

# PA2

**M11202117 呂長恩**

## Part 1 EDF

## Miss Deadline Handler

```
* Miss Deadline */
if ((OSTimeGet() - ptcb->OSTCBArrTime == ptcb->OSTCBPeriod) && (ptcb->OSTCBExecuTimeCtr > 0)) {
    printf("%2d\t MissDeadline\t task(%2d)(%2d)\t\t----- \n", OSTimeGet(), ptcb->OSTCBIId, ptcb->OSTCBCtxSwCtr);
    if (fopen_err == fopen_s(&Output_fp, ".\\Output.txt", "a")) == 0) {
        fprintf(Output_fp, "%2d\t MissDeadline\t task(%2d)(%2d)\t\t----- \n", OSTimeGet(), ptcb->OSTCBIId, 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.

```

101
102     /* Create Task Node Structure */
103     typedef struct os_task_node {
104         struct os_tcb      *tcb;
105
106     } OS_TASK_NODE;
107

```

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

```

751     OS_EXT  OS_TASK_NODE*   OSTaskHeapList;
752     OS_EXT  INT8U           OSTaskHeapLength;
753     OS_EXT  INT8U           resumeCurrTCB;
754

```

- 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.

```

976 void OSInsertTaskHeap (OS_TCB* ptcb)
977 {
978     OSTaskHeapLength++;
979     INT8U idx, parent;
980     idx = OSTaskHeapLength;
981     parent = idx / 2;
982
983     if (OSTaskHeapList[1].tcb == (OS_TASK_NODE*)0) { // if heap is empty
984         OSTaskHeapList[1].tcb = ptcb;
985         return;
986     }
987     OSTaskHeapList[OSTaskHeapLength].tcb = ptcb;
988
989     while ((idx > 1))
990     {
991         if ((OSTaskHeapList[parent].tcb->OSTCBDeadLine > OSTaskHeapList[idx].tcb->OSTCBDeadLine) ||
992             ((OSTaskHeapList[parent].tcb->OSTCBDeadLine == OSTaskHeapList[idx].tcb->OSTCBDeadLine) &&
993              (OSTaskHeapList[parent].tcb->OSTCBId > OSTaskHeapList[idx].tcb->OSTCBId)))
994         {
995             OS_TCB* temp_tcb = OSTaskHeapList[parent].tcb;
996             OSTaskHeapList[parent].tcb = OSTaskHeapList[idx].tcb;
997             OSTaskHeapList[idx].tcb = temp_tcb;
998             idx = parent;
999             parent = idx / 2;
1000         }
1001         else
1002         {
1003             break;
1004         }
1005     }
1006 }
1007

```

- 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.

```

939 void minHeapify(INT8U idx)
940 {
941     INT8U left, right, smallest;
942     left = idx * 2;
943     right = idx * 2 + 1;
944     smallest = idx;
945     if (left <= OSTaskHeapLength)
946     {
947         if (OSTaskHeapList[left].tcb->OSTCBDeadLine < OSTaskHeapList[idx].tcb->OSTCBDeadLine)
948         {
949             smallest = left;
950         }
951         if ((OSTaskHeapList[left].tcb->OSTCBDeadLine == OSTaskHeapList[idx].tcb->OSTCBDeadLine) &&
952             (OSTaskHeapList[left].tcb->OSTCBPrio < OSTaskHeapList[idx].tcb->OSTCBPrio))
953         {
954             smallest = left;
955         }
956     }
957     if (right <= OSTaskHeapLength)
958     {
959         if (OSTaskHeapList[right].tcb->OSTCBDeadLine < OSTaskHeapList[idx].tcb->OSTCBDeadLine)
960         {
961             smallest = right;
962         }
963         if ((OSTaskHeapList[right].tcb->OSTCBDeadLine == OSTaskHeapList[idx].tcb->OSTCBDeadLine) &&
964             (OSTaskHeapList[right].tcb->OSTCBPrio < OSTaskHeapList[idx].tcb->OSTCBPrio))
965         {
966             smallest = right;
967         }
968     }
969     if (smallest != idx)
970     {
971         OS_TCB* temp_tcb = OSTaskHeapList[idx].tcb;
972         OSTaskHeapList[idx].tcb = OSTaskHeapList[smallest].tcb;
973         OSTaskHeapList[smallest].tcb = temp_tcb;
974         minHeapify(smallest);
975     }
976 }

```

- 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.

