國 立 台 灣 科 技 大 學

嵌 入 式 作 業 系 統 實 作

指 導 教 授：陳 雅 淑 教 授

嵌入式作業系統實作
Embedded OS Implementation

Project 01

班 級　：　電機碩一
學 生　：　江昱霖　　張晴雯
學 號　：　M10307431 M10407506

I. 系統環境設定(system.h)

- 關閉系統常駐任務 Statistic Task (不需執行此任務)

```
#define OS_DEBUG_EN 1
#define OS_SCHED_LOCK_EN 1
#define OS_TASK_STAT_EN 0
#define OS_TASK_STAT_STK_CHK_EN 1
#define OS_TICK_STEP_EN 1
```

- 修改為系統最低優先權為 63，優先權範圍為 0~63

```
#define ALT_MAX_FD 32
#define OS_MAX_TASKS 10
#define OS_LOWEST_PRIO 63
#define OS_FLAG_EN 1
```

II. RM (Rate Monotonic Scheduling)

- **任務參數創建及初始化設定**

1. 修改 TCB (ucos_ii.h) ，加入執行時間、週期、截止時間(RM 尚未使用到)、響應時間、開始時間以及任務的計數。

```
typedef struct os_tcb {
    OS_STK          *OSTCBStkPtr;           /* Pointer to current top of stack
    INT16U          ExecTime;               // Remaining Execution Time
    INT16U          Period;
    INT16U          Deadline;
    INT16U          Resp;                   // Response Time
    INT16U          Start;                  // Start Time
    INT16U          NoJob;                  // Job Number
```

2. 定義參數及創建任務參數資料結構 task，作為創建任務的傳入參數。

```
#define TASK1_PRIORITY      1
#define TASK2_PRIORITY      2
#define TASK3_PRIORITY      3

typedef struct {
    int exec;
    int period;
}task;
```

3. 定義任務函式

```
void task1(task* pdata)
{
  int start = 0;
  int end;
  int delay;
  OSTCBCur->NoJob = 0;
  OSTCBCur->Period = pdata->period;
  OSTCBCur->ExecTime = pdata->exec;
  OSTCBCur->Start = 0;
  OSTCBCur->Deadline = 0;
  while (1)
  {
    OSTCBCur->ExecTime = pdata->exec;        // ExecTime 補充
    OSTCBCur->Deadline += pdata->period;     // Deadline 更新
    while(OSTCBCur->ExecTime){}              // 模擬任務執行
    end = OSTimeGet();                       // 取得結束時間
    delay = pdata->period - (end-start);     // 計算 slack time 作為delay模擬任務使用
    OSTCBCur->Resp = end-start;              // 計算響應時間
    start += pdata->period;                  // 計算下次開始時間
    OSTCBCur->Start = start;
    OSTimeDly(OSTCBCur->Period-OSTCBCur->Resp); // delay至deadline 模擬周期性任務
  }
}
```

4. 創建任務

```
int main(void)
{
  task t1 = {1,3};
  task t2 = {3,7};

  OSInit();
  OSTaskCreateExt(task1,
                  &t1,      //傳入任務的執行時間及周期
                  (void *)&task1_stk[TASK_STACKSIZE-1],
                  TASK1_PRIORITY,
                  TASK1_PRIORITY,
                  task1_stk,
                  TASK_STACKSIZE,
                  NULL,
                  0);
```

- **uC/OS ii Kernel 修改**

在原本的系統下，我們使用週期作為優先權定義的依據，則其就是 RM 的排程。在每個 time tick 對目前的任務執行時間作遞減，模擬精確的任務執行。當 time tick 觸發時，為一中斷事件，會確認各個任務當前的狀況，並離開中斷時會呼叫 OSIntExit()，OSIntExit()內又會呼叫 OS_Sched()，檢查是否有更高優先權的任務與任務切換，因此在這兩個函式中來顯示任務的執行情況。

1. 模擬執行時間 (OS_TimeTick)

```
        }
        OSTCBCur->ExecTime--;    // 每經過一個單位時間則執行時間遞減
        //printf("%d , %d\n",OSTCBCur->OSTCBPrio,OSTCBCur->ExecTime);
    }
```

2. 離開中斷部分顯示任務資訊 (OSIntExit)

```
if (OSRunning == OS_TRUE) {
    OS_ENTER_CRITICAL();
    if (OSIntNesting > 0) {                          /* Prevent OSIntNesting from wrapping       */
        OSIntNesting--;
    }
    if (OSIntNesting == 0) {                         /* Reschedule only if all ISRs complete ... */
        if (OSLockNesting == 0) {                    /* ... and not locked.                      */
            OS_SchedNew();
            if ((OSPrioHighRdy != OSPrioCur)&&(OSTCBCur->ExecTime!=0)) {          /* No Ctx Sw if current
                OSTCBHighRdy  = OSTCBPrioTbl[OSPrioHighRdy];
                if(OSTCBCur->OSTCBId<62) {               // not idle task
                    if(OSTCBCur->ExecTime!=0) {
                        if(OSTimeGet() >= OSTCBCur->Deadline)   // over deadline
                            printf("\t%d\tMiss Deadline\tTask(%d)(%d)\tTask(%d)(%d)\t\n",OSTimeGet(),OSTCE
                        else
                            printf("\t%d\tPreempte\tTask(%d)(%d)\tTask(%d)(%d)\t\n",OSTimeGet(),OSTCBCur->
                    }
                    else {
                        printf("\t%d\tComplete\tTask(%d)(%d)\tTask(%d)(%d)\t%d\n",OSTimeGet(),OSTCBCur->OS
                        OSTCBCur->NoJob += 1;
                    }
                }
                else {
                    printf("\t%d\tComplete\tTask(%d)(%d)\tTask(%d)(%d)\t\n",OSTimeGet(),OSTCBCur->OSTCBId,
                    OSTCBCur->NoJob +=1;                 /* inc # of job for static task*/
                }
#if OS_TASK_PROFILE_EN > 0
                OSTCBHighRdy->OSTCBCtxSwCtr++;           /* Inc. # of context switches to this task  */
#endif
                OSCtxSwCtr++;                            /* Keep track of the number of ctx switches */
                OSIntCtxSw();                            /* Perform interrupt level ctx switch       */
            }
```

