ptcb->OSTCBNext;
1058      }
1059
1060      for (int j = 0; j < numTasks; j++) {
1061          // Change the priority of each task according to the sorted priority
1062          OSTaskChangePrio(taskList[EDF_deadline_order[j].id - 1]->OSTCBPrio, EDF_deadline_order[j].newpriority);
1063      }
1064
1065  }
```

Also, I added the compare function for the quicksort function from lines 1011 to 1021.

```
1010  // Comparison function: sort by deadline, if the same, sort by id (from small to large)
1011  int compare_tasks_via_deadline(const void* a, const void* b) {
1012      EDFEntry* task1 = (EDFEntry*)a;
1013      EDFEntry* task2 = (EDFEntry*)b;
1014
1015      if (task1->deadline != task2->deadline) {
1016          return task1->deadline - task2->deadline;  // Sort by deadline
1017      }
1018      else {
1019          return task1->id - task2->id;  // If deadlines are the same, sort by id in descending order
1020      }
1021  }
```

Because I want to use the OSTaskChangePrio() function, but if multiple tasks have the same priority, the OS cannot assign a new priority to a specific task. Therefore, I have implemented the ClearAllTaskPriority() function. To maintain the tasks' priorities and consider the scheduling logic carefully, when the OS changes the priority, it also needs to consider the previous priorities to maintain the overall order. In this approach, I collect all tasks in the TCB and process them from lines 978 to 1007. The structure I use, announced from lines 970 to 974, helps me maintain the new priorities in UpdateEDFPriorities().

```c
typedef struct {
    INT16U id;
    INT16U deadline;
    INT16U newpriority;
} EDFEntry;


// Clear all task's priority
void ClearAllTaskPriority(void) {
    OS_TCB* ptcb = OSTCBList;
    OS_TCB* taskList[MAX];
    int numTasks = 0;

    // Collect all tasks
    while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {
        taskList[numTasks++] = ptcb;
        ptcb = ptcb->OSTCBNext;
    }

    // Sort by priority (high to low)
    for (int i = 0; i < numTasks - 1; i++) {
        for (int j = i + 1; j < numTasks; j++) {
            if (taskList[i]->OSTCBPrio < taskList[j]->OSTCBPrio) {
                // Swap task locations
                OS_TCB* temp = taskList[i];
                taskList[i] = taskList[j];
                taskList[j] = temp;
            }
        }
    }

    // Adjust the priority of tasks according to the order of priority
    int i = 1;
    for (int j = 0; j < numTasks; j++) {
        // Change the priority of each task according to the sorted priority
        OSTaskChangePrio(taskList[j]->OSTCBPrio, OS_TASK_IDLE_PRIO - i++);
    }
}
```

For the third location, we need to check OSTimeDly() in os_time.c at line 87. Since I already created a new function to assign new priorities, we can simply use it here.

```c
    //  Change tick to remain time for the current task
    if (remain_time > 0u) {                          /* 0 means no delay!                    */
        OS_ENTER_CRITICAL();
        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;
        }
        OSTCBCur->OSTCBDly = remain_time;            /* Load ticks in TCB                    */
        OS_TRACE_TASK_DLY(remain_time);
        OS_EXIT_CRITICAL();
        OS_Sched();                                  /* Find next task to run!               */
    }
    else {
        UpdateEDFPriorities();
    }
}
```

Implement and describe how to handle the deadline missing situation under EDF.

Implement and describe how to handle the deadline missing situation under EDF. After all, OS need to handle miss deadline when the calculated time is weird (just

need to know if task can finish it execution time in its period). Here, like RM and FIFO, from line 1196 to 1213 I added, I use the variable CheckDlyTime in os_core.c OSTimeTick() function to detect it, just when task running over its period, program stop the OS and record the miss deadline.

```
1196        //Check deadline in each delay time of task
1197        ptcb = OSTCBList;
1198        while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {
1199            int CheckDlyTime;
1200            CheckDlyTime = ptcb->OSTCBPeriodic + ptcb->OSTCBArriveTime + ptcb->OSTCBNumber * ptcb->OSTCBPeriodic - OSTimeGet();
1201            if ((int)(CheckDlyTime) < 0 && CurrentTaskID != 65535) {
1202                printf("%2d\tMissDeadline\t  task(%2d)(%2d)\t  ------------\n",
1203                    OSTimeGet() - 1, ptcb->OSTCBId, ptcb->OSTCBNumber);
1204                if (Output_err = fopen_s(&Output_fp, OUTPUT_FILE_NAME, "a") == 0) {
1205                    fprintf(Output_fp, "%2d\tMissDeadline\t  task(%2d)(%2d)\t  ------------\n",
1206                        OSTimeGet() - 1, ptcb->OSTCBId, ptcb->OSTCBNumber);
1207                    fclose(Output_fp);
1208                }
1209                OSRunning = OS_FALSE;
1210                exit(0);
1211            }
1212            ptcb = ptcb->OSTCBNext;
1213        }
```

[PART II] CUS Scheduler Implementation

Output for example task set 1 = {τ1 (1, 0, 2, 8), τ2 (2, 0, 3, 10), τ3 (3, 0, 4, 15), τ4_ServerSize (4, 25%)}

Aperiodic Jobs Set = {j0 (0, 12, 3, 25), j1 (1, 14, 2, 33)}

