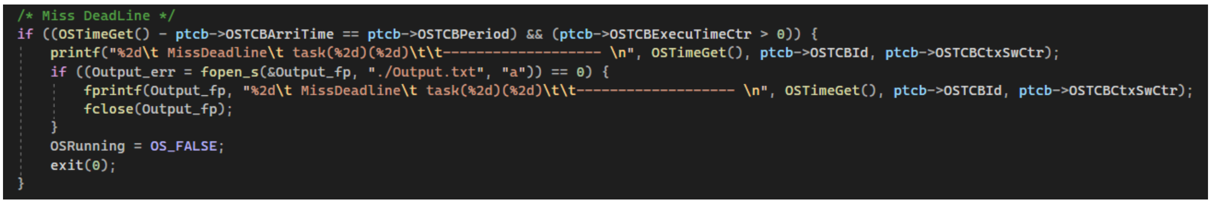
PA2

M11202117 呂長恩

Part 1 EDF

Miss Deadline Handler

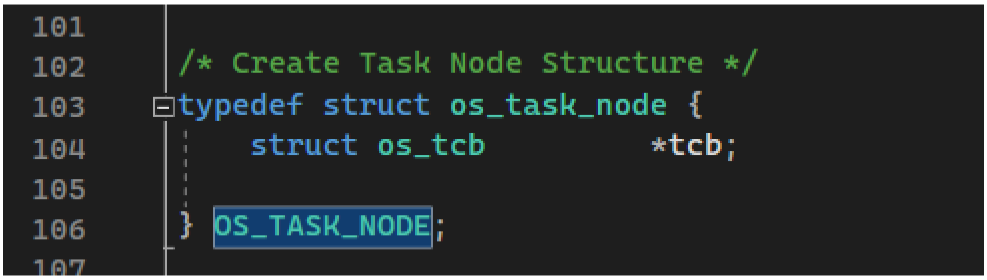


```
/* Miss DeadLine */
if ((OSTimeGet() - ptcb->OSTCBArriTime == ptcb->OSTCBPeriod) && (ptcb->OSTCBExecuTimeCtr > 0)) {
    printf("%2d\t MissDeadline\t task(%2d)(%2d)\t\t------------------- \n", OSTimeGet(), ptcb->OSTCBId, ptcb->OSTCBCtxSwCtr);
    if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
        fprintf(Output_fp, "%2d\t MissDeadline\t task(%2d)(%2d)\t\t------------------- \n", OSTimeGet(), ptcb->OSTCBId, ptcb->OSTCBCtxSwCtr);
        fclose(Output_fp);
    }
    OSRunning = OS_FALSE;
    exit(0);
}
```

- When the current time minus the arrival time equals to its period, and the task has not been completed(”OSTCBExecuTimeCtr” is greater than 0), a “miss deadline” occurs.
- Therefore, I designed a checking mechanism within the `OSTimeTick()`. Whenever each `OSTimeTick()` is triggered, it searches through **all task control blocks (TCBs)** to inspect and confirm whether any task has missed its deadline. If any task misses deadline, system will print out “miss deadline information” and shuts down the entire system.
- This checking mechanism is executed after the current TCB’s “OSTCBExecuTimeCtr” minus 1. Therefore, the occurrence of mistakenly assuming a 'miss deadline' won't happen after the current tick is executed.

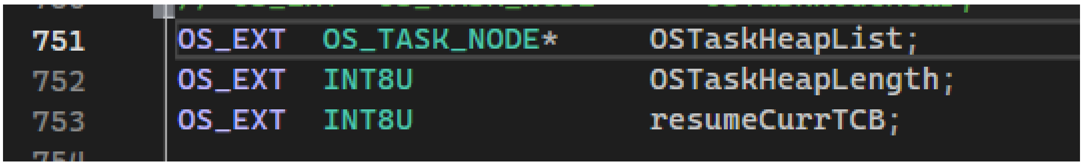
Implementation

> EDF is mainly implemented by Heap structure. Each arrived TCB is stored in **Minimum Heap**, and the task with earliest deadline will always be stored in Heap[1] by `MinHeapify()`. For example, When the current task is finished, its deadline will be updated and execute `MinHeapify()` to find the task with earliest deadline.



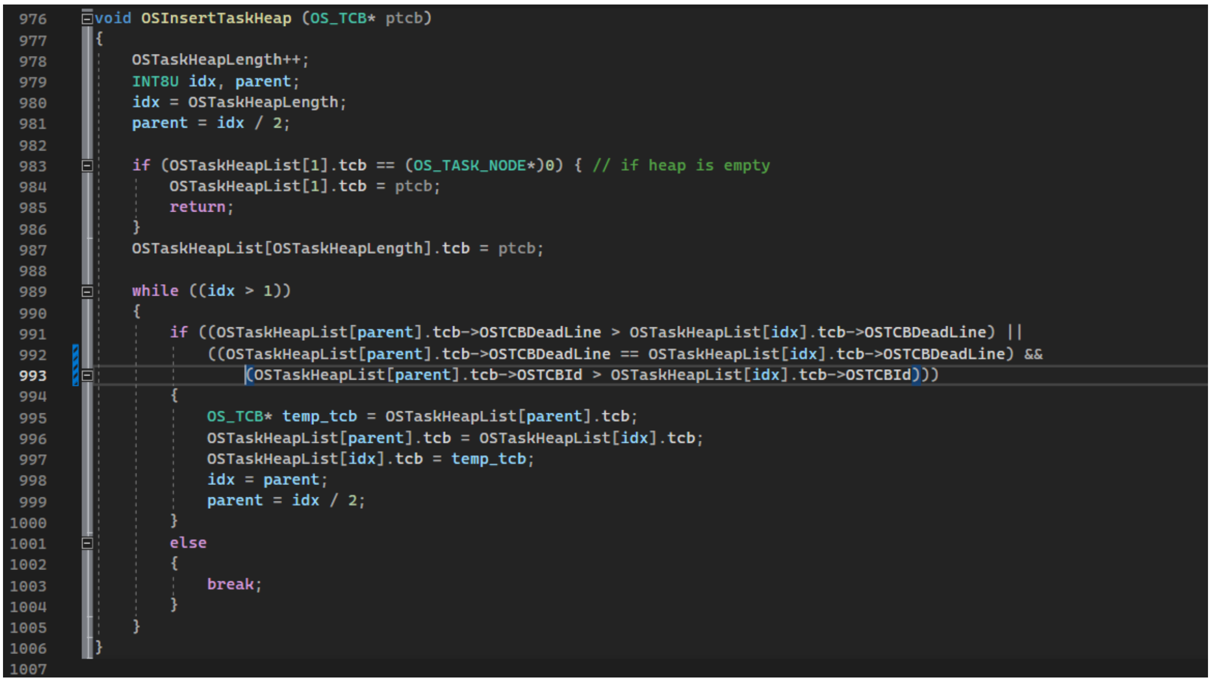
```c
/* Create Task Node Structure */
typedef struct os_task_node {
    struct os_tcb           *tcb;

} OS_TASK_NODE;
```

- file: `ucos_ii.h`
- Declare a new structure, OS_TASK_NODE, which stores TCB’s address.



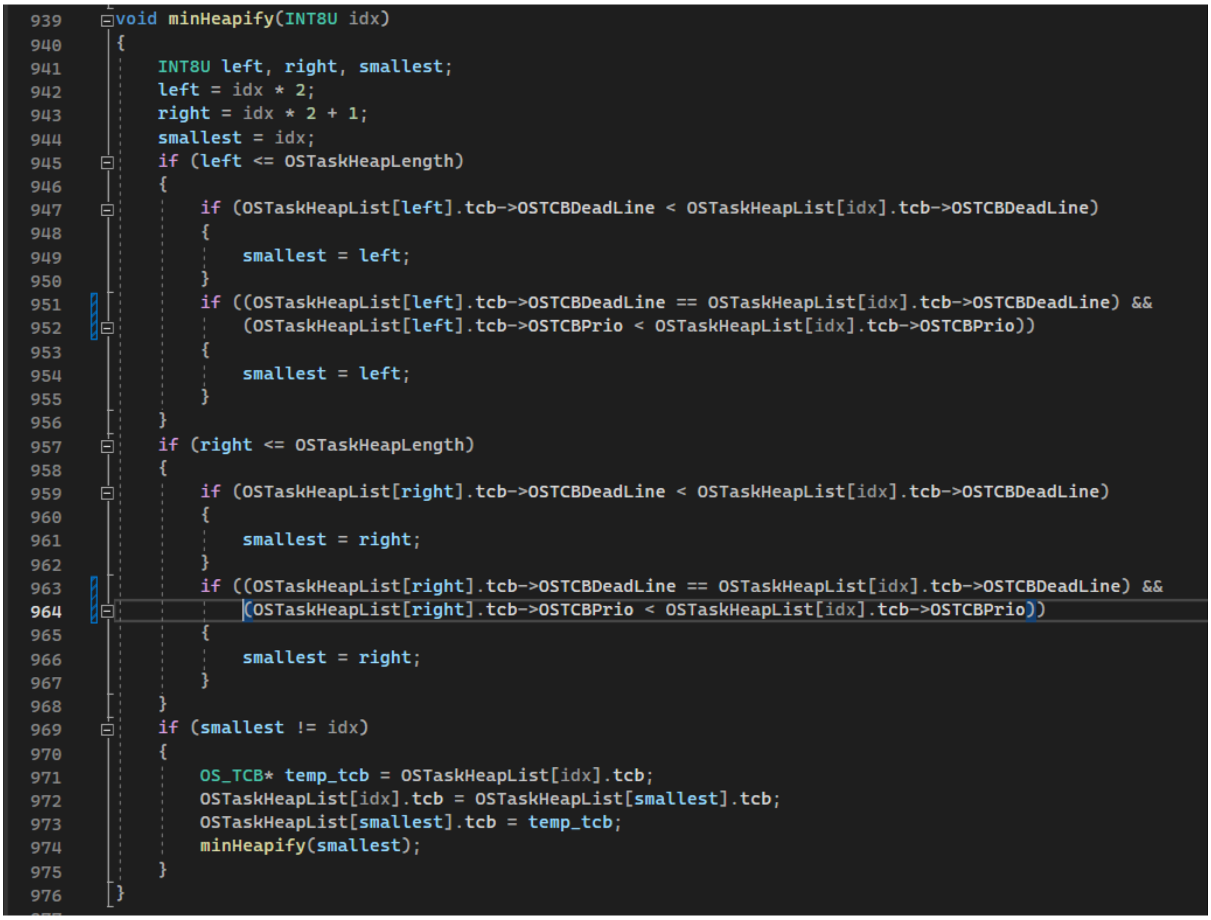
```c
OS_EXT  OS_TASK_NODE*     OSTaskHeapList;
OS_EXT  INT8U             OSTaskHeapLength;
OS_EXT  INT8U             resumeCurrTCB;
```

- file: `ucos_ii.h`
- Declare three new global variables
    - OSTaskHeapList: A pointer points the minimum heap array.
    - OSTaskHeapLength: The number of the tasks in the heap array.
    - resumeCurrTCB: If a task's completion time is equal to its deadline, set a flag to ensure that the system will only reset the deadline, arrival time, etc., after displaying preemptive time information.