```

1010 void OSStart (void)
1011 {
1012     OS_TCB *ptcb;
1013     OSTaskHeapLength = 0;
1014
1015     if (OSRunning == OS_FALSE) {
1016         OSTimeSet(0);
1017         ptcb = OSTCBLIST;
1018         while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {
1019             if (ptcb->OSTCBArriTime == OSTimeGet()) {
1020                 ptcb->OSTCBDeadLine = OSTimeGet() + ptcb->OSTCBPeriod;
1021                 OSInsertTaskHeap(ptcb);
1022             }
1023             ptcb = ptcb->OSTCBNext;
1024         }
1025
1026         OS_SchedNew(); /* Find highest priority's task priority number */
1027         OSPrioHighRdy = OSPrioHighRdy;
1028         OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy]; /* Point to highest priority task ready to run */
1029         OSTCBCur = OSTCBHighRdy;
1030
1031         printf("Tick\tEvent\t\tCurrentTask ID\t\tNextTaskID\tResponseTime\tPreemptionTime\tOSTimeDly\n");
1032         if ((Output_err = fopen_s(&Output_fp, ".\\Output.txt", "a")) == 0) {
1033             fprintf(Output_fp, "Tick\tEvent\t\tCurrentTask ID\t\tNextTaskID\tResponseTime\tPreemptionTime\tOSTimeDly\n");
1034             fclose(Output_fp);
1035         }
1036         OSStartHighRdy(); /* Execute target specific code to start task */
1037     }
1038 }
1039

```

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

```

1972 static void OS_SchedNew (void) /* Find the highest priority task */
1973 {
1974     #if OS_LOWEST_PRIO <= 63u /* See if we support up to 64 tasks */
1975         INT8U y;
1976
1977         /*y
1978          = OSUnMapTbl[OSRdyGrp];
1979         OSPrioHighRdy = (INT8U)((y << 3u) + OSUnMapTbl[OSRdyTbl[y]]);*/
1980
1981         if (OSTaskHeapList[1].tcb != (OS_TCB*)0)
1982         {
1983             OSPrioHighRdy = OSTaskHeapList[1].tcb->OSTCBPrio;
1984         }
1985         else
1986         {
1987             OSPrioHighRdy = OSTaskHeapList[0].tcb->OSTCBPrio;
1988         }
1989

```

- 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.

```

1127 if (OSRunning == OS_TRUE) {
1128     /*Exeuction Time*/
1129     if (--OSTCBCur->OSTCBExecuTimeCtr == 0u) { /* Executing */
1130         OSTaskHeapList[1].tcb = OSTaskHeapList[OSTaskHeapLength].tcb;
1131         OSTaskHeapList[OSTaskHeapLength].tcb = (OS_TCB*)0;
1132         OSTaskHeapLength--;
1133         minHeapify(1);
1134     }
1135 }

```

- file: `os_core.c`

- The code above is in the TimeTick() function. When TimeTick() is triggered, 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.

```

1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203

/* Arrive time */
if (ptcb->OSTCBArriTime == OSTimeGet()) { /* Arrive Time */
    ptcb->OSTCBDeadLine = OSTimeGet() + ptcb->OSTCBPeriod;
    OSInsertTaskHeap(ptcb);
}

