B10332024 四電資四 江承祐

PART 1 EDF Scheduler Implementation

1. Implementation Description:

The main objective of this project is to implement an earliest deadline first (EDF) scheduler. After observing the structure of this code, I decided to implement my code within OS_SchedNew().
OS_SchedNew is used within OS_IntExit, OS_Start, and OS_Sched. OS_SchedNew is responsible for deciding which task to perform next, so I decided to rewrite the code within this function.

```
void  OS_Sched (void)
{
#if OS_CRITICAL_METHOD == 3                                 /* Alloca
    OS_CPU_SR  cpu_sr = 0;
#endif

    OS_ENTER_CRITICAL();
    if (OSIntNesting == 0) {                                /* Schedu
        if (OSLockNesting == 0) {                           /* ... sc
            //ptcb = OSTCBList;

            OS_SchedNew();
            if (OSPrioHighRdy != OSPrioCur) {               /* No Ctx
                OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
#if OS_TASK_PROFILE_EN > 0
                OSTCBHighRdy->OSTCBCtxSwCtr++;              /* Inc. #
#endif
                 printf("\t%d\t Completed\tTask(%d)(%d)\tTask(%d
```

```
void  OSIntExit (void)
{
#if OS_CRITICAL_METHOD == 3                                 /*
    OS_CPU_SR  cpu_sr = 0;
#endif
    if (OSRunning == OS_TRUE) {
        OS_ENTER_CRITICAL();
        if (OSIntNesting > 0) {                             /*
            OSIntNesting--;
        }

        if (OSIntNesting == 0) {                            /*
            if (OSLockNesting == 0) {                       /*
                OS_SchedNew();
                if (OSPrioHighRdy != OSPrioCur) {           /*
                    OSTCBHighRdy  = OSTCBPrioTbl[OSPrioHighRd
#if OS_TASK_PROFILE_EN > 0
                    OSTCBHighRdy->OSTCBCtxSwCtr++;          /*
#endif
                    int preempting=0;
                    if(OSTCBCur->OSTCBPrio == OS_TASK_IDLE_PR
                        printf("\t%d\t Completed\tTask(%d)(%d
                    }
```

```
void  OSStart (void)
{
    if (OSRunning == OS_FALSE) {
        OS_SchedNew();                                      /* Find highest priority's task priority number   */
        OSPrioCur     = OSPrioHighRdy;
        OSTCBHighRdy  = OSTCBPrioTbl[OSPrioHighRdy];        /* Point to highest priority task ready to run    */
        OSTCBCur      = OSTCBHighRdy;
        OSStartHighRdy();                                   /* Execute target specific code to start task     */
    }


}
```

(1) Highest Priority Implementation

I noticed that OS_SchedNew gave a new value to OSPrioHighRdy each time it was used. I decided to give a new value to OSPrioHighRdy within this function. First we iterate through all of the TCBs within our OS. We check if the DEADLINE within each TCB, and save the TCB with the nearest deadline. We give OSPrioHighRdy the priority of the one we just saved.

```
static  void  OS_SchedNew (void)
{
#if OS_LOWEST_PRIO <= 63                         /* See if we support up to 64 tasks                   */
    INT8U   y;
    y             = OSUnMapTbl[OSRdyGrp];
    OSPrioHighRdy = (INT8U)((y << 3) + OSUnMapTbl[OSRdyTbl[y]]);
    INT8U i=99999;
    INT8U nearest=99999;
    OS_TCB     *ptcb;
    ptcb = OSTCBList;
    while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {
        if(ptcb->DEADLINE > 0 && ptcb->OSTCBDly==0 && OSTimeGet()!=ptcb->DEADLINE){
            if(ptcb->DEADLINE <= nearest){
                i=ptcb->OSTCBPrio;
                nearest=ptcb->DEADLINE;
            }

        }
        ptcb = ptcb->OSTCBNext;
    }
    if(OSPrioHighRdy != OS_TASK_IDLE_PRIO){
        OSPrioHighRdy = i;
    }
```

(2) When Two or More Tasks have the Same Deadline

```
INT8U i=99999;
INT8U nearest=99999;
OS_TCB     *ptcb;
ptcb = OSTCBList;
while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) {
    if(ptcb->DEADLINE > 0 && ptcb->OSTCBDly==0 && OSTimeGet()!=ptcb->DEADLINE){
        if(ptcb->DEADLINE <= nearest){
            i=ptcb->OSTCBPrio;
            nearest=ptcb->DEADLINE;
        }

    }
    ptcb = ptcb->OSTCBNext;
}
if(OSPrioHighRdy != OS_TASK_IDLE_PRIO){
    OSPrioHighRdy = i;
}
```