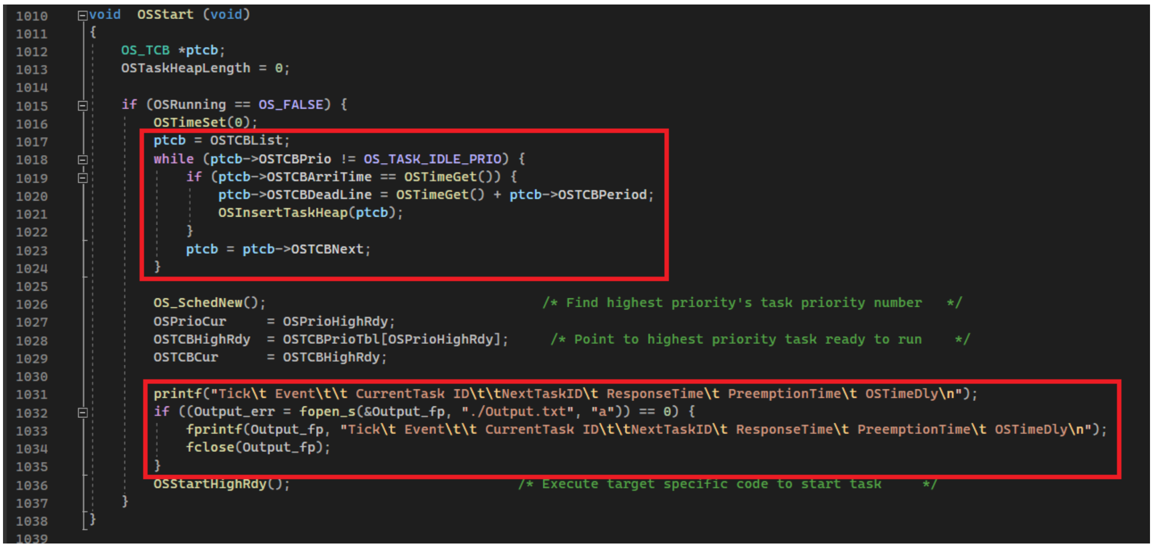
```c
void OSInsertTaskHeap (OS_TCB* ptcb)
{
    OSTaskHeapLength++;
    INT8U idx, parent;
    idx = OSTaskHeapLength;
    parent = idx / 2;

    if (OSTaskHeapList[1].tcb == (OS_TASK_NODE*)0) { // if heap is empty
        OSTaskHeapList[1].tcb = ptcb;
        return;
    }
    OSTaskHeapList[OSTaskHeapLength].tcb = ptcb;

    while ((idx > 1))
    {
        if ((OSTaskHeapList[parent].tcb->OSTCBDeadLine > OSTaskHeapList[idx].tcb->OSTCBDeadLine) ||
            ((OSTaskHeapList[parent].tcb->OSTCBDeadLine == OSTaskHeapList[idx].tcb->OSTCBDeadLine) &&
                (OSTaskHeapList[parent].tcb->OSTCBId > OSTaskHeapList[idx].tcb->OSTCBId)))
        {
            OS_TCB* temp_tcb = OSTaskHeapList[parent].tcb;
            OSTaskHeapList[parent].tcb = OSTaskHeapList[idx].tcb;
            OSTaskHeapList[idx].tcb = temp_tcb;
            idx = parent;
            parent = idx / 2;
        }
        else
        {
            break;
        }
    }
}
```

- file: `os_core.c`
- When we are going to insert a TCB into the heap, call this function to implement.
- It will check whether the heap is empty and find the right position to insert ensures the heap can working.



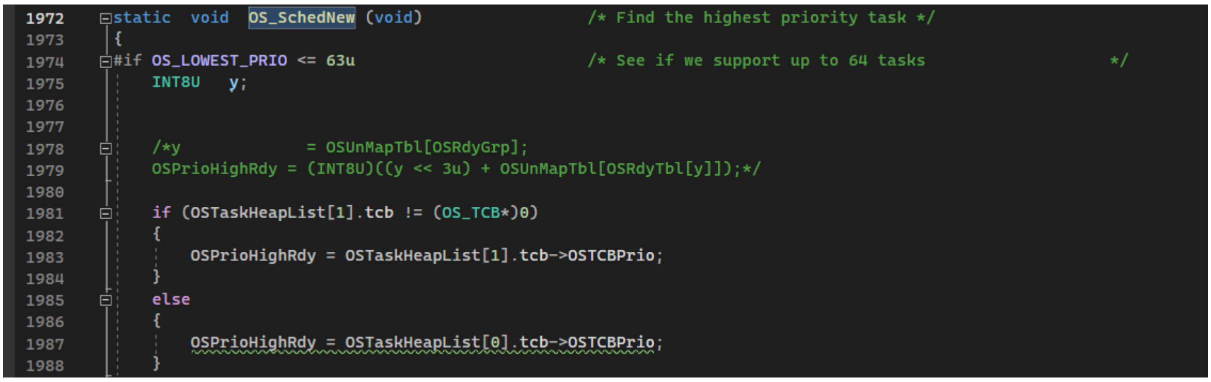
```c
void minHeapify(INT8U idx)
{
    INT8U left, right, smallest;
    left = idx * 2;
    right = idx * 2 + 1;
    smallest = idx;
    if (left <= OSTaskHeapLength)
    {
        if (OSTaskHeapList[left].tcb->OSTCBDeadLine < OSTaskHeapList[idx].tcb->OSTCBDeadLine)
        {
            smallest = left;
        }
        if ((OSTaskHeapList[left].tcb->OSTCBDeadLine == OSTaskHeapList[idx].tcb->OSTCBDeadLine) &&
            (OSTaskHeapList[left].tcb->OSTCBPrio < OSTaskHeapList[idx].tcb->OSTCBPrio))
        {
            smallest = left;
        }
    }
    if (right <= OSTaskHeapLength)
    {
        if (OSTaskHeapList[right].tcb->OSTCBDeadLine < OSTaskHeapList[idx].tcb->OSTCBDeadLine)
        {
            smallest = right;
        }
        if ((OSTaskHeapList[right].tcb->OSTCBDeadLine == OSTaskHeapList[idx].tcb->OSTCBDeadLine) &&
            (OSTaskHeapList[right].tcb->OSTCBPrio < OSTaskHeapList[idx].tcb->OSTCBPrio))
        {
            smallest = right;
        }
    }
    if (smallest != idx)
    {
        OS_TCB* temp_tcb = OSTaskHeapList[idx].tcb;
        OSTaskHeapList[idx].tcb = OSTaskHeapList[smallest].tcb;
        OSTaskHeapList[smallest].tcb = temp_tcb;
        minHeapify(smallest);
    }
}
```

- file: `os_core.c`
- This function is called when the task with earliest deadline is finished, ucos-2 has to find the next task to execute.
- `minHeapify()` can ensure the Heap[1] always stores the task with earliest deadline and maintain the heap structure.



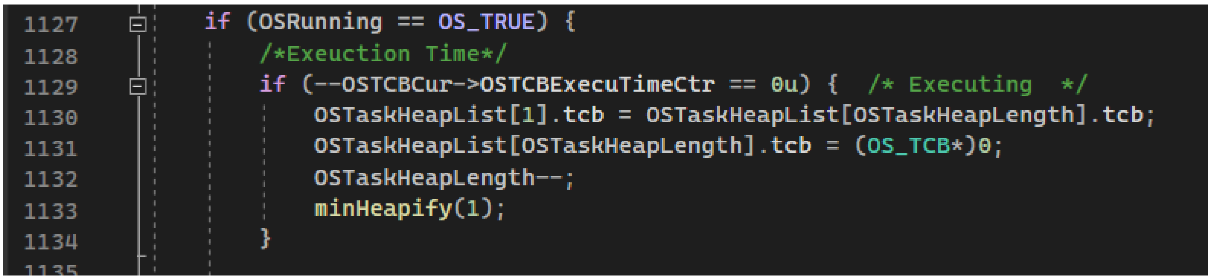
```
void  OSStart (void)
{
    OS_TCB *ptcb;
    OSTaskHeapLength = 0;

    if (OSRunning == OS_FALSE) {
        OSTimeSet(0);
        ptcb = OSTCBList;
        while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {
            if (ptcb->OSTCBArriTime == OSTimeGet()) {
                ptcb->OSTCBDeadLine = OSTimeGet() + ptcb->OSTCBPeriod;
                OSInsertTaskHeap(ptcb);
            }
            ptcb = ptcb->OSTCBNext;
        }

        OS_SchedNew();                                  /* Find highest priority's task priority number   */
        OSPrioCur     = OSPrioHighRdy;
        OSTCBHighRdy  = OSTCBPrioTbl[OSPrioHighRdy];     /* Point to highest priority task ready to run    */
        OSTCBCur      = OSTCBHighRdy;

        printf("Tick\t Event\t\t CurrentTask ID\t\tNextTaskID\t ResponseTime\t PreemptionTime\t OSTimeDly\n");
        if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
            fprintf(Output_fp, "Tick\t Event\t\t CurrentTask ID\t\tNextTaskID\t ResponseTime\t PreemptionTime\t OSTimeDly\n");
            fclose(Output_fp);
        }
        OSStartHighRdy();                            /* Execute target specific code to start task     */
    }
}
```