```
 2      Completion        task( 1)( 0)    task( 2)( 0)          2               0               6
 5      Completion        task( 2)( 0)    task( 3)( 0)          5               2               5
 9      Completion        task( 3)( 0)    task( 1)( 1)          9               5               6
11      Completion        task( 1)( 1)    task( 2)( 1)          3               1               5
12      Aperiodic job(0) arrives and sets CUS server's deadline as 24.
14      Completion        task( 2)( 1)    task( 4)( 0)          4               1               6
14      Aperiodic job(1) arrives. Do nothing.
16      Preemption        task( 4)( 0)    task( 1)( 2)
18      Completion        task( 1)( 2)    task( 4)( 0)          2               0               6
19      Aperiodic job(0) is finished.
19      Completion        task( 4)( 0)    task( 3)( 1)          7               4               N/A
20      Preemption        task( 3)( 1)    task( 2)( 2)
23      Completion        task( 2)( 2)    task( 3)( 1)          3               0               7
24      Aperiodic job(1) sets CUS server's deadline as 32.
26      Completion        task( 3)( 1)    task( 1)( 3)          11              7               4
28      Completion        task( 1)( 3)    task( 4)( 1)          4               2               4
30      Aperiodic job(1) is finished.
30      Completion        task( 4)( 1)    task( 2)( 3)          16              4               N/A
32      Preemption        task( 2)( 3)    task( 1)( 4)
34      Completion        task( 1)( 4)    task( 2)( 3)          2               0               6
35      Completion        task( 2)( 3)    task( 3)( 2)          5               2               5
39      Completion        task( 3)( 2)    task(63)              9               5               6
40      Preemption        task(63)        task( 1)( 5)
```

Output for example task set 2 = {τ1 (1, 0, 2, 4), τ2 (2, 0, 3, 9), τ3_ServerSize (3, 30%)}

Aperiodic Jobs Set = {j0 (0, 1, 3, 24), j1 (1, 11, 2, 39)}

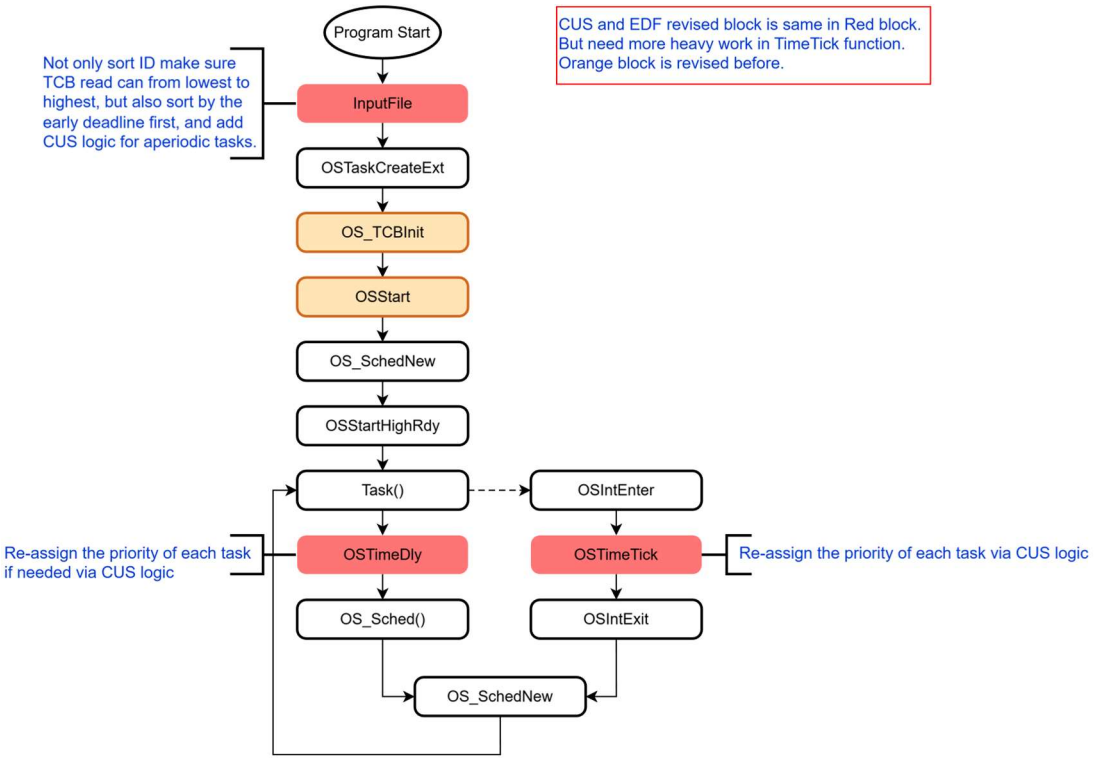
```
 1      Aperiodic job(0) arrives and sets CUS server's deadline as 11.
 2      Completion        task( 1)( 0)    task( 2)( 0)          2               0               2
 4      Preemption        task( 2)( 0)    task( 1)( 1)
 6      Completion        task( 1)( 1)    task( 2)( 0)          2               0               2
 7      Completion        task( 2)( 0)    task( 3)( 0)          7               4               2
10      Aperiodic job(0) is finished.
10      Completion        task( 3)( 0)    task( 1)( 2)          9               6               N/A
11      Aperiodic job(1) arrives and sets CUS server's deadline as 17.
12      Completion        task( 1)( 2)    task( 1)( 3)          4               2               0
14      Completion        task( 1)( 3)    task( 3)( 1)          2               0               2
16      Aperiodic job(1) is finished.
16      Completion        task( 3)( 1)    task( 2)( 1)          5               3               N/A
18      MissDeadline      task( 2)( 1)    --------------------
```

Output for example task set 3 = {τ1 (1, 0, 3, 6), τ2 (2, 0, 4, 10), τ3_ServerSize (3, 10%)}

Aperiodic Jobs Set = {j0 (0, 7, 1, 23), j1 (1, 10, 2, 25)}

```
 3      Completion        task( 1)( 0)    task( 2)( 0)          3              0              3
 7      Completion        task( 2)( 0)    task( 1)( 1)          7              3              3
 7      Aperiodic job(0) arrives and sets CUS server's deadline as 17.
10      Completion        task( 1)( 1)    task( 3)( 0)          4              1              2
10      Aperiodic job(1) arrives. But can't scheduled. (miss absolute deadline)
11      Aperiodic job(0) is finished.
11      Completion        task( 3)( 0)    task( 2)( 1)          4              3              N/A
12      Preemption        task( 2)( 1)    task( 1)( 2)
15      Completion        task( 1)( 2)    task( 2)( 1)          3              0              3
18      Completion        task( 2)( 1)    task( 1)( 3)          8              4              2
21      Completion        task( 1)( 3)    task( 2)( 2)          3              0              3
24      Preemption        task( 2)( 2)    task( 1)( 4)
27      Completion        task( 1)( 4)    task( 2)( 2)          3              0              3
28      Completion        task( 2)( 2)    task(63)              8              4              2
30      Preemption        task(63)        task( 1)( 5)
33      Completion        task( 1)( 5)    task( 2)( 3)          3              0              3
37      Completion        task( 2)( 3)    task( 1)( 6)          7              3              3
40      Completion        task( 1)( 6)    task( 2)( 4)          4              1              2
```

Implement CUS scheme



First, unlike in pure EDF, CUS requires comparing tasks based not only on their EDF deadlines but also on their CUS information provided in the input file. In ucon_ii.h, I added the necessary data fields to support CUS deadline calculation—for example, at line 77, and from lines 93 to 96, 101, 102, and 106 to 108. These newly added parameters are specifically used for computing the CUS deadlines of aperiodic tasks. And help me modify the cross function to use some parameters of aperiodic tasks.

```c
#define APERIODIC_FILE_NAME "./Aperiodicjobs.txt"
#define MAX 20
#define INFO 4
#define COMPLETION 0x01u
#define PREEMPTION 0x02u
// Output file
FILE* Output_fp;
errno_t Output_err;

typedef struct task_para_set {
    INT16U TaskID;
    INT16U TaskArriveTime;
    INT16U TaskExecutionTime;
    INT16U TaskPeriodic;
    INT16U TaskNumber;
    INT16U TaskPriority;
    INT16U ApeirodicTaskID;
    INT16U ServerUtility;
    INT16U AbsoluteDeadline;
    INT16U CalculatedDeadline;
} task_para_set;
// number of input task
int TASK_NUMBER;
// number of aperiodic task
int APERIODIC_TASK_NUMBER;
// Dynamic create the stack size
OS_STK** Task_STK;
// Create task
task_para_set TaskParameter[OS_MAX_TASKS];
task_para_set AperiodicTaskParameter[OS_MAX_TASKS];
// For record aperiodic task information cross functions
extern task_para_set AperiodicTaskCrossFunctionParameter[OS_MAX_TAS
```

Now, let's move on to app_hook.c to see how these parameters are used to calculate the CUS deadline. In the InputFile() function, from line 308 to line 320, I handle the initialization of aperiodic tasks and set the necessary parameters for the CUS server. This block initializes the server's parameters using the first aperiodic task's data, and for every aperiodic task that arrives at time 0, it sets the initial CUS deadline and logs it to the output file.