The while loop I set up starts from the last task to the first task I inputted (loops from Task3 ~ Task 1 in this figure).

```
OSTaskCreate(task1,NULL,(void *)&task1_stk[TASK_STACKSIZE-1],TASK1_PRIORITY, 1, 1, 4, 0);
OSTaskCreate(task2,NULL,(void *)&task2_stk[TASK_STACKSIZE-1],TASK2_PRIORITY, 2, 3, 5,0);
OSTaskCreate(task3,NULL,(void *)&task3_stk[TASK_STACKSIZE-1],TASK3_PRIORITY, 3, 6, 15,0);
```

With the line “ptcb->DEADLINE <= nearest”, I let the last task with the same deadline in the while loop be the next task to run (which is task 1 if all deadlines are the same).

In other words, I chose to use the first task I inputted as the prioritized task when two task deadlines were the same.

(3) Problems that occurred

A circumstance with the scheduler that I didn’t encounter before was if the next task to run was the same task that was just finished (ex: Complete From Task(1,5) ➔ Task(1,6)).

To fix this problem, I implemented an if statement to check if the delay time was 0.

```
void task1(void* pdata)
{
  INT8U prio=TASK1_PRIORITY;
  OS_TCB *ptcb= OSTCBPrioTbl[prio];
  ptcb->REMAINING_TIME    = ptcb->compute;

  while (1)
  {
    ptcb->TASK_ACTUAL_START_TIME = OSTimeGet();
    while( 0 < ptcb->REMAINING_TIME){

   }
    ptcb->RESPONSE_TIME = OSTimeGet() - ptcb->TASK_SHOULD_START_TIME;
    int todelay = ptcb->period - ptcb->RESPONSE_TIME;
    ptcb->TASK_SHOULD_START_TIME = ptcb->period + ptcb->TASK_SHOULD_START_TIME;
    ptcb->DEADLINE = ptcb->period + ptcb->DEADLINE;
    ptcb->REMAINING_TIME    = ptcb->compute;
    OSTimeDly(todelay);
    if(todelay==0){
       OS_Sched();
    }
    ptcb->RESPONSE_TIME   = 0;
  }
}
```

2. Simulation Results

(1) Task Set 1: Task1(1,4); Task2(3,6); Task3(6,24)

Current Time	Event	From Task ID	To Task ID	Response Time	Remain Time
1	Completed	Task(1)(0)	Task(2)(0)	1	
4	Completed	Task(2)(0)	Task(1)(1)	4	
5	Completed	Task(1)(1)	Task(3)(0)	1	
6	Preempted	Task(3)(0)	Task(2)(1)		5
8	Preempted	Task(2)(1)	Task(1)(2)		1
9	Completed	Task(1)(2)	Task(2)(1)	1	
10	Completed	Task(2)(1)	Task(3)(0)	4	
12	Preempted	Task(3)(0)	Task(1)(3)		3
13	Completed	Task(1)(3)	Task(2)(2)	1	
16	Completed	Task(2)(2)	Task(1)(4)	4	
17	Completed	Task(1)(4)	Task(3)(0)	1	
18	Preempted	Task(3)(0)	Task(2)(3)		2
20	Preempted	Task(2)(3)	Task(1)(5)		1
21	Completed	Task(1)(5)	Task(2)(3)	1	
22	Completed	Task(2)(3)	Task(3)(0)	4	
24	Completed	Task(3)(0)	Task(1)(6)	24	
25	Completed	Task(1)(6)	Task(2)(4)	1	
28	Completed	Task(2)(4)	Task(1)(7)	4	
29	Completed	Task(1)(7)	Task(3)(1)	1	
30	Preempted	Task(3)(1)	Task(2)(5)		5
32	Preempted	Task(2)(5)	Task(1)(8)		1
33	Completed	Task(1)(8)	Task(2)(5)	1	
34	Completed	Task(2)(5)	Task(3)(1)	4	
36	Preempted	Task(3)(1)	Task(1)(9)		3
37	Completed	Task(1)(9)	Task(2)(6)	1	
40	Completed	Task(2)(6)	Task(1)(10)	4	
41	Completed	Task(1)(10)	Task(3)(1)	1	
42	Preempted	Task(3)(1)	Task(2)(7)		2
44	Preempted	Task(2)(7)	Task(1)(11)		1
45	Completed	Task(1)(11)	Task(2)(7)	1	
46	Completed	Task(2)(7)	Task(3)(1)	4	
48	Completed	Task(3)(1)	Task(1)(12)	24	
49	Completed	Task(1)(12)	Task(2)(8)	1	
52	Completed	Task(2)(8)	Task(1)(13)	4	

In Task set 1, the EDF scheduler lets the tasks run smoothly in a pattern every 24 seconds. Task 1 allows just enough time for Task 3 to finish right before its deadline.