- file: `os_core.c`
- Check all TCBs to confirm whether any task arrives.
- Display the title.



```
static  void  OS_SchedNew (void)                 /* Find the highest priority task */
{
#if OS_LOWEST_PRIO <= 63u                        /* See if we support up to 64 tasks                  */
    INT8U   y;


    /*y             = OSUnMapTbl[OSRdyGrp];
    OSPrioHighRdy = (INT8U)((y << 3u) + OSUnMapTbl[OSRdyTbl[y]]);*/

    if (OSTaskHeapList[1].tcb != (OS_TCB*)0)
    {
        OSPrioHighRdy = OSTaskHeapList[1].tcb->OSTCBPrio;
    }
    else
    {
        OSPrioHighRdy = OSTaskHeapList[0].tcb->OSTCBPrio;
    }
```

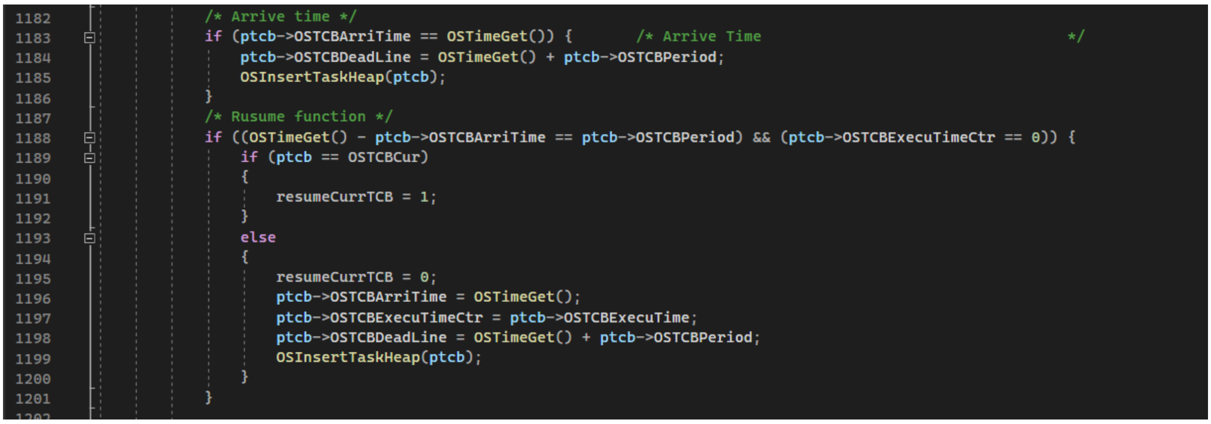
- file: `os_core.c`
- `OS_SchedNew()` generally sets OSPrioHighRdy to Heap[1]’s priority. However, if the Heap[1] doesn’t exist, then sets the OSPrioHighRdy to Heap[0]’s priority.
- The priority of Heap[0] is the priority of idle task.



```
        if (OSRunning == OS_TRUE) {
            /*Exeuction Time*/
            if (--OSTCBCur->OSTCBExecuTimeCtr == 0u) {  /* Executing  */
                OSTaskHeapList[1].tcb = OSTaskHeapList[OSTaskHeapLength].tcb;
                OSTaskHeapList[OSTaskHeapLength].tcb = (OS_TCB*)0;
                OSTaskHeapLength--;
                minHeapify(1);
            }
```

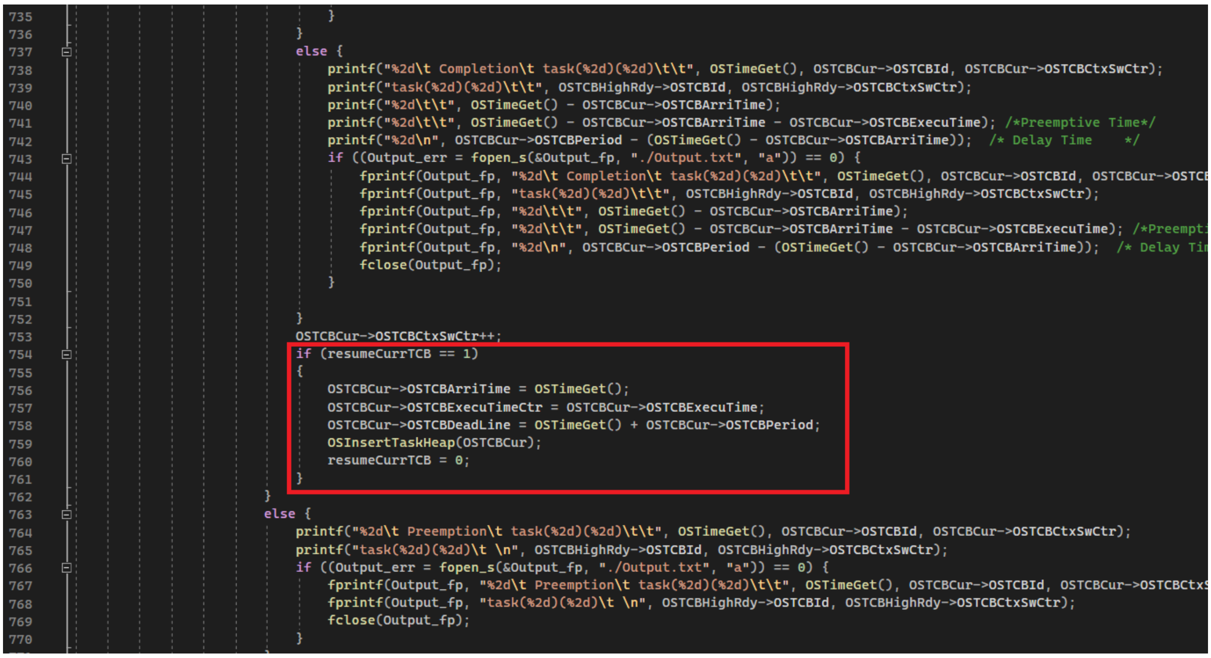
- file: `os_core.c`

- The code above is in the TimeTick() function. When TimeTick() is trigged, the current TCB will minus 1 until the remaining execution time equals to 0.
- if the remaining time equals to 0, deletes the current task in Heap, and calls minHeapify() to find the next task with the earliest deadline.



```
/* Arrive time */
if (ptcb->OSTCBArriTime == OSTimeGet()) {       /* Arrive Time                                    */
    ptcb->OSTCBDeadLine = OSTimeGet() + ptcb->OSTCBPeriod;
    OSInsertTaskHeap(ptcb);
}
/* Rusume function */
if ((OSTimeGet() - ptcb->OSTCBArriTime == ptcb->OSTCBPeriod) && (ptcb->OSTCBExecuTimeCtr == 0)) {
    if (ptcb == OSTCBCur)
    {
        resumeCurrTCB = 1;
    }
    else
    {
        resumeCurrTCB = 0;
        ptcb->OSTCBArriTime = OSTimeGet();
        ptcb->OSTCBExecuTimeCtr = ptcb->OSTCBExecuTime;
        ptcb->OSTCBDeadLine = OSTimeGet() + ptcb->OSTCBPeriod;
        OSInsertTaskHeap(ptcb);
    }
}
```

- file: `os_core.c`
- Arrive Time
  - Check all TCBs to confirm whether any task arrives.
  - If a task has just arrived, insert it into the heap.
- Resume Function
  - If any task completes a cycle and has completed within its period time, prepare to resume and insert it into the heap.
  - If the task to be resumed has just completed at the current time, set a flag and wait to change arrival time, etc., until after displaying the information.
  - If the task to be completed at a previous time, simply modify the arrival time without any impact.