```c
    if (TASK_NUMBER > 0) {
        TaskParameter[TASK_NUMBER - 1].ServerUtility = TaskParameter[TASK_NUMBER - 1].TaskArriveTime;
        InitializeFirstAperiodicTask();
        if (APERIODIC_TASK_NUMBER > 0) {
            TaskParameter[TASK_NUMBER - 1] = AperiodicTaskParameter[0];
        }
    }
    // Assign the initial priority for each task
    PriorityAssignmentInit(TaskParameter, TASK_NUMBER);

    for (int i = 0; i < APERIODIC_TASK_NUMBER; i++) {
        AperiodicTaskCrossFunctionParameter[i] = AperiodicTaskParameter[i];
    }

    // Make sure task order is start from lowest in TCB
    ReverseTaskParameter();
}
```

Lines 220 to 256 define the InitializeFirstAperiodicTask function. This function reads a file containing aperiodic task information, initializes task parameters, and assigns priorities based on the calculated deadlines. It also sorts the tasks by their arrival

times and ensures proper priority assignment for scheduling. Note that line 258 is used to store parameters for the cross function.

```
220  void InitializeFirstAperiodicTask(void) {
221      errno_t err;
222      if ((err = fopen_s(&fp, APERIODIC_FILE_NAME, "r")) == 0) {}
223      char str[MAX];
224      char* ptr;
225      char* pTmp = NULL;
226      int TaskInfo[INFO], i, j = 0;
227      APERIODIC_TASK_NUMBER = 0;
228      while (!feof(fp))
229      {
230          i = 0;
231          memset(str, 0, sizeof(str));
232          fgets(str, sizeof(str) - 1, fp);
233          ptr = strtok_s(str, " ", &pTmp);
234          while (ptr != NULL)
235          {
236              TaskInfo[i] = atoi(ptr);
237              ptr = strtok_s(NULL, " ", &pTmp);
238              if (i == 0) {
239                  AperiodicTaskParameter[j].TaskID = TASK_NUMBER;
240                  AperiodicTaskParameter[j].ApeirodicTaskID = APERIODIC_TASK_NUMBER++;
241              }
242              else if (i == 1)
243                  AperiodicTaskParameter[j].TaskArriveTime = TaskInfo[i];
244              else if (i == 2)
245                  AperiodicTaskParameter[j].TaskExecutionTime = TaskInfo[i];
246              else if (i == 3)
247                  AperiodicTaskParameter[j].AbsoluteDeadline = TaskInfo[i];
248              i++;
249          }
250          AperiodicTaskParameter[j].TaskPriority = j;
251          j++;
252      }
253      qsort(AperiodicTaskParameter, APERIODIC_TASK_NUMBER, sizeof(task_para_set), compareCUSTasksInit);
254
255      GivePriorityForAperiodicTask(APERIODIC_TASK_NUMBER);
256  }
257
258  task_para_set AperiodicTaskCrossFunctionParameter[OS_MAX_TASKS];
```

Next, first see opt_div (line 169 to 172) is used the function in GivePriorityForAperiodicTask from line 174 to 218 assigns priorities to aperiodic tasks based on their arrival times and calculated deadlines. It ensures that tasks are sorted and scheduled in a way that respects their deadlines and utility constraints. The function also adjusts deadlines dynamically to avoid conflicts with previously scheduled tasks from 201 to 212.

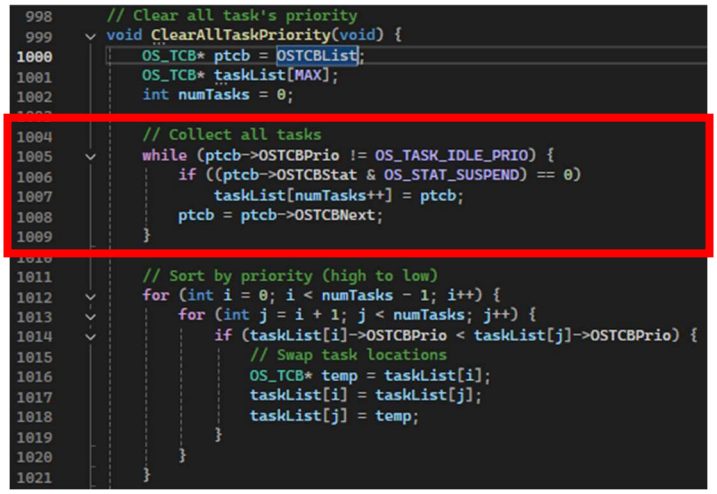
```
169  int opt_div(int a, int b) {
170      float result = (float)(a) / (float)(b) * 100;
171      return (int)result;
172  }
```

```
void GivePriorityForAperiodicTask(int aperiodic_task_number) {
    int serverUtility = TaskParameter[TASK_NUMBER - 1].ServerUtility;
    for (int i = 0; i < aperiodic_task_number; i++) {
        // Simulate a CUS deadline based on utility: Deadline = ArrivalTime + ExecutionTime / Utility
        AperiodicTaskParameter[i].TaskPeriodic = CeilDiv(AperiodicTaskParameter[i].TaskExecutionTime, serverUtility);
        AperiodicTaskParameter[i].CalculatedDeadline =
            AperiodicTaskParameter[i].TaskArriveTime + AperiodicTaskParameter[i].TaskPeriodic;
    }
    // Then sort according to the CalculatedDeadline arrival time
    task_para_set temp;
    for (int i = 0; i < aperiodic_task_number - 1; i++) {
        for (int j = i + 1; j < aperiodic_task_number; j++) {
            if (AperiodicTaskParameter[i].TaskArriveTime > AperiodicTaskParameter[j].TaskArriveTime) {
                temp = AperiodicTaskParameter[i];
                AperiodicTaskParameter[i] = AperiodicTaskParameter[j];
                AperiodicTaskParameter[j] = temp;
            }
        }
    }

    for (int i = 0; i < aperiodic_task_number; i++) {
        if (i == 0) {
            AperiodicTaskParameter[i].CalculatedDeadline =
                AperiodicTaskParameter[i].TaskArriveTime + AperiodicTaskParameter[i].TaskPeriodic;
        }
        else {
            // It is necessary to determine that the arrive time is less than or equal to the CUS deadline before the previous CUS deadline is used.
            if (AperiodicTaskParameter[i - 1].CalculatedDeadline >= AperiodicTaskParameter[i].TaskArriveTime) {
                int j = i - 1;
                while (j >= 0) {
                    int prevDeadline = AperiodicTaskParameter[j].CalculatedDeadline;
                    int prevAbsDeadline = AperiodicTaskParameter[j].AbsoluteDeadline;

                    if (prevAbsDeadline >= prevDeadline) {
                        AperiodicTaskParameter[i].CalculatedDeadline = prevDeadline + AperiodicTaskParameter[i].TaskPeriodic;
                        break; // When you find the one that meets the conditions, it ends.
                    }

                    j--; // Continue to search for earlier tasks
                }
            }
        }
    }

}
```

In the compareTasksInit function from line 113 to128 is used to sort tasks based on their deadlines during initialization. For periodic tasks, the deadline is calculated as the sum of arrival time and period. For the special aperiodic task (identified by TASK_NUMBER), the deadline is derived from its calculated deadline value. The function ensures that tasks with earlier deadlines are prioritized. If two tasks have the same deadline, the task with the smaller TaskID is given higher priority. And the compareCUSTasksInit function from line 132 to 142 is designed to sort custom aperiodic tasks based on their arrival times. It is typically used in scenarios where tasks must be ordered by their arrival sequence rather than deadline. If two tasks arrive at the same time, the task with the smaller ApeirodicTaskID is prioritized.