/* Resume 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.

```

735     }
736 }
737 else {
738     printf("%2d\t Completion\t task(%2d)(%2d)\t\t", OSTimeGet(), OSTCBCur->OSTCBId, OSTCBCur->OSTCBctxSwCtr);
739     printf("task(%2d)(%2d)\t\t", OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBctxSwCtr);
740     printf("%2d\t\t", OSTimeGet() - OSTCBCur->OSTCBArriTime);
741     printf("%2d\t\t", OSTimeGet() - OSTCBCur->OSTCBArriTime - OSTCBCur->OSTCBExecuTime); /*Preemptive Time*/
742     printf("%2d\n", OSTCBCur->OSTCBPeriod - (OSTimeGet() - OSTCBCur->OSTCBArriTime)); /* Delay Time */
743     if ((Output_err = fopen_s(&Output_fp, ".\\Output.txt", "a")) == 0) {
744         fprintf(Output_fp, "%2d\t Completion\t task(%2d)(%2d)\t\t", OSTimeGet(), OSTCBCur->OSTCBId, OSTCBCur->OSTCBctxSwCtr);
745         fprintf(Output_fp, "task(%2d)(%2d)\t\t", OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBctxSwCtr);
746         fprintf(Output_fp, "%2d\t\t", OSTimeGet() - OSTCBCur->OSTCBArriTime);
747         fprintf(Output_fp, "%2d\t\t", OSTimeGet() - OSTCBCur->OSTCBArriTime - OSTCBCur->OSTCBExecuTime); /*Preemptive Time*/
748         fprintf(Output_fp, "%2d\n", OSTCBCur->OSTCBPeriod - (OSTimeGet() - OSTCBCur->OSTCBArriTime)); /* Delay Time */
749         fclose(Output_fp);
750     }
751 }
752 OSTCBCur->OSTCBctxSwCtr++;
753 if (ResumeCurrTCB == 1)
754 {
755     OSTCBCur->OSTCBArriTime = OSTimeGet();
756     OSTCBCur->OSTCBExecuTimeCtr = OSTCBCur->OSTCBExecuTime;
757     OSTCBCur->OSTCBDeadline = OSTimeGet() + OSTCBCur->OSTCBPeriod;
758     OSInsertTaskHeap(OSTCBCur);
759     resumeCurrTCB = 0;
760 }
761 }
762 else {
763     printf("%2d\t Preemption\t task(%2d)(%2d)\t\t", OSTimeGet(), OSTCBCur->OSTCBId, OSTCBCur->OSTCBctxSwCtr);
764     printf("task(%2d)(%2d)\t \n", OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBctxSwCtr);
765     if ((Output_err = fopen_s(&Output_fp, ".\\Output.txt", "a")) == 0) {
766         fprintf(Output_fp, "%2d\t Preemption\t task(%2d)(%2d)\t\t", OSTimeGet(), OSTCBCur->OSTCBId, OSTCBCur->OSTCBctxSwCtr);
767         fprintf(Output_fp, "task(%2d)(%2d)\t \n", OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBctxSwCtr);
768         fclose(Output_fp);
769     }
770 }

```

- 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.

```

97  /* Constant Utilization Server */
98  typedef struct cus_para_set {
99      float serverSize;
100     INT16U serverBudget;
101     INT16U serverDeadline;
102 } cus_para_set;
103 /* Constant Utilization Server */
104
105 /* Aperiodic Task Structure */
106 typedef struct aperiodic_job_para_set {
107     INT16U JobNo;
108     INT16U JobArriveTime;
109     INT16U JobExecutionTime;
110     INT16U JobExecutionTimeCtr;
111     INT16U JobDeadline;
112 } aperiodic_job_para_set;
113
114 typedef struct aperiodic_job_node {
115     struct aperiodic_job_para_set* job;
116     struct aperiodic_job_para_set* next;
117 } aperiodic_job_node;
118
119 /* Aperiodic Task Structure */
120
121 /*Dynamic Create the Stack Size*/
122 OS_STK** Task_STK;
123
124 /*Create Task*/
125 task_para_set          TaskParameter[OS_MAX_TASKS];
126 aperiodic_job_para_set AperiodicJob[2];
127 cus_para_set           CUS;
128
129 /* Create Task Node Structure */
130 typedef struct os_task_node {
131     struct os_tcb          *tcb;
132
133 } 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 job queue.