(2) Task Set 2: Task1(1,3); Task2(2,8); Task3(4,15); Task4(5,20)

Current Time	Event	From Task ID	To Task ID	Response Time	Remain Time
1	Completed	Task(1)(0)	Task(2)(0)	1	
3	Completed	Task(2)(0)	Task(1)(1)	3	
4	Completed	Task(1)(1)	Task(3)(0)	1	
6	Preempted	Task(3)(0)	Task(1)(2)		2
7	Completed	Task(1)(2)	Task(3)(0)	1	
9	Completed	Task(3)(0)	Task(1)(3)	9	
10	Completed	Task(1)(3)	Task(2)(1)	1	
12	Completed	Task(2)(1)	Task(1)(4)	4	
13	Completed	Task(1)(4)	Task(4)(0)	1	
15	Preempted	Task(4)(0)	Task(1)(5)		3
16	Completed	Task(1)(5)	Task(4)(0)	1	
19	Completed	Task(4)(0)	Task(1)(6)	19	
20	Completed	Task(1)(6)	Task(2)(2)	2	
21	Preempted	Task(2)(2)	Task(1)(7)		1
22	Completed	Task(1)(7)	Task(2)(2)	1	
23	Completed	Task(2)(2)	Task(3)(1)	7	
24	Preempted	Task(3)(1)	Task(1)(8)		3
25	Completed	Task(1)(8)	Task(3)(1)	1	
27	Preempted	Task(3)(1)	Task(1)(9)		1
28	Completed	Task(1)(9)	Task(3)(1)	1	
29	Completed	Task(3)(1)	Task(2)(3)	14	
31	Completed	Task(2)(3)	Task(1)(10)	7	
32	Completed	Task(1)(10)	Task(2)(4)	2	
33	Preempted	Task(2)(4)	Task(1)(11)		1
34	Completed	Task(1)(11)	Task(2)(4)	1	
35	Completed	Task(2)(4)	Task(4)(1)	3	
36	Preempted	Task(4)(1)	Task(1)(12)		4
37	Completed	Task(1)(12)	Task(4)(1)	1	
40	Miss Deadline	Task(4)(1)	----------		

In Task Set 2, Task4 misses its deadline by 1 tick. The main reason for this is probably because Task 1's period is too short, and doesn't allow enough time for other tasks to compute.

3. Schedulability Analysis

EDF scheduler is a scheduler that can let most tasks meet its deadline. However, it requires longer periods of time to let the other tasks meet their deadline.
If the period of time for one task is too short, then its deadline will often be the nearest. When this happens, the other tasks won't be computed long enough for them to meet their own deadline. An example of this is like Task Set 2. Task 1's period is only 3, and its deadline is always about to arrive. This eventually leads to Task 4 failing to meet its deadline.

4. Experience/Impression

This project let me understand the concept and shortcomings of the EDF scheduler more clearly. Simulating tasks and observing when the tasks would miss their deadline helped me recognize how to optimize the tasks to let them run smoothly within a EDF scheduler.
I feel like an EDF is able to accomplish meet task deadlines more efficiently than the rate-monotonic scheduler. The rate-monotonic gives priorities to task based on the

shortest period. However, the rate-monotonic isn't dynamic enough for longer period tasks to meet its deadline. EDF computes the most urgent task and lets tasks efficiently finish most of the time.

PART 2 Constant Utilization Server

1. Implementation Description:

The main objective of this part is to implement a constant utilization server (CUS). After observing the structure of this code, I decided to implement my code within OSTimeTick()
OSTimeTick is run for each tick. I decided to implement my CUS decision making here, so I can check if a new task has arrived whenever I want to.

(1) Information

First, I added the necessary information I needed for the aperiodic tasks.
The information includes:

a) The CUS server size of the aperiodic task
b) The current number of the aperiodic task running
c) An array of the arrival time for each aperiodic job
d) An array of the actual arrival time for each aperiodic job (arrival time could be affected by a previous job's deadline)
e) An array of the computation time for each aperiodic job
f) An array of the deadline for each aperiodic job

```
typedef struct serverinfo{
    double          server;
    int             currenttask;
    int             aperiodictasknum;
    int             arrivetime[20];
    int             actualarrivetime[20];
    int             compute[20];
    int             deadline[20];
} SERVER_INFO;

typedef struct os_tcb {
    OS_STK          *OSTCBStkPtr;           /* Pointer
    INT32U             task_id;
    INT32U             task_times;
    INT32U             period;
    INT32U             compute;
    INT32U             computing;
    INT32U             TASK_SHOULD_START_TIME;
    INT32U             TASK_ACTUAL_START_TIME;
    INT32U             REMAINING_TIME;
    INT32U             RESPONSE_TIME;
    INT32U             DEADLINE;
    SERVER_INFO        APERIODIC_INFO;
```