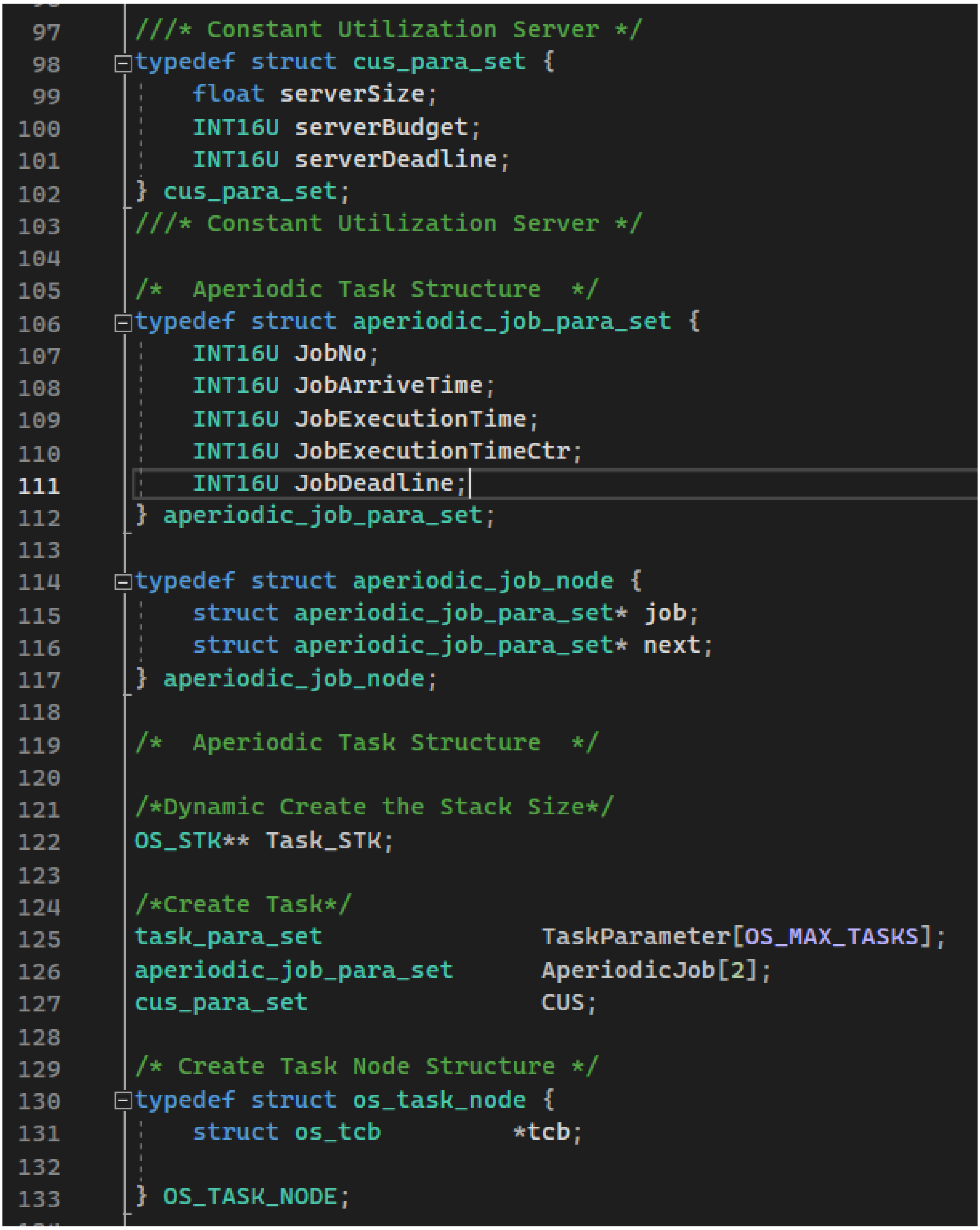
```
        }
    }
    else {
        printf("%2d\t Completion\t task(%2d)(%2d)\t\t", OSTimeGet(), OSTCBCur->OSTCBId, OSTCBCur->OSTCBCtxSwCtr);
        printf("task(%2d)(%2d)\t\t", OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBCtxSwCtr);
        printf("%2d\t\t", OSTimeGet() - OSTCBCur->OSTCBArriTime);
        printf("%2d\t\t", OSTimeGet() - OSTCBCur->OSTCBArriTime - OSTCBCur->OSTCBExecuTime); /*Preemptive Time*/
        printf("%2d\n", OSTCBCur->OSTCBPeriod - (OSTimeGet() - OSTCBCur->OSTCBArriTime));  /* Delay Time    */
        if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
            fprintf(Output_fp, "%2d\t Completion\t task(%2d)(%2d)\t\t", OSTimeGet(), OSTCBCur->OSTCBId, OSTCBCur->OSTCE
            fprintf(Output_fp, "task(%2d)(%2d)\t\t", OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBCtxSwCtr);
            fprintf(Output_fp, "%2d\t\t", OSTimeGet() - OSTCBCur->OSTCBArriTime);
            fprintf(Output_fp, "%2d\t\t", OSTimeGet() - OSTCBCur->OSTCBArriTime - OSTCBCur->OSTCBExecuTime); /*Preempti
            fprintf(Output_fp, "%2d\n", OSTCBCur->OSTCBPeriod - (OSTimeGet() - OSTCBCur->OSTCBArriTime));  /* Delay Ti
            fclose(Output_fp);
        }

    }
    OSTCBCur->OSTCBCtxSwCtr++;
    if (resumeCurrTCB == 1)
    {
        OSTCBCur->OSTCBArriTime = OSTimeGet();
        OSTCBCur->OSTCBExecuTimeCtr = OSTCBCur->OSTCBExecuTime;
        OSTCBCur->OSTCBDeadLine = OSTimeGet() + OSTCBCur->OSTCBPeriod;
        OSInsertTaskHeap(OSTCBCur);
        resumeCurrTCB = 0;
    }
}
else {
    printf("%2d\t Preemption\t task(%2d)(%2d)\t\t", OSTimeGet(), OSTCBCur->OSTCBId, OSTCBCur->OSTCBCtxSwCtr);
    printf("task(%2d)(%2d)\t \n", OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBCtxSwCtr);
    if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
        fprintf(Output_fp, "%2d\t Preemption\t task(%2d)(%2d)\t\t", OSTimeGet(), OSTCBCur->OSTCBId, OSTCBCur->OSTCBCtxS
        fprintf(Output_fp, "task(%2d)(%2d)\t \n", OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBCtxSwCtr);
        fclose(Output_fp);
    }
```

- file: `os_core.c` ->OSIntExit (void)
- After displaying the information, the Arrive time is updated here.

Part 2 CUS

Implementation

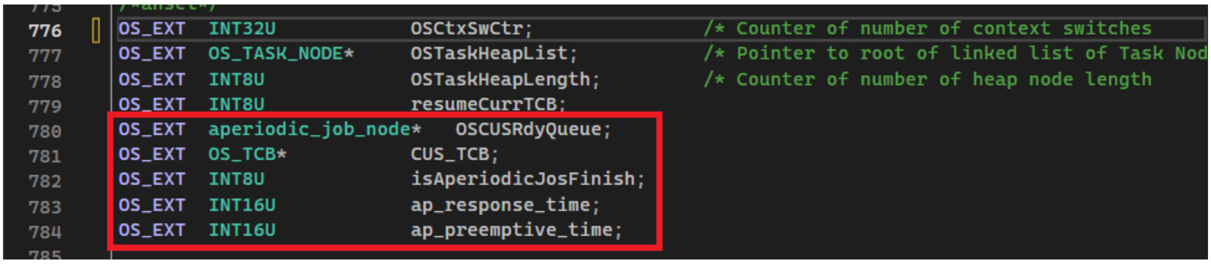
> I added a CUS structure to store serverSize, serverBudget, and serverDeadline to ensure the correctness of CUS information. Because the Server’s task may change its deadline based on whether the current server queue has an aperiodic job, to make EDF select it for execution or not, a new CUS structure is needed to store the actual variables of CUS.
> If there are no aperiodic jobs currently available for execution, set the deadline of the server's task to 99, preventing EDF from selecting it for execution. If there are aperiodic jobs available for execution, set the deadline of the server's task to the CUS’s serverDeadline, allowing EDF to select it for execution later.



```c
///* Constant Utilization Server */
typedef struct cus_para_set {
    float serverSize;
    INT16U serverBudget;
    INT16U serverDeadline;
} cus_para_set;
///* Constant Utilization Server */

/*  Aperiodic Task Structure  */
typedef struct aperiodic_job_para_set {
    INT16U JobNo;
    INT16U JobArriveTime;
    INT16U JobExecutionTime;
    INT16U JobExecutionTimeCtr;
    INT16U JobDeadline;
} aperiodic_job_para_set;

typedef struct aperiodic_job_node {
    struct aperiodic_job_para_set* job;
    struct aperiodic_job_para_set* next;
} aperiodic_job_node;

/*  Aperiodic Task Structure  */

/*Dynamic Create the Stack Size*/
OS_STK** Task_STK;

/*Create Task*/
task_para_set               TaskParameter[OS_MAX_TASKS];
aperiodic_job_para_set      AperiodicJob[2];
cus_para_set                CUS;

/* Create Task Node Structure */
typedef struct os_task_node {
    struct os_tcb         *tcb;

} OS_TASK_NODE;
```

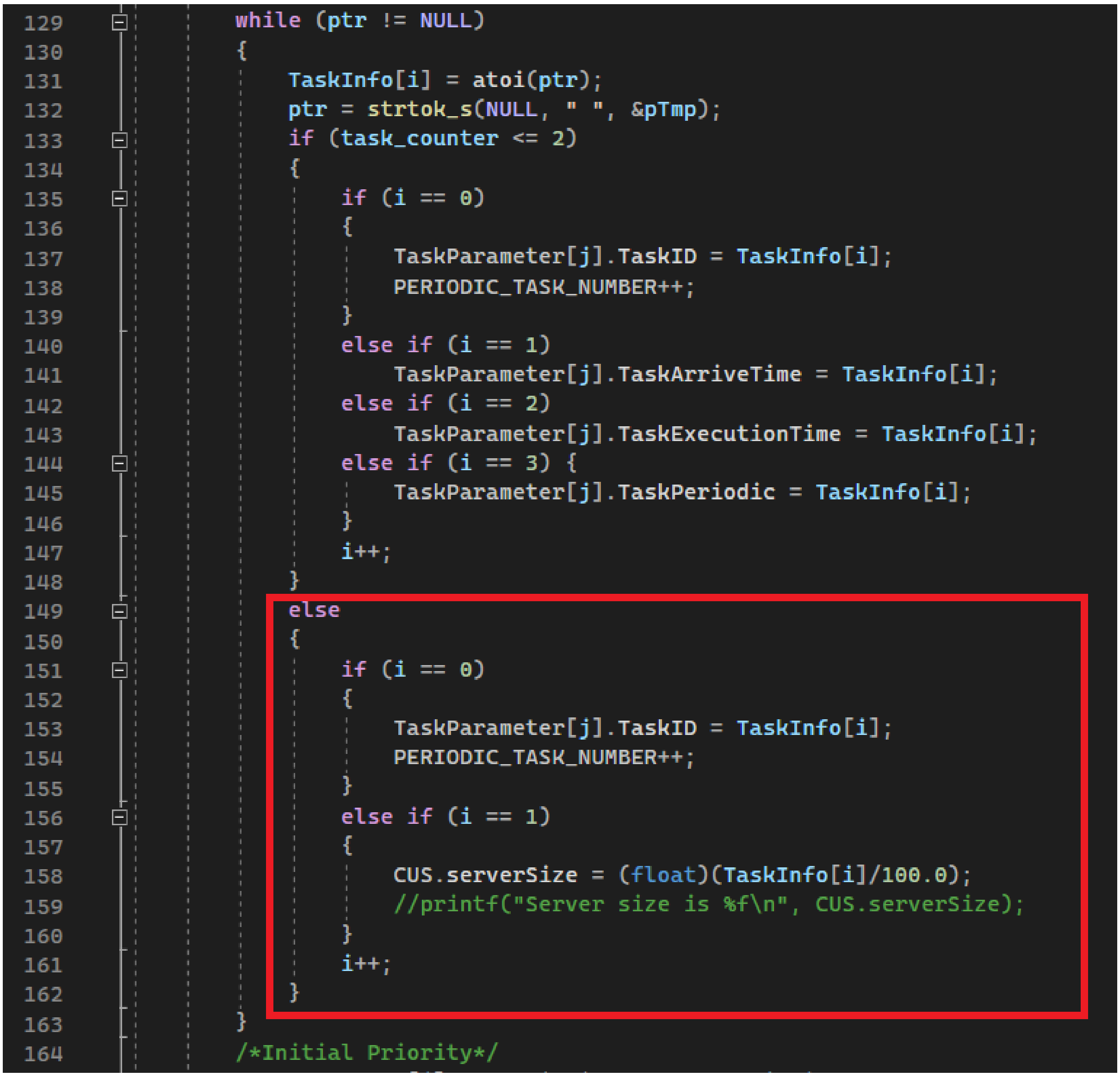
- file: `ucos_ii.h`
- Declare a new structure, cus_para_set, which stores CUS’ s information.
- Declare a new structure, aperiodic_job_para_set, which stores aperiodic job’ s information.