```
int compareTasksInit(const void* a, const void* b) {
    task_para_set* taskA = (task_para_set*)a;
    task_para_set* taskB = (task_para_set*)b;
    int deadlineA = taskA->TaskArriveTime + taskA->TaskPeriodic;
    int deadlineB = taskB->TaskArriveTime + taskB->TaskPeriodic;
    if (taskA->TaskID == TASK_NUMBER)
        deadlineA = taskA->TaskArriveTime + taskA->CalculatedDeadline;
    if (taskB->TaskID == TASK_NUMBER)
        deadlineB = taskB->TaskArriveTime + taskB->CalculatedDeadline;
    // Sort by deadline (smaller deadlines first)
    if (deadlineA != deadlineB) {
        return deadlineA - deadlineB; // Deadline Smaller ones go first
    }
    else {
        return taskA->TaskID - taskB->TaskID; // The ID with the smallest number goes first
    }
}


int compareCUSTasksInit(const void* a, const void* b) {
    task_para_set* taskA = (task_para_set*)a;
    task_para_set* taskB = (task_para_set*)b;
    int deadlineA = taskA->TaskArriveTime;
    int deadlineB = taskB->TaskArriveTime;
    // Sort by deadline (smaller deadlines first)
    if (deadlineA != deadlineB) {
        return deadlineA - deadlineB; // Deadline Smaller ones go first
    }
    else {
        return taskA->ApeirodicTaskID - taskB->ApeirodicTaskID; // The ID with the smallest number goes first
    }
}
```

Back in os_core.c, for CUS deadline scheduling, we need to ensure that additional checks are in place to maintain correct scheduling behavior. Let's first look at the ClearAllTaskPriority function. Since aperiodic tasks can complete and be suspended to avoid redundant executions, we need to be careful not to modify the priority of suspended tasks when rescheduling all tasks using kernel functions. Therefore, I added a condition between lines 1004 and 1009 to filter out suspended tasks during priority resetting. Other part in this function is same as EDF.



```
998     // Clear all task's priority
999     void ClearAllTaskPriority(void) {
1000        OS_TCB* ptcb = OSTCBList;
1001        OS_TCB* taskList[MAX];
1002        int numTasks = 0;
1003
1004        // Collect all tasks
1005        while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {
1006            if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) == 0)
1007                taskList[numTasks++] = ptcb;
1008            ptcb = ptcb->OSTCBNext;
1009        }
1010
1011        // Sort by priority (high to low)
1012        for (int i = 0; i < numTasks - 1; i++) {
1013            for (int j = i + 1; j < numTasks; j++) {
1014                if (taskList[i]->OSTCBPrio < taskList[j]->OSTCBPrio) {
1015                    // Swap task locations
1016                    OS_TCB* temp = taskList[i];
1017                    taskList[i] = taskList[j];
1018                    taskList[j] = temp;
1019                }
1020            }
1021        }
```

Due to the addition of suspended tasks and CUS logic, the UpdateEDFPriorities function also needs to be modified to prevent repeatedly modifying the priority of suspended tasks. Specifically, lines 1056 to 1060 introduce a condition to determine when to use the CUS deadline. Similarly, lines 1067 to 1075 apply this logic to adjust the task priorities accordingly. Finally, lines 1087 to 1091 use OSTaskChangePrio to ensure that the task priorities are updated to reflect the correct order expected by CUS scheduling.

```
1044    // Update all task's priority via CUS
1045    void UpdateEDFPriorities(void) {
1046        OS_TCB* ptcb;
1047        EDFEntry  EDF_deadline_order[MAX];
1048        int count = 0;
1049        int current_susepnd_task_priority = -1;
1050        // Create CUS list
1051        ptcb = OSTCBList;
1052        while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {
1053            if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) == 0) { // Only add READY/RUNNING tasks
1054                EDFEntry entry;
1055                entry.id = ptcb->OSTCBId;
1056                if (ptcb->OSTCBId != TASK_NUMBER)
1057                    entry.deadline = ptcb->OSTCBArriveTime + ptcb->OSTCBPeriodic * ptcb->OSTCBNumber + ptcb->OSTCBPeriodic;
1058                else
1059                    entry.deadline = ptcb->OSTCBCalculatedDeadline;
1060                EDF_deadline_order[count++] = entry;
1061            }
1062            else {
1063                current_susepnd_task_priority = ptcb->OSTCBPrio;
1064            }
1065            ptcb = ptcb->OSTCBNext;
1066        }
1067        // First sort by deadline from small to large. If the deadlines are the same, sort by id from small to large.
1068        qsort(EDF_deadline_order, count, sizeof(EDFEntry), compare_tasks_via_deadline);
1069        // Update the priority of each task
1070        for (int i = 0; i < count; i++) {
1071            if (current_susepnd_task_priority <= i)
1072                EDF_deadline_order[i].newpriority = i + 1;
1073            else
1074                EDF_deadline_order[i].newpriority = i;
1075        }
1076        // Reset all task's priority
1077        ClearAllTaskPriority();
1078        // Assign the priority of each task
1079        OS_TCB* taskList[MAX];
1080        ptcb = OSTCBList;
1081        int numTasks = 0;
1082        int i = 0;
1083        while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {
1084            if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) == 0)
1085                taskList[numTasks++] = ptcb;
1086            ptcb = ptcb->OSTCBNext;
1087        }
1088        for (int j = 0; j < numTasks; j++)
1089            // Change the priority of each task according to the sorted priority
1090            OSTaskChangePrio(taskList[EDF_deadline_order[j].id - 1]->OSTCBPrio, EDF_deadline_order[j].newpriority);
1091    }
```

Moreover, I added a function to handle the case when the time tick is 0. We need to initialize the aperiodic task first. Please refer to lines 1093 to 1135, which help the OS initialize the task for later use.