(2) Application Layer

I. Main function information input

```
int main(void)
{
  int arrive[10]={3,15,0,0,0};
  int deadline[10]={4,3,0,0,0};
  //printf(
  OSTaskCreate(task1,NULL,(void *)&task1_stk[TASK_STACKSIZE-1],TASK1_PRIORITY, 1, 2, 5, 0,0,NULL,NULL);
  OSTaskCreate(task2,NULL,(void *)&task2_stk[TASK_STACKSIZE-1],TASK2_PRIORITY, 2, 1, 10,0,0,NULL,NULL);
  OSTaskCreate(task3,NULL,(void *)&task3_stk[TASK_STACKSIZE-1],TASK3_PRIORITY, 3, 6, 20,0,0,NULL,NULL);
  //OSTaskCreate(task4,NULL,(void *)&task4_stk[TASK_STACKSIZE-1],TASK4_PRIORITY, 4, 5, 20, 0,0,NULL,NULL);
  OSTaskCreate(aptask,NULL,(void *)&aperiodic_stk[TASK_STACKSIZE-1],APERIODIC_PRIORITY, 4, 0, 0,0.2,APERIODIC_TASK_NUM,arrive,deadline);
```

a) The array “arrive” is used to input the arrival time of each job.
b) The array “deadline” is used to input the computation time of each job.
c) “0.2” is the server size of the aperiodic task.
d) “APERIODIC_TASK_NUM” is the number of aperiodic jobs that we expect to compute.

II. Running the Aperiodic task

```
void aptask(void* pdata)
{
  INT8U prio=APERIODIC_PRIORITY;
  OS_TCB *ptcb= OSTCBPrioTbl[prio];
  //ptcb->REMAINING_TIME     = ptcb->compute;
  while (1)
  {
    ptcb->TASK_ACTUAL_START_TIME = OSTimeGet();
    while( 0 < ptcb->REMAINING_TIME){

    }
    ptcb->RESPONSE_TIME = OSTimeGet() - ptcb->TASK_SHOULD_START_TIME;
    int todelay = ptcb->DEADLINE - OSTimeGet();
    OSTimeDly(todelay);
    ptcb->RESPONSE_TIME    = 0;
  }
}
```

The aperiodic task is slightly different than a normal task. The new remaining time is not immediately given and is given when a new aperiodic job arrives.