- Declare a new structure, aperiodic_job_node, which stores aperiodic job' s information for CUS's aperiodic hob queue.



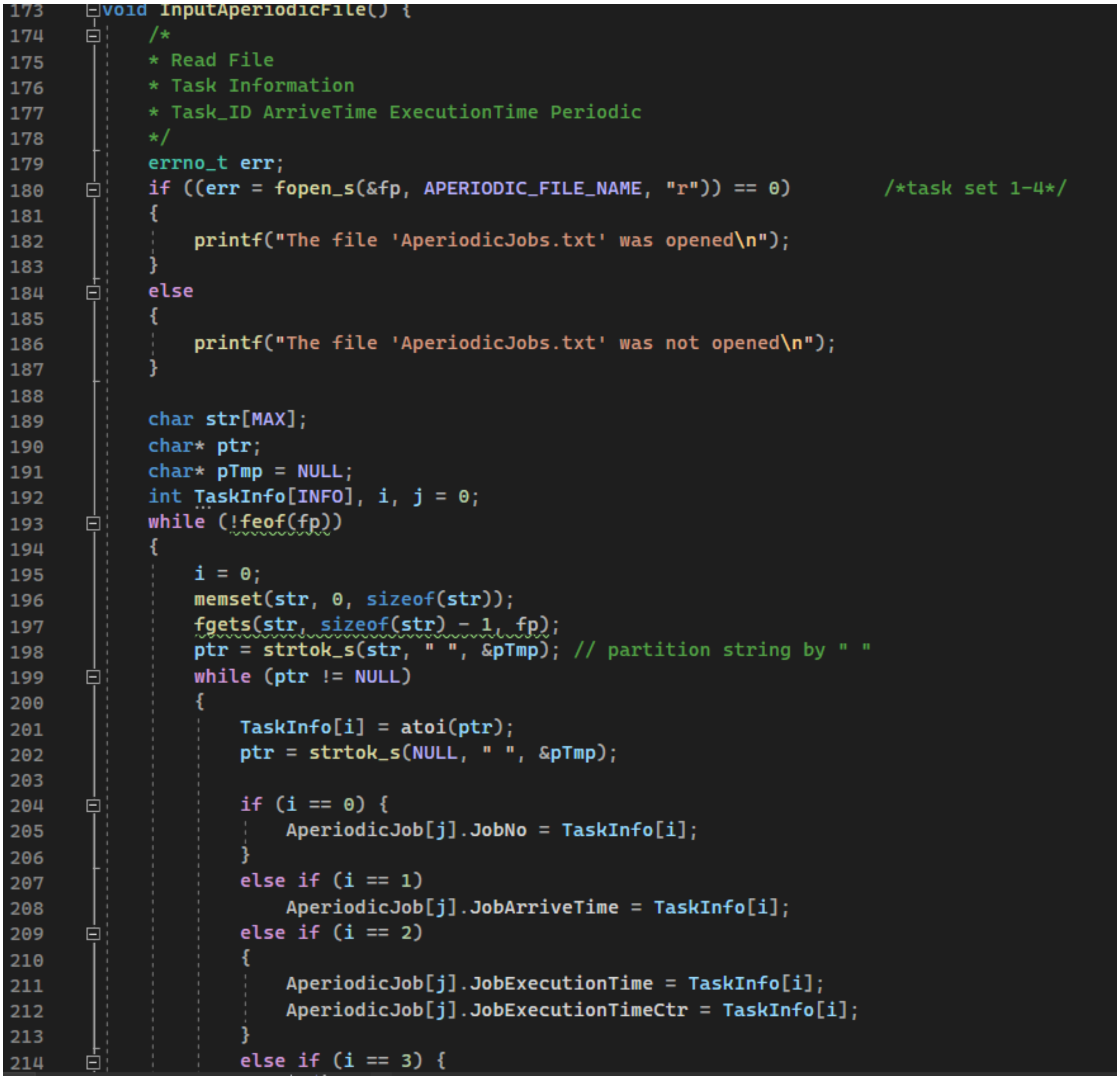
```
OS_EXT   INT32U              OSCtxSwCtr;                 /* Counter of number of context switches
OS_EXT   OS_TASK_NODE*       OSTaskHeapList;             /* Pointer to root of linked list of Task Nod
OS_EXT   INT8U               OSTaskHeapLength;           /* Counter of number of heap node length
OS_EXT   INT8U               resumeCurrTCB;
OS_EXT   aperiodic_job_node*   OSCUSRdyQueue;
OS_EXT   OS_TCB*             CUS_TCB;
OS_EXT   INT8U               isAperiodicJosFinish;
OS_EXT   INT16U              ap_response_time;
OS_EXT   INT16U              ap_preemptive_time;
```

- file: `ucos_ii.h`
- Declare a new global variable, aperiodic_job_node, a pointer points to CUS's aperiodic job queue.
- Declare a new global variable, cus_TCB, a pointer points to TCB of CUS's task.
- Declare a new global variable, isAperiodicJobFinish, a flag represents whether the current aperiodic job is finished or not.
- Declare a new global variable, ap_response_time, stores the response time of aperiodic job.
- Declare a new global variable, ap_preemptive_time, stores the preemptive time of aperiodic job.

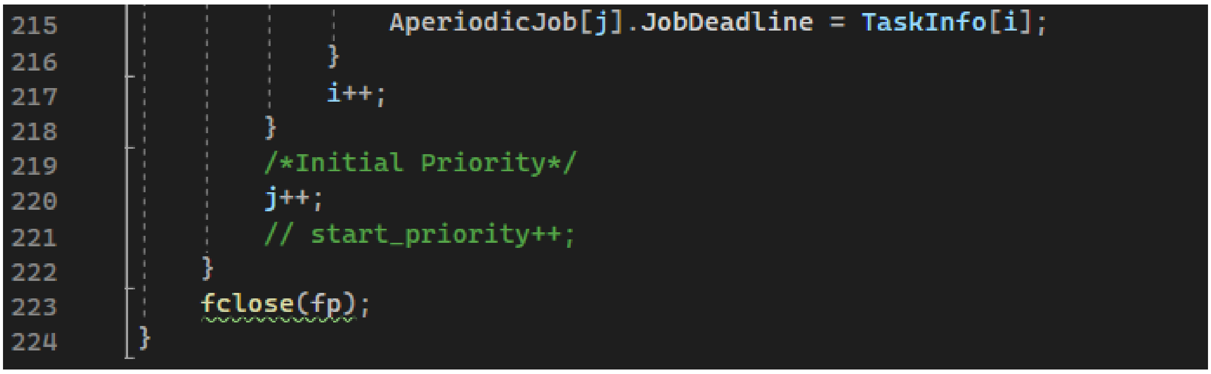


```
while (ptr != NULL)
{
    TaskInfo[i] = atoi(ptr);
    ptr = strtok_s(NULL, " ", &pTmp);
    if (task_counter <= 2)
    {
        if (i == 0)
        {
            TaskParameter[j].TaskID = TaskInfo[i];
            PERIODIC_TASK_NUMBER++;
        }
        else if (i == 1)
            TaskParameter[j].TaskArriveTime = TaskInfo[i];
        else if (i == 2)
            TaskParameter[j].TaskExecutionTime = TaskInfo[i];
        else if (i == 3) {
            TaskParameter[j].TaskPeriodic = TaskInfo[i];
        }
        i++;
    }
    else
    {
        if (i == 0)
        {
            TaskParameter[j].TaskID = TaskInfo[i];
            PERIODIC_TASK_NUMBER++;
        }
        else if (i == 1)
        {
            CUS.serverSize = (float)(TaskInfo[i]/100.0);
            //printf("Server size is %f\n", CUS.serverSize);
        }
        i++;
    }
}
/*Initial Priority*/
```

- file: `ucos_ii.h`->InputPeriodicFile()
- Added functionality to read serverID and serverSize.