```
1093    void taskEnterCheckMissabsAndArrSetDeadline(void) {
1094        ...
1095        OS_TCB* ptcb;
1096        while (AperiodicTaskCrossFunctionParameter[ArrivedAperiodicJobCounter].TaskArriveTime == 0 &&
1097            AperiodicTaskCrossFunctionParameter[ArrivedAperiodicJobCounter].CalculatedDeadline  >  0) {
1098            // Check if the aperiodic task is missed absolute deadline
1099            if (AperiodicTaskCrossFunctionParameter[ArrivedAperiodicJobCounter].CalculatedDeadline >
1100                AperiodicTaskCrossFunctionParameter[ArrivedAperiodicJobCounter].AbsoluteDeadline) {
1101                NumberMissAbsoluteDeadlineTask++;
1102                MissDeadlineIDStack[MissAbsoluteDeadline++] = AperiodicTaskCrossFunctionParameter[ArrivedAperiodicJobCounter].ApeirodicTaskID;
1103            }
1104            else {
1105                if (!AperiodicFlag) {
1106                    ArriveAndSetDeadline = 1;
1107                    DynamicCUSDeadline = AperiodicTaskCrossFunctionParameter[ArrivedAperiodicJobCounter].CalculatedDeadline;
1108                    ArriveAndSetTaskID = AperiodicTaskCrossFunctionParameter[ArrivedAperiodicJobCounter].ApeirodicTaskID;
1109                    ArriveAndSetCounter = ArrivedAperiodicJobCounter;
1110                    // Set new aperiodic task deadline
1111                    ptcb = OSTCBList;
1112                    while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {
1113                        if (ptcb->OSTCBId == TASK_NUMBER) {
1114                            ptcb->OSTCBArriveTime = AperiodicTaskParameter[ArrivedAperiodicJobCounter].TaskArriveTime;
1115                            ptcb->OSTCBExecutionTime = AperiodicTaskParameter[ArrivedAperiodicJobCounter].TaskExecutionTime;
1116                            ptcb->OSTCBAbsoluteDeadline = AperiodicTaskParameter[ArrivedAperiodicJobCounter].AbsoluteDeadline;
1117                            ptcb->OSTCBCalculatedDeadline = AperiodicTaskParameter[ArrivedAperiodicJobCounter].CalculatedDeadline;
1118                            ptcb->OSTCBPeriodic = AperiodicTaskCrossFunctionParameter[ArrivedAperiodicJobCounter].TaskPeriodic;
1119                            ptcb->OSTCBApeirodicTaskID = AperiodicTaskCrossFunctionParameter[ArrivedAperiodicJobCounter].ApeirodicTaskID;
1120                            ptcb->OSTCBCPUTime = 0;
1121                            OSTaskResume(SuspendPriority);
1122                            UpdateEDFPriorities();
1123                        }
1124                        ptcb = ptcb->OSTCBNext;
1125                    }
1126                    AperiodicFlag++;
1127                }
1128                else {
1129                    DoNothingIDStack[DoNothing++] = AperiodicTaskCrossFunctionParameter[ArrivedAperiodicJobCounter].ApeirodicTaskID;
1130                    StealHaveAperiodicJob++;
1131                }
1132            }
1133            ArrivedAperiodicJobCounter++;
1134        }
1135    }
```

First, in line 1150, a specific ID is added for the aperiodic task so we can identify which task has completed or been preempted. When the time tick is 0, from line 1152 to 1153, the OS will execute the function mentioned above.

```
1150    int next_apeioridc_task_number = OSTCBCur->OSTCBApeirodicTaskID;
1151    // print inital aperiodic task information
1152    if (OSTimeGet() == 0) {
1153        taskEnterCheckMissabsAndArrSetDeadline();
1154    }
```

Look at line 1157 first, it is new added for consider the task is aperiodic. To ensure we can print the desired information, a condition was added between lines 1159 to 1171, line 1184 to 1192, line 1205 to 1218 and line 1241 to 1250, to check whether an aperiodic task has finished or is perrmption.

```
1157            && !(next_task_id == TASK_NUMBER && CurrentTaskID == TASK_NUMBER)) {
1158            if (next_task_id == 65535) {
1159                if (AperiodicCalculatedDeadline > 0) {
1160                    printf("%2d\tAperiodic job(%d) is finished.\n",
1161                        OSTimeGet(), AperiodicNumber);
1162                    printf("%2d\tCompletion\t  task(%2d)(%2d)\t  task(63)\t\t %d\t\t %d\t\tN/A\n",
1163                        OSTimeGet(), CurrentTaskID, AperiodicNumber, CompletionResponseTime, CompletionPreemptionTime);
1164                    if (Output_err = fopen_s(&Output_fp, OUTPUT_FILE_NAME, "a") == 0) {
1165                        fprintf(Output_fp, "%2d\tAperiodic job(%d) is finished.\n",
1166                            OSTimeGet(), AperiodicNumber);
1167                        fprintf(Output_fp, "%2d\tCompletion\t  task(%2d)(%2d)\t  task(63)\t\t %d\t\t %d\t\tN/A\n",
1168                            OSTimeGet(), CurrentTaskID, AperiodicNumber, CompletionResponseTime, CompletionPreemptionTime);
1169                        fclose(Output_fp);
1170                    }
1171                    AperiodicFlag--;
1172                }
```

```
1184                if (next_task_id != TASK_NUMBER) {
1185                    printf("%2d\tPreemption\t  task(63)\t  task(%2d)(%2d)\n",
1186                        OSTimeGet(), next_task_id, next_task_number);
1187                    if (Output_err = fopen_s(&Output_fp, OUTPUT_FILE_NAME, "a") == 0) {
1188                        fprintf(Output_fp, "%2d\tPreemption\t  task(63)\t  task(%2d)(%2d)\n",
1189                            OSTimeGet(), next_task_id, next_task_number);
1190                        fclose(Output_fp);
1191                    }
1192                }
```

```
1205            if (AperiodicCalculatedDeadline > 0) {
1206                printf("%2d\tAperiodic job(%d) is finished.\n",
1207                    OSTimeGet(), AperiodicNumber);
1208                printf("%2d\tCompletion\t  task(%2d)(%2d)\t  task(%2d)(%2d)\t\t %d\t\t %d\t\tN/A\n",
1209                    OSTimeGet(), CurrentTaskID, AperiodicNumber, next_task_id, next_task_number, CompletionResponseTime, CompletionPreemptionTime);
1210                if (Output_err = fopen_s(&Output_fp, OUTPUT_FILE_NAME, "a") == 0) {
1211                    fprintf(Output_fp, "%2d\tAperiodic job(%d) is finished.\n",
1212                        OSTimeGet(), AperiodicNumber);
1213                    fprintf(Output_fp, "%2d\tCompletion\t  task(%2d)(%2d)\t  task(%2d)(%2d)\t\t %d\t\t %d\t\tN/A\n",
1214                        OSTimeGet(), CurrentTaskID, AperiodicNumber, next_task_id, next_task_number, CompletionResponseTime, CompletionPreemptionTime);
1215                    fclose(Output_fp);
1216                }
1217                AperiodicFlag--;
1218            }
```

```
1241                if (CurrentTaskID != TASK_NUMBER) {
1242                    printf("%2d\tPreemption\t  task(%2d)(%2d)\t  task(%2d)(%2d)\n",
1243                        OSTimeGet(), CurrentTaskID, CurrentTaskNumber, next_task_id, next_task_number);
1244                    if (Output_err = fopen_s(&Output_fp, OUTPUT_FILE_NAME, "a") == 0) {
1245                        fprintf(Output_fp, "%2d\tPreemption\t  task(%2d)(%2d)\t  task(%2d)(%2d)\n",
1246                            OSTimeGet(), CurrentTaskID, CurrentTaskNumber, next_task_id, next_task_number);
1247                        fclose(Output_fp);
1248                    }
1249
1250                }
```

Then, from line 1264 to 1303 is responsible for handling the CUS task's deadline setting. This section determines when the CUS deadline should be updated and prints the corresponding information according to the PA2 specifications.