(3) OSTimeTick

I. Deciding the appropriate way to set up an aperiodic job

Because my EDF implementation kept on considering the aperiodic task’s deadline (which it shouldn’t), I decided to suspend the aperiodic task when there were no job’s available. When a new job arrives, I immediately resume the aperiodic task.

```
if(ptcb->OSTCBPrio==0){
    int i=0;
    if(ptcb->task_times>=ptcb->APERIODIC_INFO.aperiodictasknum){
        //ptcb->APERIODIC_INFO.aperiodictasknum=0;
        OSTaskSuspend(ptcb->OSTCBPrio);
    }
    for(i=0;i<ptcb->APERIODIC_INFO.aperiodictasknum;i++){
        if(ptcb->APERIODIC_INFO.arrivetime[i]==OSTimeGet()){
            if(i>0){
             if(ptcb->APERIODIC_INFO.arrivetime[i] < ptcb->APERIODIC_INFO.deadline[i-1]){
                printf("At Time %d, a new aperiodic %d job arrives but does nothing\n",OSTimeGet()/SCALE,i);
                ptcb->APERIODIC_INFO.actualarrivetime[i]=ptcb->APERIODIC_INFO.deadline[i-1];
                }
                else{
                 OSTaskResume(ptcb->OSTCBPrio);
                  ptcb->REMAINING_TIME = ptcb->APERIODIC_INFO.compute[i];
                  ptcb->TASK_SHOULD_START_TIME=ptcb->APERIODIC_INFO.arrivetime[i];
                  ptcb->DEADLINE = ptcb->APERIODIC_INFO.actualarrivetime[i] + ptcb->APERIODIC_INFO.compute[i]/ptcb->APERI
                  ptcb->APERIODIC_INFO.deadline[i] = ptcb->DEADLINE;
                  printf("At Time %d, a new aperiodic job arrives and its deadline is updated to %d\n",OSTimeGet()/SCALE,
                 }
            }
            else{
                OSTaskResume(ptcb->OSTCBPrio);
                  ptcb->REMAINING_TIME = ptcb->APERIODIC_INFO.compute[i];
                  ptcb->TASK_SHOULD_START_TIME=ptcb->APERIODIC_INFO.arrivetime[i];
                  ptcb->DEADLINE = ptcb->APERIODIC_INFO.actualarrivetime[i] + ptcb->APERIODIC_INFO.compute[i]/ptcb->APERI
                  ptcb->APERIODIC_INFO.deadline[i] = ptcb->DEADLINE;
                  printf("At Time %d, a new aperiodic job arrives and its deadline is updated to %d\n",OSTimeGet()/SCALE,
            }

        }
        if(ptcb->APERIODIC_INFO.arrivetime[i]!=ptcb->APERIODIC_INFO.actualarrivetime[i]){
         if(ptcb->APERIODIC_INFO.actualarrivetime[i]==OSTimeGet()){
```

```
}
if(ptcb->APERIODIC_INFO.arrivetime[i]!=ptcb->APERIODIC_INFO.actualarrivetime[i]){
 if(ptcb->APERIODIC_INFO.actualarrivetime[i]==OSTimeGet()){
    OSTaskResume(ptcb->OSTCBPrio);
    ptcb->REMAINING_TIME=ptcb->APERIODIC_INFO.compute[i];
    ptcb->TASK_SHOULD_START_TIME=ptcb->APERIODIC_INFO.arrivetime[i];
    ptcb->DEADLINE = ptcb->APERIODIC_INFO.actualarrivetime[i] + ptcb->APERIODIC_INFO.compute[i]/ptcb->APERIODIC_INF
    ptcb->APERIODIC_INFO.deadline[i]=ptcb->DEADLINE;
    printf("At Time %d, aperiodic job's deadline is updated to %d\n",OSTimeGet()/SCALE,ptcb->DEADLINE);
 }
}
```

II. Deciding when to run a job

There were three main circumstances I considered.

A. The first job arrives normally.
B. A new job arrives before the previous job's deadline.
C. A new job arrives after the previous job's deadline.

a) The first job arrives normally.

This implementation is pretty straight forward. If the current time equals the first aperiodic job's arrival time, then we resume the task. give the aperiodic task's TCB the information necessary for this job, and calculate the deadline of the current job.

```
if(ptcb->OSTCBPrio==0){
    int i=0;
    if(ptcb->task_times>=ptcb->APERIODIC_INFO.aperiodictasknum){
        //ptcb->APERIODIC_INFO.aperiodictasknum=0;
        OSTaskSuspend(ptcb->OSTCBPrio);
    }
    for(i=0;i<ptcb->APERIODIC_INFO.aperiodictasknum;i++){
        if(ptcb->APERIODIC_INFO.arrivetime[i]==OSTimeGet()){
            if(i>0){
             if(ptcb->APERIODIC_INFO.arrivetime[i] < ptcb->APERIODIC_INFO.deadline[i-1]){
                printf("At Time %d, a new aperiodic %d job arrives but does nothing\n",OSTimeGet()/SCALE,i);
                ptcb->APERIODIC_INFO.actualarrivetime[i]=ptcb->APERIODIC_INFO.deadline[i-1];
                }
                else{
                 OSTaskResume(ptcb->OSTCBPrio);
                  ptcb->REMAINING_TIME = ptcb->APERIODIC_INFO.compute[i];
                  ptcb->TASK_SHOULD_START_TIME=ptcb->APERIODIC_INFO.arrivetime[i];
                  ptcb->DEADLINE = ptcb->APERIODIC_INFO.actualarrivetime[i] + ptcb->APERIODIC_INFO.compute[i]/ptcb->APERI
                  ptcb->APERIODIC_INFO.deadline[i] = ptcb->DEADLINE;
                  printf("At Time %d, a new aperiodic job arrives and its deadline is updated to %d\n",OSTimeGet()/SCALE,
                 }
            }
            else{
                 OSTaskResume(ptcb->OSTCBPrio);
                  ptcb->REMAINING_TIME = ptcb->APERIODIC_INFO.compute[i];
                  ptcb->TASK_SHOULD_START_TIME=ptcb->APERIODIC_INFO.arrivetime[i];
                  ptcb->DEADLINE = ptcb->APERIODIC_INFO.actualarrivetime[i] + ptcb->APERIODIC_INFO.compute[i]/ptcb->APERI
                  ptcb->APERIODIC_INFO.deadline[i] = ptcb->DEADLINE;
                  printf("At Time %d, a new aperiodic job arrives and its deadline is updated to %d\n",OSTimeGet()/SCALE,
            }
        }
        if(ptcb->APERIODIC_INFO.arrivetime[i]!=ptcb->APERIODIC_INFO.actualarrivetime[i]){
         if(ptcb->APERIODIC_INFO.actualarrivetime[i]==OSTimeGet()){
```

b) A new job arrives after the previous job's deadline.