```

775  /*an32u*/
776  OS_EXT INT32U      OSCtxSwCtr;          /* Counter of number of context switches
777  OS_EXT OS_TASK_NODE* OSTaskHeapList;    /* Pointer to root of linked list of Task Node
778  OS_EXT INT8U       OSTaskHeapLength;    /* Counter of number of heap node length
779  OS_EXT INT8U       resumeCurrTCB;
780  OS_EXT aperiodic_job_node* OSCUSRdyQueue;
781  OS_EXT OS_TCB*     CUS_TCB;
782  OS_EXT INT8U       isAperiodicJobFinish;
783  OS_EXT INT16U      ap_response_time;
784  OS_EXT INT16U      ap_preemptive_time;
785

```

- 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.



```

129 while (ptr != NULL)
130 {
131     TaskInfo[i] = atoi(ptr);
132     ptr = strtok_s(NULL, " ", &pTmp);
133     if (task_counter <= 2)
134     {
135         if (i == 0)
136         {
137             TaskParameter[j].TaskID = TaskInfo[i];
138             PERIODIC_TASK_NUMBER++;
139         }
140         else if (i == 1)
141             TaskParameter[j].TaskArriveTime = TaskInfo[i];
142         else if (i == 2)
143             TaskParameter[j].TaskExecutionTime = TaskInfo[i];
144         else if (i == 3) {
145             TaskParameter[j].TaskPeriodic = TaskInfo[i];
146         }
147         i++;
148     }
149     else
150     {
151         if (i == 0)
152         {
153             TaskParameter[j].TaskID = TaskInfo[i];
154             PERIODIC_TASK_NUMBER++;
155         }
156         else if (i == 1)
157         {
158             CUS.serverSize = (float)(TaskInfo[i]/100.0);
159             //printf("Server size is %f\n", CUS.serverSize);
160         }
161         i++;
162     }
163 }
164 /*Initial Priority*/

```

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

```

173 void InputAperiodicFile() {
174     /*
175      * Read File
176      * Task Information
177      * Task_ID ArriveTime ExecutionTime Periodic
178      */
179     errno_t err;
180     if ((err = fopen_s(&fp, APERIODIC_FILE_NAME, "r")) == 0) /*task set 1-4*/
181     {
182         printf("The file 'AperiodicJobs.txt' was opened\n");
183     }
184     else
185     {
186         printf("The file 'AperiodicJobs.txt' was not opened\n");
187     }
188
189     char str[MAX];
190     char* ptr;
191     char* pTmp = NULL;
192     int TaskInfo[INFO], i, j = 0;
193     while (!feof(fp))
194     {
195         i = 0;
196         memset(str, 0, sizeof(str));
197         fgets(str, sizeof(str) - 1, fp);
198         ptr = strtok_s(str, " ", &pTmp); // partition string by " "
199         while (ptr != NULL)
200         {
201             TaskInfo[i] = atoi(ptr);
202             ptr = strtok_s(NULL, " ", &pTmp);
203
204             if (i == 0) {
205                 AperiodicJob[j].JobNo = TaskInfo[i];
206             }
207             else if (i == 1)
208                 AperiodicJob[j].JobArriveTime = TaskInfo[i];
209             else if (i == 2)
210             {
211                 AperiodicJob[j].JobExecutionTime = TaskInfo[i];
212                 AperiodicJob[j].JobExecutionTimeCtr = TaskInfo[i];
213             }
214             else if (i == 3) {
215                 AperiodicJob[j].JobDeadline = TaskInfo[i];
216             }
217             i++;
218         }
219         /*Initial Priority*/
220         j++;
221         // start_priority++;
222     }
223     fclose(fp);
224 }

```

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