```
1264    while (MissAbsoluteDeadline) {
1265        printf("%2d\tAperiodic job(%d) arrives. But can't scheduled. (miss absolute deadline)\n",
1266            OSTimeGet(), MissDeadlineIDStack[NumberMissAbsoluteDeadlineTask - MissAbsoluteDeadline]);
1267        if (Output_err = fopen_s(&Output_fp, OUTPUT_FILE_NAME, "a") == 0) {
1268            fprintf(Output_fp, "%2d\tAperiodic job(%d) arrives. But can't scheduled. (miss absolute deadline)\n",
1269                OSTimeGet(), MissDeadlineIDStack[NumberMissAbsoluteDeadlineTask - MissAbsoluteDeadline]);
1270            fclose(Output_fp);
1271        }
1272        MissAbsoluteDeadline--;
1273    }
1274    if (ArriveAndSetDeadline) {
1275        printf("%2d\tAperiodic job(%d) arrives and sets CUS server's deadline as %2d.\n",
1276            OSTimeGet(), ArriveAndSetTaskID, AperiodicTaskCrossFunctionParameter[ArriveAndSetCounter].CalculatedDeadline);
1277        if (Output_err = fopen_s(&Output_fp, OUTPUT_FILE_NAME, "a") == 0) {
1278            fprintf(Output_fp, "%2d\tAperiodic job(%d) arrives and sets CUS server's deadline as %2d.\n",
1279                OSTimeGet(), ArriveAndSetTaskID, AperiodicTaskCrossFunctionParameter[ArriveAndSetCounter].CalculatedDeadline);
1280            fclose(Output_fp);
1281        }
1282        ArriveAndSetDeadline = !ArriveAndSetDeadline;
1283    }
1284    if (OnlySetDeadline) {
1285        printf("%2d\tAperiodic job(%d) sets CUS server's deadline as %d.\n",
1286            OSTimeGet(), NumberCompleteAperiodicTask,
1287            AperiodicTaskCrossFunctionParameter[NumberCompleteAperiodicTask].CalculatedDeadline);
1288        if (Output_err = fopen_s(&Output_fp, OUTPUT_FILE_NAME, "a") == 0) {
1289            fprintf(Output_fp, "%2d\tAperiodic job(%d) sets CUS server's deadline as %d.\n",
1290                OSTimeGet(), NumberCompleteAperiodicTask,
1291                AperiodicTaskCrossFunctionParameter[NumberCompleteAperiodicTask].CalculatedDeadline);
1292            fclose(Output_fp);
1293        }
1294        OnlySetDeadline = !OnlySetDeadline;
1295    }
1296    while (DoNothing) {
1297        printf("%2d\tAperiodic job(%d) arrives. Do nothing.\n", OSTimeGet(), DoNothingIDStack[StealHaveAperiodicJob - DoNothing]);
1298        if (Output_err = fopen_s(&Output_fp, OUTPUT_FILE_NAME, "a") == 0) {
1299            fprintf(Output_fp, "%2d\tAperiodic job(%d) arrives. Do nothing.\n", OSTimeGet(), DoNothingIDStack[StealHaveAperiodicJob - DoNothing]);
1300            fclose(Output_fp);
1301        }
1302        DoNothing--;
1303    }
```

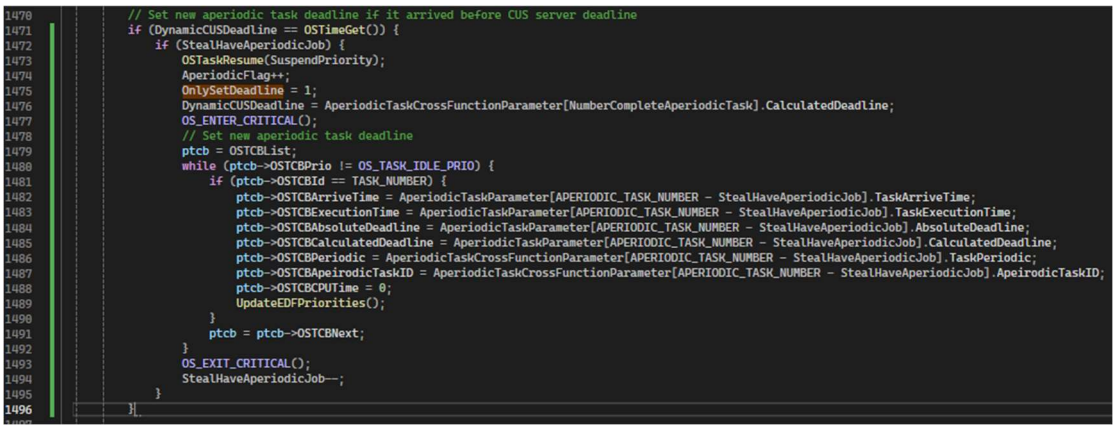
Additional time checks were added to handle the CUS task's behavior, especially in lines 1378 to 1390, which were specifically introduced for this purpose.