When an aperiodic job arrives after the first job, we start comparing its arrival time to the previous job's deadline. If the arrival time is smaller than the previous job's deadline, that means the aperiodic task is still delaying and we cannot run the new job immediately.

If the arrival time is after the previous job's deadline, then we can run the job normally. If so, we resume the aperiodic task, apply the job's information into the TCB and calculate the new deadline.

```
if(ptcb->OSTCBPrio==0){
    int i=0;
    if(ptcb->task_times>=ptcb->APERIODIC_INFO.aperiodictasknum){
         //ptcb->APERIODIC_INFO.aperiodictasknum=0;
         OSTaskSuspend(ptcb->OSTCBPrio);
    }
    for(i=0;i<ptcb->APERIODIC_INFO.aperiodictasknum;i++){
        if(ptcb->APERIODIC_INFO.arrivetime[i]==OSTimeGet()){
            if(i>0){
             if(ptcb->APERIODIC_INFO.arrivetime[i] < ptcb->APERIODIC_INFO.deadline[i-1]){
                printf("At Time %d, a new aperiodic %d job arrives but does nothing\n",OSTimeGet()/SCALE,i);
                ptcb->APERIODIC_INFO.actualarrivetime[i]=ptcb->APERIODIC_INFO.deadline[i-1];
                }
                else{
                OSTaskResume(ptcb->OSTCBPrio);
                 ptcb->REMAINING_TIME = ptcb->APERIODIC_INFO.compute[i];
                 ptcb->TASK_SHOULD_START_TIME=ptcb->APERIODIC_INFO.arrivetime[i];
                 ptcb->DEADLINE = ptcb->APERIODIC_INFO.actualarrivetime[i] + ptcb->APERIODIC_INFO.compute[i]/ptcb->APERI
                 ptcb->APERIODIC_INFO.deadline[i] = ptcb->DEADLINE;
                 printf("At Time %d, a new aperiodic job arrives and its deadline is updated to %d\n",OSTimeGet()/SCALE,
                }
            }
            else{
                 OSTaskResume(ptcb->OSTCBPrio);
                 ptcb->REMAINING_TIME = ptcb->APERIODIC_INFO.compute[i];
                 ptcb->TASK_SHOULD_START_TIME=ptcb->APERIODIC_INFO.arrivetime[i];
                 ptcb->DEADLINE = ptcb->APERIODIC_INFO.actualarrivetime[i] + ptcb->APERIODIC_INFO.compute[i]/ptcb->APERI
                 ptcb->APERIODIC_INFO.deadline[i] = ptcb->DEADLINE;
                 printf("At Time %d, a new aperiodic job arrives and its deadline is updated to %d\n",OSTimeGet()/SCALE,
            }

        }
        if(ptcb->APERIODIC_INFO.arrivetime[i]!=ptcb->APERIODIC_INFO.actualarrivetime[i]){
         if(ptcb->APERIODIC_INFO.actualarrivetime[i]==OSTimeGet()){
```