```

2485     if ((id != 3) && (prio != OS_TASK_IDLE_PRIO)) { // periodic
2486         ptcb->OSTCBArriTime = TaskParameter[id - 1].TaskArriveTime; /* Store arrive time */
2487         ptcb->OSTCBExecuTime = TaskParameter[id - 1].TaskExecutionTime; /* Store execution time */
2488         ptcb->OSTCBExecuTimeCtr = TaskParameter[id - 1].TaskExecutionTime; /* Store execution time to count */
2489         ptcb->OSTCBPeriod = TaskParameter[id - 1].TaskPeriodic;
2490     }
2491     if (id == 3) { // CUS
2492         ptcb->OSTCBArriTime = 0;
2493         ptcb->OSTCBExecuTime = 99;
2494         ptcb->OSTCBDeadline = 99;
2495         ptcb->OSTCBPeriod = 99;
2496         ptcb->OSTCBCUSBudget = 0;
2497         CUS_TCB = ptcb;
2498     }
2499

```

- Initialize parameters for the CUS task.

```

1037 void OSPopCUSQueue(void)
1038 {
1039     aperiodic_job_node* temp_node = OSCUSRdyQueue;
1040     OSCUSRdyQueue = OSCUSRdyQueue->next;
1041     free(temp_node);
1042 }
1043
1044 void OSInsertCUSQueue(aperiodic_job_para_set* job)
1045 {
1046     aperiodic_job_node* new_node = (aperiodic_job_node*)malloc(sizeof(aperiodic_job_node));
1047     new_node->job = job;
1048     new_node->next = ((aperiodic_job_para_set*)0);
1049     if (OSCUSRdyQueue == (aperiodic_job_node*)0) // empty queue
1050     {
1051         OSCUSRdyQueue = new_node;
1052         return;
1053     }
1054
1055     // only for 2 job
1056     if (OSCUSRdyQueue->job->JobDeadline < job->JobDeadline)
1057     {
1058         OSCUSRdyQueue->next = job;
1059     }
1060     else
1061     {
1062         new_node->next = OSCUSRdyQueue;
1063         OSCUSRdyQueue->next = (aperiodic_job_node*)0;
1064         OSCUSRdyQueue = new_node;
1065     }
1066 }
1067

```

- 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.

```

1069 void OSStart (void)
1070 {
1071     OS_TCB *ptcb;
1072     CUS.serverDeadline = 99;
1073     CUS.serverBudget = 0;
1074     OSTaskHeapLength = 0;
1075     OSCUSRdyQueue = (Aperiodic_job_node*)0;
1076     isAperiodicJosFinish = 0;
1077
1078     INT8U aperiodic_idx = 0;
1079
1080     if (OSRunning == OS_FALSE) {
1081         OSTimeSet(0);
1082         while (aperiodic_idx < 2) // aperiodic job arrive
1083         {
1084             if (AperiodicJob[aperiodic_idx].JobArriveTime == OSTimeGet())
1085             {
1086                 CUS.serverDeadline = OSTimeGet() + (int)(AperiodicJob[aperiodic_idx].JobExecutionTimeCtr / CUS.serverSize);
1087                 CUS.serverBudget = AperiodicJob[aperiodic_idx].JobExecutionTimeCtr;
1088                 OSInsertCUSQueue(&AperiodicJob[aperiodic_idx]);
1089                 printf("%2d\t Aperiodic job(%d) arrives and sets CUS server's deadline as %2d.\n", OSTimeGet(), aperiodic_idx, CUS.serverDeadline);
1090                 if ((Output_err = fopen_s(&Output_fp, ".\\Output.txt", "a")) == 0) {
1091                     fprintf(Output_fp, "%2d\t Aperiodic job(%d) arrives and sets CUS server's deadline as %2d.\n", OSTimeGet(),
1092                         aperiodic_idx, CUS.serverDeadline);
1093                     fclose(Output_fp);
1094                 }
1095             }
1096             aperiodic_idx++;
1097         }
1098         CUS_TCB->OSTCBDeadline = CUS.serverDeadline;
1099     }
1100 }

```

- 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.