```
1378        if (RemainTime == 0 && DynamicCUSDeadline > 0 && OSTCBCur->OSTCBCalculatedDeadline > 0) { // If is budget task need compute another
1379            CurrentTaskNumber = AperiodicNumber;
1380            CompletionResponseTime = OSTimeGet() - OSTCBCur->OSTCBArriveTime;
1381            if (OSTCBCur->OSTCBArriveTime <= CompleteApriodicFinishedTime) {
1382                CompletionPreemptionTime = OSTimeGet() - CompleteApriodicCalculatedDeadline - OSTCBCur->OSTCBExecutionTime;
1383            }
1384            else {
1385                CompletionPreemptionTime = OSTimeGet() - OSTCBCur->OSTCBArriveTime - OSTCBCur->OSTCBExecutionTime;
1386            }
1387            CompletionTimeDly = OSTCBCur->OSTCBPeriodic - OSTimeGet() + OSTCBCur->OSTCBArriveTime;
1388        }
1389        AperiodicAbsolabeDeadline = OSTCBCur->OSTCBAbsoluteDeadline;
1390        AperiodicCalculatedDeadline = OSTCBCur->OSTCBCalculatedDeadline;
```

From line 1430 to line 1468, I set a new aperiodic task deadline if no aperiodic task is currently active by checking the arrival time. I also check whether the current aperiodic task has missed its absolute deadline and assign the next aperiodic task's information to the budget accordingly.

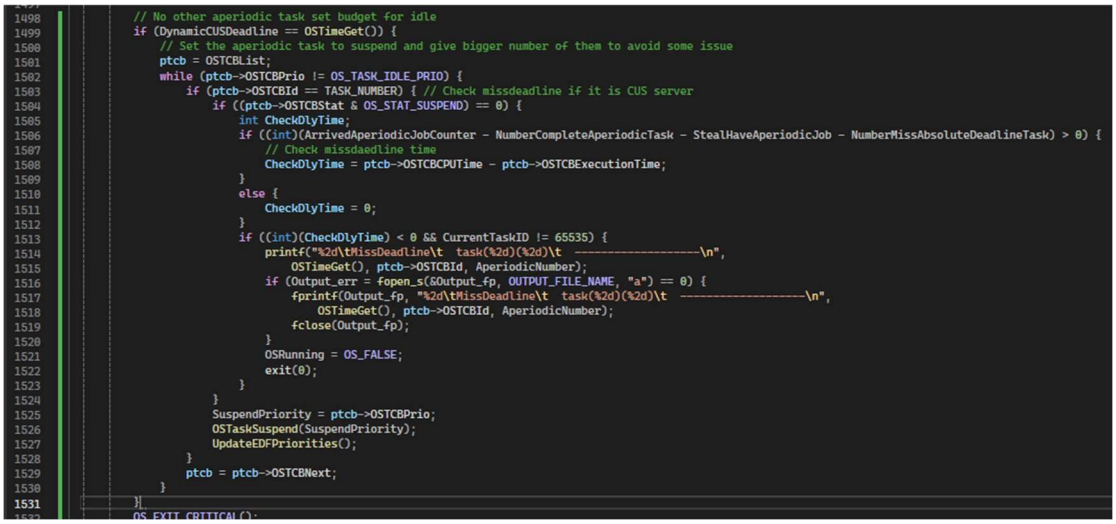
```
1430        // Set new aperiodic task deadline if there is no aperiodic task by checking arrive time
1431        while (AperiodicTaskCrossFunctionParameter[ArrivedAperiodicJobCounter].TaskArriveTime == OSTimeGet()) {
1432            // Check if the aperiodic task is missed absolute deadline
1433            if (AperiodicTaskCrossFunctionParameter[ArrivedAperiodicJobCounter].CalculatedDeadline >
1434                AperiodicTaskCrossFunctionParameter[ArrivedAperiodicJobCounter].AbsoluteDeadline) {
1435                NumberMissAbsoluteDeadlineTask++;
1436                MissDeadlineIDStack[MissAbsoluteDeadline++] = AperiodicTaskCrossFunctionParameter[ArrivedAperiodicJobCounter].ApeirodicTaskID;
1437            }
1438            else {
1439                if (!AperiodicFlag) {
1440                    ArriveAndSetDeadline = 1;
1441                    DynamicCUSDeadline = AperiodicTaskCrossFunctionParameter[ArrivedAperiodicJobCounter].CalculatedDeadline;
1442                    ArriveAndSetTaskID = AperiodicTaskCrossFunctionParameter[ArrivedAperiodicJobCounter].ApeirodicTaskID;
1443                    ArriveAndSetCounter = ArrivedAperiodicJobCounter;
1444                    // Set new aperiodic task deadline
1445                    ptcb = OSTCBList;
1446                    while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {
1447                        if (ptcb->OSTCBId == TASK_NUMBER) {
1448                            ptcb->OSTCBArriveTime = AperiodicTaskParameter[ArrivedAperiodicJobCounter].TaskArriveTime;
1449                            ptcb->OSTCBExecutionTime = AperiodicTaskParameter[ArrivedAperiodicJobCounter].TaskExecutionTime;
1450                            ptcb->OSTCBAbsoluteDeadline = AperiodicTaskParameter[ArrivedAperiodicJobCounter].AbsoluteDeadline;
1451                            ptcb->OSTCBCalculatedDeadline = AperiodicTaskParameter[ArrivedAperiodicJobCounter].CalculatedDeadline;
1452                            ptcb->OSTCBPeriodic = AperiodicTaskCrossFunctionParameter[ArrivedAperiodicJobCounter].TaskPeriodic;
1453                            ptcb->OSTCBApeirodicTaskID = AperiodicTaskCrossFunctionParameter[ArrivedAperiodicJobCounter].ApeirodicTaskID;
1454                            ptcb->OSTCBCPUTime = 0;
1455                            OSTaskResume(SuspendPriority);
1456                            UpdateEDFPriorities();
1457                        }
1458                        ptcb = ptcb->OSTCBNext;
1459                    }
1460                    AperiodicFlag++;
1461                }
1462                else {
1463                    DoNothingIDStack[DoNothing++] = AperiodicTaskCrossFunctionParameter[ArrivedAperiodicJobCounter].ApeirodicTaskID;
1464                    StealHaveAperiodicJob++;
1465                }
1466            }
1467            ArrivedAperiodicJobCounter++;
1468        }
```

In the previous section, we set the deadline when a new aperiodic task arrives. In contrast, lines 1470 to 1496 only update the CUS deadline, since the task has already arrived and is simply waiting for execution.



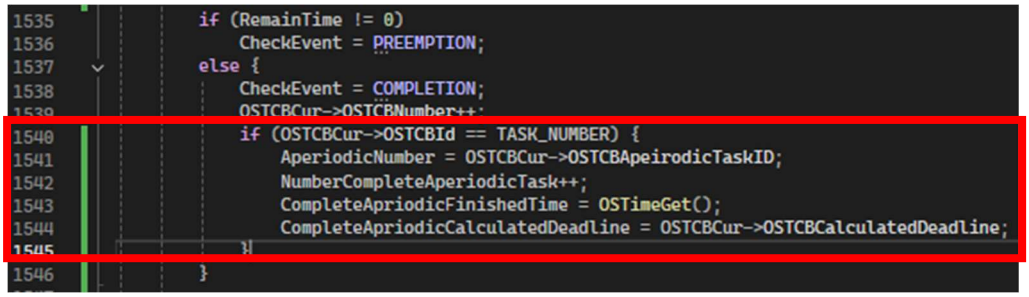
```
1470        // Set new aperiodic task deadline if it arrived before CUS server deadline
1471        if (DynamicCUSDeadline == OSTimeGet()) {
1472            if (StealHaveAperiodicJob) {
1473                OSTaskResume(SuspendPriority);
1474                AperiodicFlag++;
1475                OnlySetDeadline = 1;
1476                DynamicCUSDeadline = AperiodicTaskCrossFunctionParameter[NumberCompleteAperiodicTask].CalculatedDeadline;
1477                OS_ENTER_CRITICAL();
1478                // Set new aperiodic task deadline
1479                ptcb = OSTCBList;
1480                while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {
1481                    if (ptcb->OSTCBId == TASK_NUMBER) {
1482                        ptcb->OSTCBArriveTime = AperiodicTaskParameter[APERIODIC_TASK_NUMBER - StealHaveAperiodicJob].TaskArriveTime;
1483                        ptcb->OSTCBExecutionTime = AperiodicTaskParameter[APERIODIC_TASK_NUMBER - StealHaveAperiodicJob].TaskExecutionTime;
1484                        ptcb->OSTCBAbsoluteDeadline = AperiodicTaskParameter[APERIODIC_TASK_NUMBER - StealHaveAperiodicJob].AbsoluteDeadline;
1485                        ptcb->OSTCBCalculatedDeadline = AperiodicTaskParameter[APERIODIC_TASK_NUMBER - StealHaveAperiodicJob].CalculatedDeadline;
1486                        ptcb->OSTCBPeriodic = AperiodicTaskCrossFunctionParameter[APERIODIC_TASK_NUMBER - StealHaveAperiodicJob].TaskPeriodic;
1487                        ptcb->OSTCBApeirodicTaskID = AperiodicTaskCrossFunctionParameter[APERIODIC_TASK_NUMBER - StealHaveAperiodicJob].ApeirodicTaskID;
1488                        ptcb->OSTCBCPUTime = 0;
1489                        UpdateEDFPriorities();
1490                    }
1491                    ptcb = ptcb->OSTCBNext;
1492                }
1493                OS_EXIT_CRITICAL();
1494                StealHaveAperiodicJob--;
1495            }
1496        }
```

Finally, to detect whether all aperiodic tasks are completed, lines 1498 to 1531 perform this check. Additionally, this section also detects if any aperiodic task has missed its deadline. At the end of each aperiodic task, we also suspend the aperiodic server budget to prevent further scheduling.