c) A new job arrives after the previous job's deadline.
When an aperiodic job arrives after the first job, we start comparing its arrival time to the previous job's deadline. If the arrival time is smaller than the previous job's deadline, that means the aperiodic task is still delaying and we cannot run the new job immediately.
If this happens, we have to update the arrival time of the new job. We change the "actual arrival time" of the new job into the deadline of the previous aperiodic job. When the OS eventually reaches the previous job's deadline, it simultaneously starts the new job's computation. Once this happens, we apply the job's information into the TCB and calculate the new deadline according to the new arrival time.

```
if(ptcb->OSTCBPrio==0){
    int i=0;
    if(ptcb->task_times>=ptcb->APERIODIC_INFO.aperiodictasknum){
         //ptcb->APERIODIC_INFO.aperiodictasknum=0;
         OSTaskSuspend(ptcb->OSTCBPrio);
    }
    for(i=0;i<ptcb->APERIODIC_INFO.aperiodictasknum;i++){
        if(ptcb->APERIODIC_INFO.arrivetime[i]==OSTimeGet()){
            if(i>0){
             if(ptcb->APERIODIC_INFO.arrivetime[i] < ptcb->APERIODIC_INFO.deadline[i-1]){
                printf("At Time %d, a new aperiodic %d job arrives but does nothing\n",OSTimeGet()/SCALE,i);
                ptcb->APERIODIC_INFO.actualarrivetime[i]=ptcb->APERIODIC_INFO.deadline[i-1];
                }
                else{
                 OSTaskResume(ptcb->OSTCBPrio);
                  ptcb->REMAINING_TIME = ptcb->APERIODIC_INFO.compute[i];
                  ptcb->TASK_SHOULD_START_TIME=ptcb->APERIODIC_INFO.arrivetime[i];
                  ptcb->DEADLINE = ptcb->APERIODIC_INFO.actualarrivetime[i] + ptcb->APERIODIC_INFO.compute[i]/ptcb->APERI
                  ptcb->APERIODIC_INFO.deadline[i] = ptcb->DEADLINE;
                  printf("At Time %d, a new aperiodic job arrives and its deadline is updated to %d\n",OSTimeGet()/SCALE,
                 }
            }
            else{
                 OSTaskResume(ptcb->OSTCBPrio);
                  ptcb->REMAINING_TIME = ptcb->APERIODIC_INFO.compute[i];
                  ptcb->TASK_SHOULD_START_TIME=ptcb->APERIODIC_INFO.arrivetime[i];
                  ptcb->DEADLINE = ptcb->APERIODIC_INFO.actualarrivetime[i] + ptcb->APERIODIC_INFO.compute[i]/ptcb->APERI
                  ptcb->APERIODIC_INFO.deadline[i] = ptcb->DEADLINE;
                  printf("At Time %d, a new aperiodic job arrives and its deadline is updated to %d\n",OSTimeGet()/SCALE,
            }

        }
        if(ptcb->APERIODIC_INFO.arrivetime[i]!=ptcb->APERIODIC_INFO.actualarrivetime[i]){
         if(ptcb->APERIODIC_INFO.actualarrivetime[i]==OSTimeGet()){
```

```
}
if(ptcb->APERIODIC_INFO.arrivetime[i]!=ptcb->APERIODIC_INFO.actualarrivetime[i]){
 if(ptcb->APERIODIC_INFO.actualarrivetime[i]==OSTimeGet()){
      OSTaskResume(ptcb->OSTCBPrio);
      ptcb->REMAINING_TIME=ptcb->APERIODIC_INFO.compute[i];
      ptcb->TASK_SHOULD_START_TIME=ptcb->APERIODIC_INFO.arrivetime[i];
      ptcb->DEADLINE = ptcb->APERIODIC_INFO.actualarrivetime[i] + ptcb->APERIODIC_INFO.compute[i]/ptcb->APERIODIC_INF
      ptcb->APERIODIC_INFO.deadline[i]=ptcb->DEADLINE;
      printf("At Time %d, aperiodic job's deadline is updated to %d\n",OSTimeGet()/SCALE,ptcb->DEADLINE);
 }
}
```

2. Simulation Results

(1) Task Set 1: Task1(1,3); Task2(4,15); Task3(3,20) CUS Server: 1/4

Aperiodic Job: {arrival time, computation}, Job1: (1,5), (22,4)

```
Current Time     Event          From Task ID    To Task ID     Response Time    Remain Time
At Time 1, a new aperiodic job 0 arrives and its deadline is updated to 21
        1        Completed      Task(1)(0)      Task(2)(0)     1
        3        Preempted      Task(2)(0)      Task(1)(1)                      2
        4        Completed      Task(1)(1)      Task(2)(0)     1
        6        Completed      Task(2)(0)      Task(1)(2)     6
        7        Completed      Task(1)(2)      Task(3)(0)     1
        9        Preempted      Task(3)(0)      Task(1)(3)                      1
        10       Completed      Task(1)(3)      Task(3)(0)     1
        11       Completed      Task(3)(0)      Task(4)(0)     11
        12       Preempted      Task(4)(0)      Task(1)(4)                      4
        13       Completed      Task(1)(4)      Task(4)(0)     1
        15       Preempted      Task(4)(0)      Task(1)(5)                      2
        16       Completed      Task(1)(5)      Task(4)(0)     1
        18       Completed      Task(4)(0)      Task(1)(6)     17
Aperiodic Task 0's deadline is 21
        19       Completed      Task(1)(6)      Task(2)(1)     1
        21       Preempted      Task(2)(1)      Task(1)(7)                      2
At Time 22, a new aperiodic job 1 arrives and its deadline is updated to 38
        22       Completed      Task(1)(7)      Task(2)(1)     1
        24       Completed      Task(2)(1)      Task(1)(8)     9
        25       Completed      Task(1)(8)      Task(4)(1)     1
        27       Preempted      Task(4)(1)      Task(1)(9)                      2
        28       Completed      Task(1)(9)      Task(4)(1)     1
        30       Completed      Task(4)(1)      Task(1)(10)    8
Aperiodic Task 1's deadline is 38
        31       Completed      Task(1)(10)     Task(3)(1)     1
        33       Preempted      Task(3)(1)      Task(1)(11)                     1
        34       Completed      Task(1)(11)     Task(3)(1)     1
        35       Completed      Task(3)(1)      Task(2)(2)     15
        36       Preempted      Task(2)(2)      Task(1)(12)                     3
        37       Completed      Task(1)(12)     Task(2)(2)     1
        39       Preempted      Task(2)(2)      Task(1)(13)                     1
        40       Completed      Task(1)(13)     Task(2)(2)     1
        41       Completed      Task(2)(2)      Task(3)(2)     11
        42       Preempted      Task(3)(2)      Task(1)(14)                     2
        43       Completed      Task(1)(14)     Task(3)(2)     1
        45       Completed      Task(3)(2)      Task(1)(15)    5
        46       Completed      Task(1)(15)     Task(2)(3)     1
        48       Preempted      Task(2)(3)      Task(1)(16)                     2
        49       Completed      Task(1)(16)     Task(2)(3)     1
```