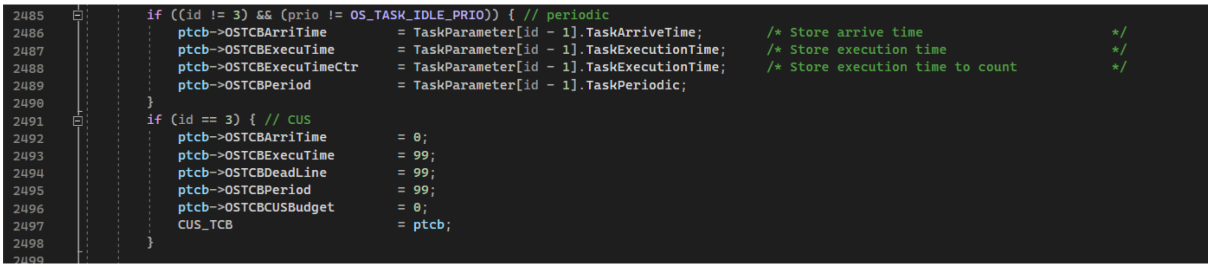


```
void InputAperiodicFile() {
    /*
    * Read File
    * Task Information
    * Task_ID ArriveTime ExecutionTime Periodic
    */
    errno_t err;
    if ((err = fopen_s(&fp, APERIODIC_FILE_NAME, "r")) == 0)        /*task set 1-4*/
    {
        printf("The file 'AperiodicJobs.txt' was opened\n");
    }
    else
    {
        printf("The file 'AperiodicJobs.txt' was not opened\n");
    }

    char str[MAX];
    char* ptr;
    char* pTmp = NULL;
    int TaskInfo[INFO], i, j = 0;
    while (!feof(fp))
    {
        i = 0;
        memset(str, 0, sizeof(str));
        fgets(str, sizeof(str) - 1, fp);
        ptr = strtok_s(str, " ", &pTmp); // partition string by " "
        while (ptr != NULL)
        {
            TaskInfo[i] = atoi(ptr);
            ptr = strtok_s(NULL, " ", &pTmp);

            if (i == 0) {
                AperiodicJob[j].JobNo = TaskInfo[i];
            }
            else if (i == 1)
                AperiodicJob[j].JobArriveTime = TaskInfo[i];
            else if (i == 2)
            {
                AperiodicJob[j].JobExecutionTime = TaskInfo[i];
                AperiodicJob[j].JobExecutionTimeCtr = TaskInfo[i];
            }
            else if (i == 3) {
```



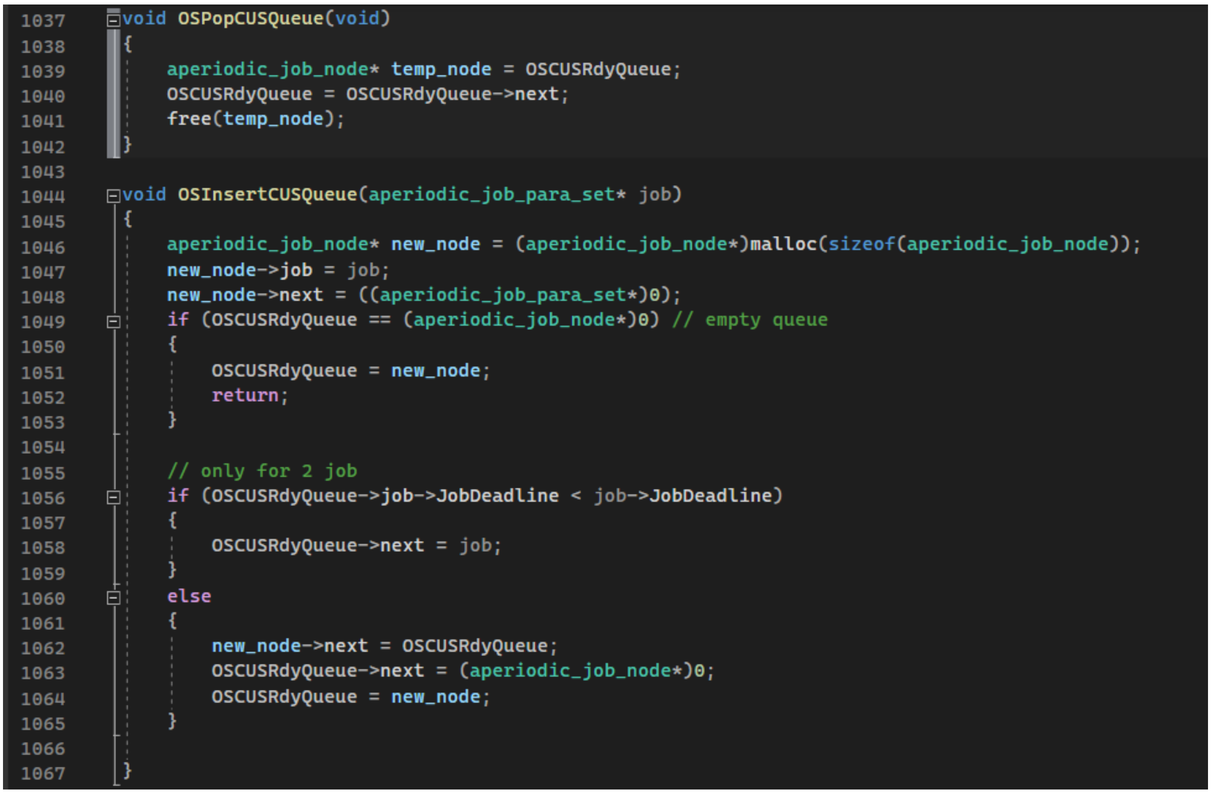
```
                AperiodicJob[j].JobDeadline = TaskInfo[i];
            }
            i++;
        }
        /*Initial Priority*/
        j++;
        // start_priority++;
    }
    fclose(fp);
}
```

- file: `ucos_ii.h`->InputAperiodicFile()
- Added functionality to read aperiodic jobs.



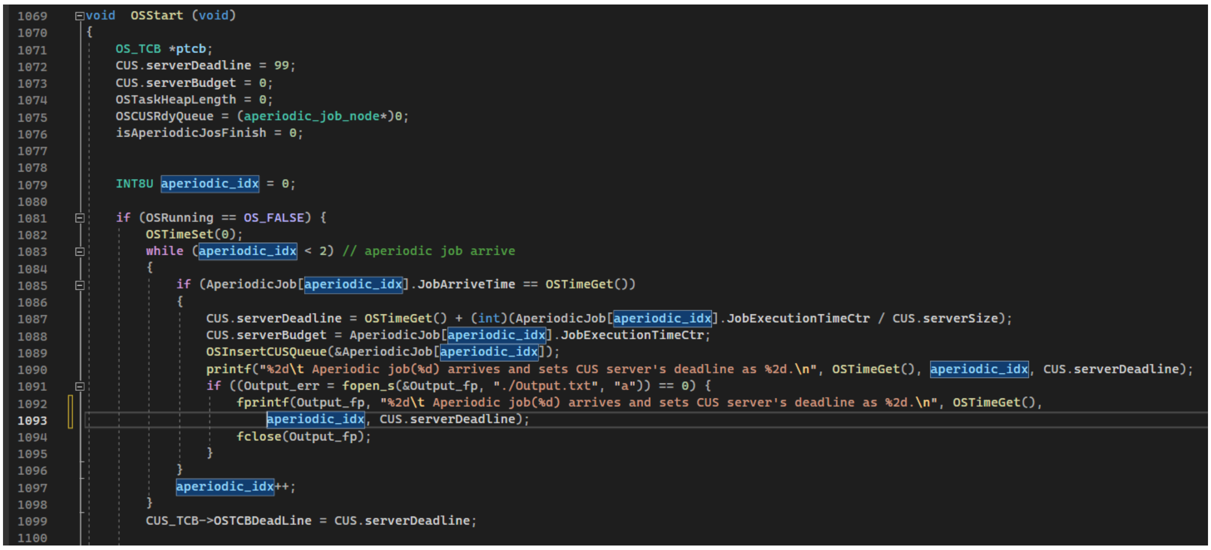
```c
    if ((id != 3) && (prio != OS_TASK_IDLE_PRIO)) { // periodic
        ptcb->OSTCBArriTime         = TaskParameter[id - 1].TaskArriveTime;       /* Store arrive time                */
        ptcb->OSTCBExecuTime        = TaskParameter[id - 1].TaskExecutionTime;    /* Store execution time             */
        ptcb->OSTCBExecuTimeCtr     = TaskParameter[id - 1].TaskExecutionTime;    /* Store execution time to count    */
        ptcb->OSTCBPeriod           = TaskParameter[id - 1].TaskPeriodic;
    }
    if (id == 3) { // CUS
        ptcb->OSTCBArriTime         = 0;
        ptcb->OSTCBExecuTime        = 99;
        ptcb->OSTCBDeadLine         = 99;
        ptcb->OSTCBPeriod           = 99;
        ptcb->OSTCBCUSBudget        = 0;
        CUS_TCB                     = ptcb;
    }
```

- Initialize parameters for the CUS task.