```
1498        // No other aperiodic task set budget for idle
1499        if (DynamicCUSDeadline == OSTimeGet()) {
1500            // Set the aperiodic task to suspend and give bigger number of them to avoid some issue
1501            ptcb = OSTCBList;
1502            while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {
1503                if (ptcb->OSTCBId == TASK_NUMBER) { // Check missdeadline if it is CUS server
1504                    if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) == 0) {
1505                        int CheckDlyTime;
1506                        if ((int)(ArrivedAperiodicJobCounter - NumberCompleteAperiodicTask - StealHaveAperiodicJob - NumberMissAbsoluteDeadlineTask) > 0) {
1507                            // Check missdaedline time
1508                            CheckDlyTime = ptcb->OSTCBCPUTime - ptcb->OSTCBExecutionTime;
1509                        }
1510                        else {
1511                            CheckDlyTime = 0;
1512                        }
1513                        if ((int)(CheckDlyTime) < 0 && CurrentTaskID != 65535) {
1514                            printf("%2d\tMissDeadline\t  task(%2d)(%2d)\t  --------------------\n",
1515                                OSTimeGet(), ptcb->OSTCBId, AperiodicNumber);
1516                            if (Output_err = fopen_s(&Output_fp, OUTPUT_FILE_NAME, "a") == 0) {
1517                                fprintf(Output_fp, "%2d\tMissDeadline\t  task(%2d)(%2d)\t  --------------------\n",
1518                                    OSTimeGet(), ptcb->OSTCBId, AperiodicNumber);
1519                                fclose(Output_fp);
1520                            }
1521                            OSRunning = OS_FALSE;
1522                            exit(0);
1523                        }
1524                    }
1525                    SuspendPriority = ptcb->OSTCBPrio;
1526                    OSTaskSuspend(SuspendPriority);
1527                    UpdateEDFPriorities();
1528                }
1529                ptcb = ptcb->OSTCBNext;
1530            }
1531        }
```

Finally, in lines 1540 to 1545, we record additional information about the aperiodic task's budget. This is used to track how many aperiodic tasks have been completed so far, as well as to store the completion time of the previous aperiodic task. This information helps provide a reference for comparison when handling the next aperiodic task.



```
1535            if (RemainTime != 0)
1536                CheckEvent = PREEMPTION;
1537            else {
1538                CheckEvent = COMPLETION;
1539                OSTCBCur->OSTCBNumber++;
1540                if (OSTCBCur->OSTCBId == TASK_NUMBER) {
1541                    AperiodicNumber = OSTCBCur->OSTCBApeirodicTaskID;
1542                    NumberCompleteAperiodicTask++;
1543                    CompleteApriodicFinishedTime = OSTimeGet();
1544                    CompleteApriodicCalculatedDeadline = OSTCBCur->OSTCBCalculatedDeadline;
1545                }
1546            }
```

Going back a bit, the parameters defined from lines 969 to 989 are used in the above-mentioned logic for the CUS calculations.

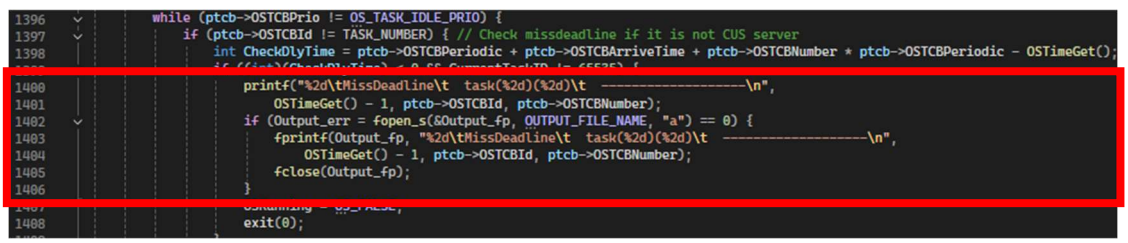
```
969   INT16U CurrentAperiodicTaskID;
970   INT32U AperiodicAbsolabeDeadline;
971   INT32U AperiodicCalculatedDeadline;
972   INT32U DynamicCUSDeadline = 0;
973   INT32U SuspendPriority = 0;
974   INT32U CompleteApriodicFinishedTime = 0;
975   INT32U CompleteApriodicCalculatedDeadline = 0;
976   INT8U ArrivedAperiodicJobCounter = 0;
977   INT8U StealHaveAperiodicJob = 0;
978   INT8U AperiodicNumber = 0;
979   INT8U ArriveAndSetCounter = 0;
980   INT8U ArriveAndSetTaskID;
981   INT8U NumberMissAbsoluteDeadlineTask = 0;
982   INT8U NumberCompleteAperiodicTask = 0;
983   INT8U AperiodicFlag = 0;
984   INT8U DoNothing = 0;
985   BOOLEAN MissAbsoluteDeadline = 0;
986   BOOLEAN ArriveAndSetDeadline = 0;
987   BOOLEAN OnlySetDeadline = 0;
988   INT8U DoNothingIDStack[OS_MAX_TASKS];
989   INT8U MissDeadlineIDStack[OS_MAX_TASKS];
```

Implement and describe how to handle the deadline missing situation under CUS.

As mentioned earlier, the EDF algorithm checks for missed deadlines between lines 1400 and 1406. In the context of CUS, the missed deadline situation is handled similarly but with some additional checks to accommodate the task execution under CUS constraints.

In the CUS case, missed deadlines are checked in the block between lines 1514 and 1520. The basic idea remains the same: we need to check whether the execution time of the task has exceeded its required execution time by the deadline. However, a special consideration is made here because, during task completion, the execution time of the task is recorded as zero. This could potentially lead to false "missed deadline" detections after an aperiodic task completes.

To prevent this, we introduce an additional check to ensure that the task's completion status is accurately tracked. Specifically, at line 1506, I check whether the aperiodic task is actually completed, so that we don't mistakenly flag a task as missing its deadline after completion.



```
1502            while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {
1503                if (ptcb->OSTCBId == TASK_NUMBER) { // Check missdeadline if it is CUS server
1504                    if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) == 0) {
1505                        int CheckDlyTime;
1506                        if ((int)(ArrivedAperiodicJobCounter - NumberCompleteAperiodicTask - StealHaveAperiodicJob - NumberMissAbsoluteDeadlineTask) > 0) {
1507                            // Check missdaedline time
1508                            CheckDlyTime = ptcb->OSTCBCPUTime - ptcb->OSTCBExecutionTime;
1509                        }
1510                        else {
1511                            CheckDlyTime = 0;
1512                        }
1513                        if ((int)(CheckDlyTime) < 0 && CurrentTaskID != 65535) {
1514                            printf("%2d\tMissDeadline\t  task(%2d)(%2d)\t  ------------------\n",
1515                                OSTimeGet(), ptcb->OSTCBId, AperiodicNumber);
1516                            if (Output_err = fopen_s(&Output_fp, OUTPUT_FILE_NAME, "a") == 0) {
1517                                fprintf(Output_fp, "%2d\tMissDeadline\t  task(%2d)(%2d)\t  ------------------\n",
1518                                    OSTimeGet(), ptcb->OSTCBId, AperiodicNumber);
1519                                fclose(Output_fp);
1520                            }
1521                            OSRunning = OS_FALSE;
1522                            exit(0);
1523                        }
1524                    }
1525                    SuspendPriority = ptcb->OSTCBPrio;
```