(2) Task Set 1: Task1(2,5); Task2(1,10); Task3(6,20) CUS Server: 1/5
Aperiodic Job: {arrival time, computation}, Job1: (3,4), (15,3)

```
Current Time     Event          From Task ID    To Task ID     Response Time    Remain Time
        2        Completed      Task(1)(0)      Task(2)(0)      2
At Time 3, a new aperiodic job 0 arrives and its deadline is updated to 23
        3        Completed      Task(2)(0)      Task(3)(0)      3
        5        Preempted      Task(3)(0)      Task(1)(1)                      4
        7        Completed      Task(1)(1)      Task(3)(0)      2
        10       Preempted      Task(3)(0)      Task(1)(2)                      1
        12       Completed      Task(1)(2)      Task(2)(1)      2
        13       Completed      Task(2)(1)      Task(3)(0)      3
        14       Completed      Task(3)(0)      Task(4)(0)      14
At Time 15, a new aperiodic 1 job arrives but does nothing
        15       Preempted      Task(4)(0)      Task(1)(3)                      3
        17       Completed      Task(1)(3)      Task(4)(0)      2
        20       Completed      Task(4)(0)      Task(1)(4)      17
Aperiodic Task 0's deadline is 23
        22       Completed      Task(1)(4)      Task(2)(2)      2
At Time 23, aperiodic job's deadline is updated to 38
        23       Completed      Task(2)(2)      Task(4)(1)      3
        25       Preempted      Task(4)(1)      Task(1)(5)                      1
        27       Completed      Task(1)(5)      Task(4)(1)      2
        28       Completed      Task(4)(1)      Task(3)(1)      13
Aperiodic Task 1's deadline is 38
        30       Preempted      Task(3)(1)      Task(1)(6)                      4
        32       Completed      Task(1)(6)      Task(2)(3)      2
        33       Completed      Task(2)(3)      Task(3)(1)      3
        35       Preempted      Task(3)(1)      Task(1)(7)                      2
        37       Completed      Task(1)(7)      Task(3)(1)      2
        39       Completed      Task(3)(1)      Task(63)(0)     19
        40       Completed      Task(63)(0)     Task(1)(8)
        42       Completed      Task(1)(8)      Task(2)(4)      2
        43       Completed      Task(2)(4)      Task(3)(2)      3
        45       Preempted      Task(3)(2)      Task(1)(9)                      4
        47       Completed      Task(1)(9)      Task(3)(2)      2
        50       Preempted      Task(3)(2)      Task(1)(10)                     1
        52       Completed      Task(1)(10)     Task(2)(5)      2
```

.

3. Experience/Impression

This project let me understand how to successfully implement aperiodic tasks. Basically an aperiodic task is a task that checks if a new job about to arrive at all times. Using the algorithm to compute the deadline of an aperiodic job paired with the EDF algorithm, I feel like this scheduler is able to meet task deadlines efficiently. As long as, the original task set is able to achieve its deadlines, the aperiodic task merely uses up the idling time to finish its tasks, and doesn't affect the performance of the entire OS as much. This leads to many possibilities, as tasks can be run as aperiodic tasks, without having to be afraid of affecting the entire performance. Over the course of this project, I felt I have attained an even better understanding and perspective of the EDF scheduler and CUS.