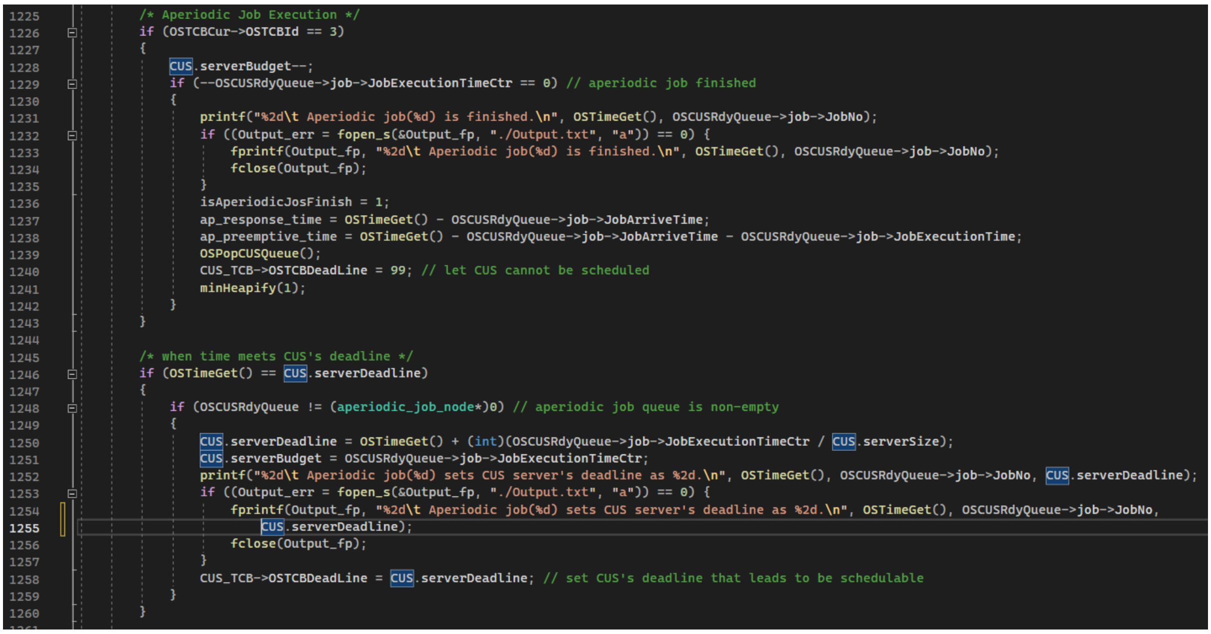
```c
void OSPopCUSQueue(void)
{
    aperiodic_job_node* temp_node = OSCUSRdyQueue;
    OSCUSRdyQueue = OSCUSRdyQueue->next;
    free(temp_node);
}

void OSInsertCUSQueue(aperiodic_job_para_set* job)
{
    aperiodic_job_node* new_node = (aperiodic_job_node*)malloc(sizeof(aperiodic_job_node));
    new_node->job = job;
    new_node->next = ((aperiodic_job_para_set*)0);
    if (OSCUSRdyQueue == (aperiodic_job_node*)0) // empty queue
    {
        OSCUSRdyQueue = new_node;
        return;
    }

    // only for 2 job
    if (OSCUSRdyQueue->job->JobDeadline < job->JobDeadline)
    {
        OSCUSRdyQueue->next = job;
    }
    else
    {
        new_node->next = OSCUSRdyQueue;
        OSCUSRdyQueue->next = (aperiodic_job_node*)0;
        OSCUSRdyQueue = new_node;
    }

}
```

- file: `os_core.c`
- OSPopCUSQueue()
  - After the current aperiodic job is executed, pop it out from the queue.
- OSInsertCUSQueue()
  - When a new aperiodic job arrives, insert it into the queue based on its deadline.



```c
void  OSStart (void)
{
    OS_TCB *ptcb;
    CUS.serverDeadline = 99;
    CUS.serverBudget = 0;
    OSTaskHeapLength = 0;
    OSCUSRdyQueue = (aperiodic_job_node*)0;
    isAperiodicJosFinish = 0;


    INT8U aperiodic_idx = 0;

    if (OSRunning == OS_FALSE) {
        OSTimeSet(0);
        while (aperiodic_idx < 2) // aperiodic job arrive
        {
            if (AperiodicJob[aperiodic_idx].JobArriveTime == OSTimeGet())
            {
                CUS.serverDeadline = OSTimeGet() + (int)(AperiodicJob[aperiodic_idx].JobExecutionTimeCtr / CUS.serverSize);
                CUS.serverBudget = AperiodicJob[aperiodic_idx].JobExecutionTimeCtr;
                OSInsertCUSQueue(&AperiodicJob[aperiodic_idx]);
                printf("%2d\t Aperiodic job(%d) arrives and sets CUS server's deadline as %2d.\n", OSTimeGet(), aperiodic_idx, CUS.serverDeadline);
                if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
                    fprintf(Output_fp, "%2d\t Aperiodic job(%d) arrives and sets CUS server's deadline as %2d.\n", OSTimeGet(),
                        aperiodic_idx, CUS.serverDeadline);
                    fclose(Output_fp);
                }
            }
            aperiodic_idx++;
        }
        CUS_TCB->OSTCBDeadLine = CUS.serverDeadline;

```

- file: `os_core.c`
- Initialize various parameters, with CUS.serverDeadline set to 99.
- Iterate through all aperiodic jobs, checking if any job has arrived. If so, add it to the CUS queue.



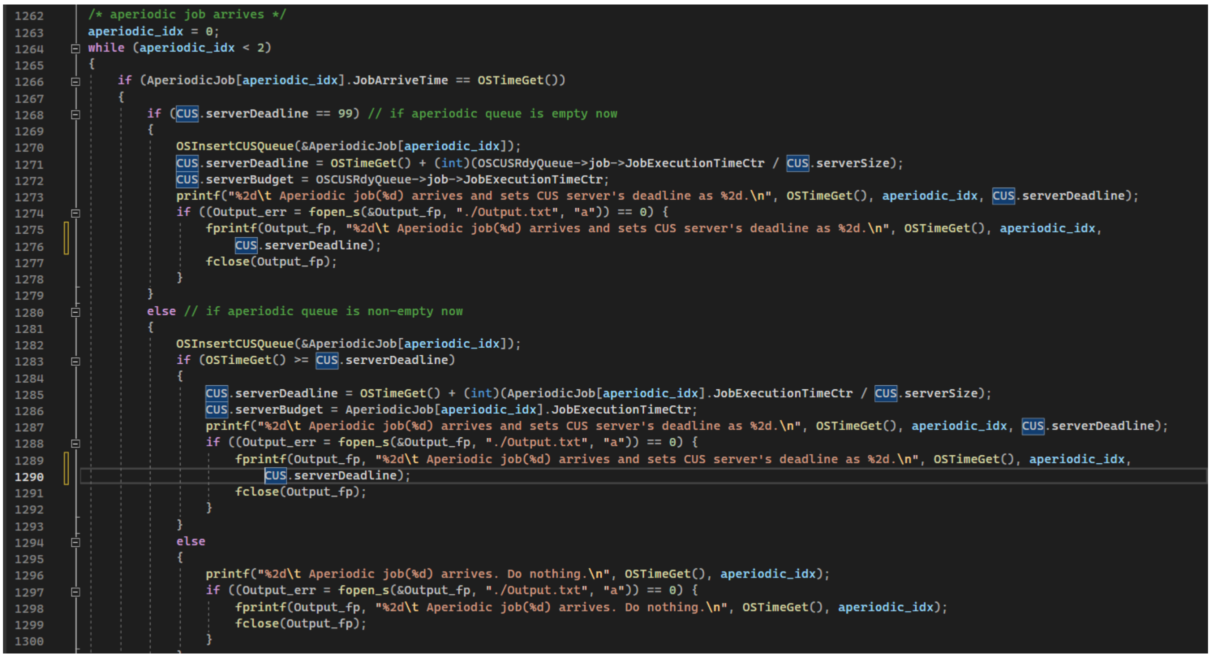
```c
        /* Aperiodic Job Execution */
        if (OSTCBCur->OSTCBId == 3)
        {
            CUS.serverBudget--;
            if (--OSCUSRdyQueue->job->JobExecutionTimeCtr == 0) // aperiodic job finished
            {
                printf("%2d\t Aperiodic job(%d) is finished.\n", OSTimeGet(), OSCUSRdyQueue->job->JobNo);
                if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
                    fprintf(Output_fp, "%2d\t Aperiodic job(%d) is finished.\n", OSTimeGet(), OSCUSRdyQueue->job->JobNo);
                    fclose(Output_fp);
                }
                isAperiodicJosFinish = 1;
                ap_response_time = OSTimeGet() - OSCUSRdyQueue->job->JobArriveTime;
                ap_preemptive_time = OSTimeGet() - OSCUSRdyQueue->job->JobArriveTime - OSCUSRdyQueue->job->JobExecutionTime;
                OSPopCUSQueue();
                CUS_TCB->OSTCBDeadLine = 99; // let CUS cannot be scheduled
                minHeapify(1);
            }
        }

        /* when time meets CUS's deadline */
        if (OSTimeGet() == CUS.serverDeadline)
        {
            if (OSCUSRdyQueue != (aperiodic_job_node*)0) // aperiodic job queue is non-empty
            {
                CUS.serverDeadline = OSTimeGet() + (int)(OSCUSRdyQueue->job->JobExecutionTimeCtr / CUS.serverSize);
                CUS.serverBudget = OSCUSRdyQueue->job->JobExecutionTimeCtr;
                printf("%2d\t Aperiodic job(%d) sets CUS server's deadline as %2d.\n", OSTimeGet(), OSCUSRdyQueue->job->JobNo, CUS.serverDeadline);
                if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
                    fprintf(Output_fp, "%2d\t Aperiodic job(%d) sets CUS server's deadline as %2d.\n", OSTimeGet(), OSCUSRdyQueue->job->JobNo,
                        CUS.serverDeadline);
                    fclose(Output_fp);
                }
                CUS_TCB->OSTCBDeadLine = CUS.serverDeadline; // set CUS's deadline that leads to be schedulable
            }
        }
```

- file: `os_core.c`->TimeTick()
- Aperiodic Job Execution
    - CUS.Bedget minus 1.
    - If the aperiodic job completes its execution, display information, record the current response time and preemptive time, and set the CUS deadline to 99

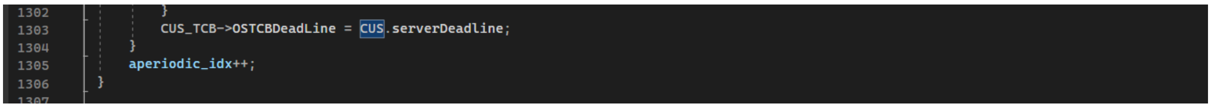
    to prevent EDF from selecting CUS.