```

1225 /* Aperiodic Job Execution */
1226 if (OSTCBCur->OSTCBId == 3)
1227 {
1228     CUS.serverBudget--;
1229     if (-OSCUSRdyQueue->job->JobExecutionTimeCtr == 0) // aperiodic job finished
1230     {
1231         printf("%2d\t Aperiodic job(%d) is finished.\n", OSTimeGet(), OSCUSRdyQueue->job->JobNo);
1232         if ((Output_err = fopen_s(&Output_fp, ".\\Output.txt", "a")) == 0) {
1233             fprintf(Output_fp, "%2d\t Aperiodic job(%d) is finished.\n", OSTimeGet(), OSCUSRdyQueue->job->JobNo);
1234             fclose(Output_fp);
1235         }
1236         isAperiodicJosFinish = 1;
1237         ap_response_time = OSTimeGet() - OSCUSRdyQueue->job->JobArriveTime;
1238         ap_preemptive_time = OSTimeGet() - OSCUSRdyQueue->job->JobArriveTime - OSCUSRdyQueue->job->JobExecutionTime;
1239         OSPopCUSQueue();
1240         CUS_TCB->OSTCBDeadline = 99; // let CUS cannot be scheduled
1241         minHeapify(1);
1242     }
1243 }
1244
1245 /* when time meets CUS's deadline */
1246 if (OSTimeGet() == CUS.serverDeadline)
1247 {
1248     if (OSCUSRdyQueue != (Aperiodic_job_node*)0) // aperiodic job queue is non-empty
1249     {
1250         CUS.serverDeadline = OSTimeGet() + (int)(OSCUSRdyQueue->job->JobExecutionTimeCtr / CUS.serverSize);
1251         CUS.serverBudget = OSCUSRdyQueue->job->JobExecutionTimeCtr;
1252         printf("%2d\t Aperiodic job(%d) sets CUS server's deadline as %2d.\n", OSTimeGet(), OSCUSRdyQueue->job->JobNo, CUS.serverDeadline);
1253         if ((Output_err = fopen_s(&Output_fp, ".\\Output.txt", "a")) == 0) {
1254             fprintf(Output_fp, "%2d\t Aperiodic job(%d) sets CUS server's deadline as %2d.\n", OSTimeGet(), OSCUSRdyQueue->job->JobNo,
1255                 CUS.serverDeadline);
1256             fclose(Output_fp);
1257         }
1258         CUS_TCB->OSTCBDeadline = CUS.serverDeadline; // set CUS's deadline that leads to be schedulable
1259     }
1260 }

```

- 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.

```
1262  /* aperiodic job arrives */
1263  aperiodic_idx = 0;
1264  while (aperiodic_idx < 2)
1265  {
1266      if (AperiodicJob[aperiodic_idx].JobArriveTime == OSTimeGet())
1267      {
1268          if (CUS.serverDeadline == 99) // if aperiodic queue is empty now
1269          {
1270              OSInsertCUSQueue(&AperiodicJob[aperiodic_idx]);
1271              CUS.serverDeadline = OSTimeGet() + (int)(OSCUSRdyQueue->job->JobExecutionTimeCtr / CUS.serverSize);
1272              CUS.serverBudget = OSCUSRdyQueue->job->JobExecutionTimeCtr;
1273              printf("%2d\t Aperiodic job(%d) arrives and sets CUS server's deadline as %2d.\n", OSTimeGet(), aperiodic_idx, CUS.serverDeadline);
1274              if ((Output_err = fopen_s(&Output_fp, ".\\Output.txt", "a")) == 0) {
1275                  fprintf(Output_fp, "%2d\t Aperiodic job(%d) arrives and sets CUS server's deadline as %2d.\n", OSTimeGet(), aperiodic_idx,
1276                      CUS.serverDeadline);
1277                  fclose(Output_fp);
1278              }
1279          }
1280          else // if aperiodic queue is non-empty now
1281          {
1282              OSInsertCUSQueue(&AperiodicJob[aperiodic_idx]);
1283              if (OSTimeGet() >= CUS.serverDeadline)
1284              {
1285                  CUS.serverDeadline = OSTimeGet() + (int)(AperiodicJob[aperiodic_idx].JobExecutionTimeCtr / CUS.serverSize);
1286                  CUS.serverBudget = AperiodicJob[aperiodic_idx].JobExecutionTimeCtr;
1287                  printf("%2d\t Aperiodic job(%d) arrives and sets CUS server's deadline as %2d.\n", OSTimeGet(), aperiodic_idx, CUS.serverDeadline);
1288                  if ((Output_err = fopen_s(&Output_fp, ".\\Output.txt", "a")) == 0) {
1289                      fprintf(Output_fp, "%2d\t Aperiodic job(%d) arrives and sets CUS server's deadline as %2d.\n", OSTimeGet(), aperiodic_idx,
1290                          CUS.serverDeadline);
1291                      fclose(Output_fp);
1292                  }
1293              }
1294              else
1295              {
1296                  printf("%2d\t Aperiodic job(%d) arrives. Do nothing.\n", OSTimeGet(), aperiodic_idx);
1297                  if ((Output_err = fopen_s(&Output_fp, ".\\Output.txt", "a")) == 0) {
1298                      fprintf(Output_fp, "%2d\t Aperiodic job(%d) arrives. Do nothing.\n", OSTimeGet(), aperiodic_idx);
1299                      fclose(Output_fp);
1300                  }
1301              }
1302          }
1303          CUS_TCB->OSTCBDeadline = CUS.serverDeadline;
1304          aperiodic_idx++;
1305      }
1306  }
1307
```

- 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.