    - Set `isAperiodicJobFinish` to 1, allowing it to display the correct information when undergoing a task switch.
    - If the aperiodic job completes its execution, update the jobs in the CUS queue, and perform minHeapify to find the next task with the earliest deadline.

- when time meets CUS's deadline
    - Check if there are any jobs in the current queue that can be processed. If so, update the deadline and budget.

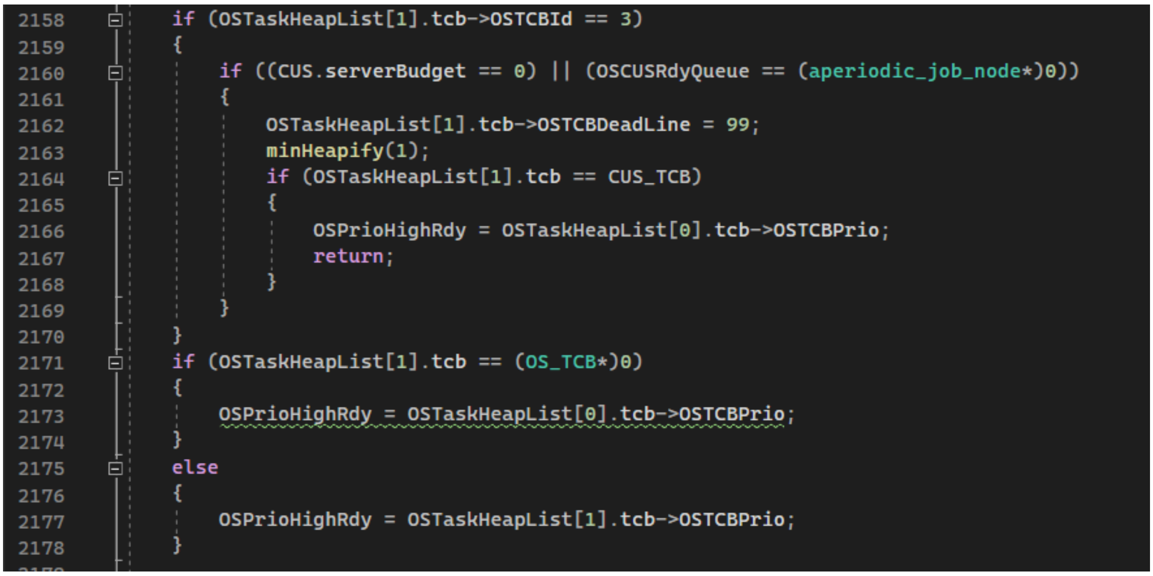


```c
/* aperiodic job arrives */
aperiodic_idx = 0;
while (aperiodic_idx < 2)
{
    if (AperiodicJob[aperiodic_idx].JobArriveTime == OSTimeGet())
    {
        if (CUS.serverDeadline == 99) // if aperiodic queue is empty now
        {
            OSInsertCUSQueue(&AperiodicJob[aperiodic_idx]);
            CUS.serverDeadline = OSTimeGet() + (int)(OSCUSRdyQueue->job->JobExecutionTimeCtr / CUS.serverSize);
            CUS.serverBudget = OSCUSRdyQueue->job->JobExecutionTimeCtr;
            printf("%2d\t Aperiodic job(%d) arrives and sets CUS server's deadline as %2d.\n", OSTimeGet(), aperiodic_idx, CUS.serverDeadline);
            if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
                fprintf(Output_fp, "%2d\t Aperiodic job(%d) arrives and sets CUS server's deadline as %2d.\n", OSTimeGet(), aperiodic_idx,
                    CUS.serverDeadline);
                fclose(Output_fp);
            }
        }
        else // if aperiodic queue is non-empty now
        {
            OSInsertCUSQueue(&AperiodicJob[aperiodic_idx]);
            if (OSTimeGet() >= CUS.serverDeadline)
            {
                CUS.serverDeadline = OSTimeGet() + (int)(AperiodicJob[aperiodic_idx].JobExecutionTimeCtr / CUS.serverSize);
                CUS.serverBudget = AperiodicJob[aperiodic_idx].JobExecutionTimeCtr;
                printf("%2d\t Aperiodic job(%d) arrives and sets CUS server's deadline as %2d.\n", OSTimeGet(), aperiodic_idx, CUS.serverDeadline);
                if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
                    fprintf(Output_fp, "%2d\t Aperiodic job(%d) arrives and sets CUS server's deadline as %2d.\n", OSTimeGet(), aperiodic_idx,
                        CUS.serverDeadline);
                    fclose(Output_fp);
                }
            }
            else
            {
                printf("%2d\t Aperiodic job(%d) arrives. Do nothing.\n", OSTimeGet(), aperiodic_idx);
                if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
                    fprintf(Output_fp, "%2d\t Aperiodic job(%d) arrives. Do nothing.\n", OSTimeGet(), aperiodic_idx);
                    fclose(Output_fp);
                }
```



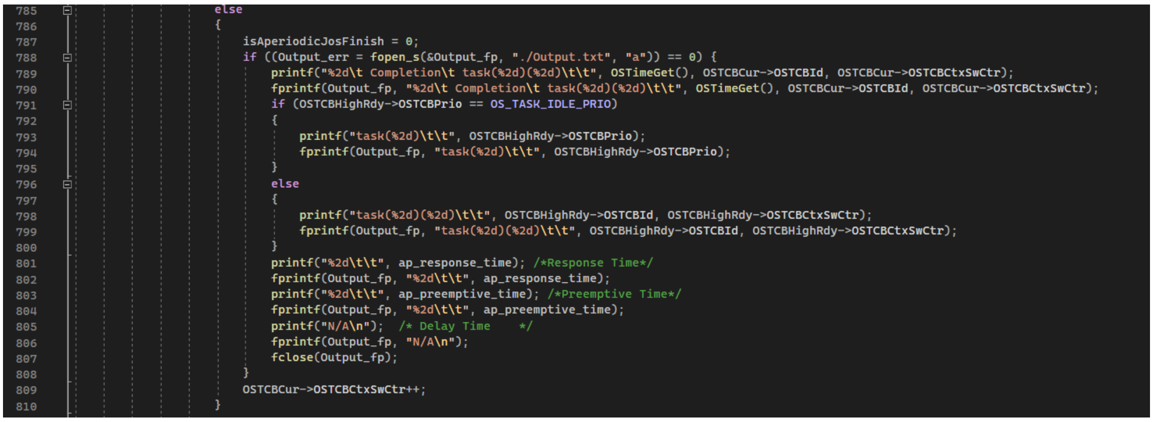
```c
        }
        CUS_TCB->OSTCBDeadLine = CUS.serverDeadline;
    }
    aperiodic_idx++;
}
```

- file: `os_core.c`->TimeTick()
- Aperiodic job arrives
    - Check if the current CUS queue is empty. If it is, add a job and update the CUS deadline and budget.
    - If it is not empty, check whether the CUS deadline has been reached. If the deadline has been reached, update the original deadline and budget; otherwise, do nothing.
    - Add the newly arrived job to the CUS queue.



```c
if (OSTaskHeapList[1].tcb->OSTCBId == 3)
{
    if ((CUS.serverBudget == 0) || (OSCUSRdyQueue == (aperiodic_job_node*)0))
    {
        OSTaskHeapList[1].tcb->OSTCBDeadLine = 99;
        minHeapify(1);
        if (OSTaskHeapList[1].tcb == CUS_TCB)
        {
            OSPrioHighRdy = OSTaskHeapList[0].tcb->OSTCBPrio;
            return;
        }
    }
}
if (OSTaskHeapList[1].tcb == (OS_TCB*)0)
{
    OSPrioHighRdy = OSTaskHeapList[0].tcb->OSTCBPrio;
}
else
{
    OSPrioHighRdy = OSTaskHeapList[1].tcb->OSTCBPrio;
}
```

- file: `os_core.c`->OS_SchedNew()
- Check if the current task with the earliest deadline is an aperiodic job. If it is, ensure that it still has budget available. If it doesn't have budget, then look for the next task with the earliest deadline.
- If the current task only involves CUS and there are no jobs available for execution, execute the idle task.



```c
else
{
    isAperiodicJosFinish = 0;
    if ((Output_err = fopen_s(&Output_fp, "./Output.txt", "a")) == 0) {
        printf("%2d\t Completion\t task(%2d)(%2d)\t\t", OSTimeGet(), OSTCBCur->OSTCBId, OSTCBCur->OSTCBCtxSwCtr);
        fprintf(Output_fp, "%2d\t Completion\t task(%2d)(%2d)\t\t", OSTimeGet(), OSTCBCur->OSTCBId, OSTCBCur->OSTCBCtxSwCtr);
        if (OSTCBHighRdy->OSTCBPrio == OS_TASK_IDLE_PRIO)
        {
            printf("task(%2d)\t\t", OSTCBHighRdy->OSTCBPrio);
            fprintf(Output_fp, "task(%2d)\t\t", OSTCBHighRdy->OSTCBPrio);
        }
        else
        {
            printf("task(%2d)(%2d)\t\t", OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBCtxSwCtr);
            fprintf(Output_fp, "task(%2d)(%2d)\t\t", OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBCtxSwCtr);
        }
        printf("%2d\t\t", ap_response_time); /*Response Time*/
        fprintf(Output_fp, "%2d\t\t", ap_response_time);
        printf("%2d\t\t", ap_preemptive_time); /*Preemptive Time*/
        fprintf(Output_fp, "%2d\t\t", ap_preemptive_time);
        printf("N/A\n");  /* Delay Time    */
        fprintf(Output_fp, "N/A\n");
        fclose(Output_fp);
    }
    OSTCBCur->OSTCBCtxSwCtr++;
}
```

- file: `os_core.c`->OS_IntExit()
- Display information in the requested format when an aperiodic job completes.