```

2158     if (OSTaskHeapList[1].tcb->OSTCBId == 3)
2159     {
2160         if ((CUS.serverBudget == 0) || (OSCUSRdyQueue == (aperiodic_job_node*)0))
2161         {
2162             OSTaskHeapList[1].tcb->OSTCBDeadline = 99;
2163             minHeapify(1);
2164             if (OSTaskHeapList[1].tcb == CUS_TCB)
2165             {
2166                 OSPrioHighRdy = OSTaskHeapList[0].tcb->OSTCBPrio;
2167                 return;
2168             }
2169         }
2170     }
2171     if (OSTaskHeapList[1].tcb == (OS_TCB*)0)
2172     {
2173         OSPrioHighRdy = OSTaskHeapList[0].tcb->OSTCBPrio;
2174     }
2175     else
2176     {
2177         OSPrioHighRdy = OSTaskHeapList[1].tcb->OSTCBPrio;
2178     }

```

- 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.

```

785     else
786     {
787         isAperiodicJobFinish = 0;
788         if ((Output_err = fopen_s(&Output_fp, ".Output.txt", "a")) == 0) {
789             printf("%2d\t Completion\t task(%2d)\t\t", OSTimeGet(), OSTCBCur->OSTCBId, OSTCBCur->OSTCBCtxSwCtr);
790             fprintf(Output_fp, "%2d\t Completion\t task(%2d)\t\t", OSTimeGet(), OSTCBCur->OSTCBId, OSTCBCur->OSTCBCtxSwCtr);
791             if (OSTCBHighRdy->OSTCBPrio == OS_TASK_IDLE_PRIO)
792             {
793                 printf("task(%2d)\t\t", OSTCBHighRdy->OSTCBPrio);
794                 fprintf(Output_fp, "task(%2d)\t\t", OSTCBHighRdy->OSTCBPrio);
795             }
796             else
797             {
798                 printf("task(%2d)(%2d)\t\t", OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBCtxSwCtr);
799                 fprintf(Output_fp, "task(%2d)(%2d)\t\t", OSTCBHighRdy->OSTCBId, OSTCBHighRdy->OSTCBCtxSwCtr);
800             }
801             printf("%2d\t\t", ap_response_time); /*Response Time*/
802             fprintf(Output_fp, "%2d\t\t", ap_response_time);
803             printf("%2d\t\t", ap_preemptive_time); /*Preemptive Time*/
804             fprintf(Output_fp, "%2d\t\t", ap_preemptive_time);
805             printf("N/A\n"); /* Delay Time */
806             fprintf(Output_fp, "N/A\n");
807             fclose(Output_fp);
808         }
809         OSTCBCur->OSTCBCtxSwCtr++;
810     }

```

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