3. Scheduler 顯示任務資訊 (OS_Sched)，與 OSIntExit 增加部分大致相同。

```
if (OSIntNesting == 0) {                          /* Schedule only if all ISRs done and ...       */
    if (OSLockNesting == 0) {                     /* ... scheduler is not locked                  */
        OS_SchedNew();

        if (OSPrioHighRdy != OSPrioCur) {         /* No Ctx Sw if current task is highest rdy     */
            OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];
            if (OSTCBCur->OSTCBId<62){
                if(OSTCBCur->ExecTime!=0) {
                    if(OSTimeGet() >= OSTCBCur->Deadline)
                        printf("\t%d\tMiss Deadline\tTask(%d)(%d)\tTask(%d)(%d)\t\n",OSTimeGet(),OSTCBCur
                    else
                        printf("\t%d\tPreempte\tTask(%d)(%d)\tTask(%d)(%d)\t\n",OSTimeGet(),OSTCBCur->OST
                }
                else {
                    printf("\t%d\tComplete\tTask(%d)(%d)\tTask(%d)(%d)\t%d\n",OSTimeGet(),OSTCBCur->OSTCB
                    OSTCBCur->NoJob += 1;
                }
            }
            else {
                printf("\t%d\tComplete\tTask(%d)(%d)\tTask(%d)(%d)\t\n",OSTimeGet(),OSTCBCur->OSTCBId,OST
                OSTCBCur->NoJob +=1;              /* Plus # of job for static task*/
            }
#if OS_TASK_PROFILE_EN > 0
            OSTCBHighRdy->OSTCBCtxSwCtr++;        /* Inc. # of context switches to this task      */
#endif
            OSCtxSwCtr++;                         /* Increment context switch counter             */
            OS_TASK_SW();                         /* Perform a context switch*/
        }
```

- **問題與討論**

由於是利用 ExecTime 來模擬任務執行，雖然當下 ExecTime 以歸零視為任務完成，但其實還有設定 Delay 的部分才算整個任務完成，當 TimeTick=16 時，因為 Task2 還未完成 Delay 的設定而被 Task1 搶占，導致後面排程的出錯。因此必須在 OSIntExit 處理雖執行時間為零卻被搶佔導致錯誤的問題。

```
14      Complete        Task(1)(3)      Task(2)(1)      2
16      Complete        Task(2)(1)      Task(1)(4)      6
18      Complete        Task(1)(4)      Task(2)(2)      2
18      Complete        Task(2)(2)      Task(63)(0)     8
20      Complete        Task(63)(0)     Task(1)(5)
```

利用一判斷式來阻擋在執行時間為零且還沒完成 Delay 設定而被搶佔，則當執行時間為零時，不會進入作任務置換，可以順利執行設定 Delay，完成整個任務。

```
if ((OSPrioHighRdy != OSPrioCur)&&(OSTCBCur->ExecTime!=0)) {
    OSTCBHighRdy  = OSTCBPrioTbl[OSPrioHighRdy];
```

- **結果(Time Tick 0 - 42)**

1. Task set 1 模擬:

----------Task Set 1 { t1(1,3) , t2(3,7) }----------

Current Time	Event	From Task ID	To Task ID	Response Time
1	Complete	Task(1)(0)	Task(2)(0)	1
3	Preempte	Task(2)(0)	Task(1)(1)	
4	Complete	Task(1)(1)	Task(2)(0)	1
5	Complete	Task(2)(0)	Task(63)(0)	5
6	Complete	Task(63)(0)	Task(1)(2)	
7	Complete	Task(1)(2)	Task(2)(1)	1
9	Preempte	Task(2)(1)	Task(1)(3)	
10	Complete	Task(1)(3)	Task(2)(1)	1
11	Complete	Task(2)(1)	Task(63)(1)	4
12	Complete	Task(63)(1)	Task(1)(4)	
13	Complete	Task(1)(4)	Task(63)(2)	1
14	Complete	Task(63)(2)	Task(2)(2)	
15	Preempte	Task(2)(2)	Task(1)(5)	
16	Complete	Task(1)(5)	Task(2)(2)	1
18	Complete	Task(2)(2)	Task(1)(6)	4
19	Complete	Task(1)(6)	Task(63)(3)	1
21	Complete	Task(63)(3)	Task(1)(7)	
22	Complete	Task(1)(7)	Task(2)(3)	1
24	Preempte	Task(2)(3)	Task(1)(8)	
25	Complete	Task(1)(8)	Task(2)(3)	1
26	Complete	Task(2)(3)	Task(63)(4)	5
27	Complete	Task(63)(4)	Task(1)(9)	
28	Complete	Task(1)(9)	Task(2)(4)	1
30	Preempte	Task(2)(4)	Task(1)(10)	
31	Complete	Task(1)(10)	Task(2)(4)	1
32	Complete	Task(2)(4)	Task(63)(5)	4
33	Complete	Task(63)(5)	Task(1)(11)	
34	Complete	Task(1)(11)	Task(63)(6)	1
35	Complete	Task(63)(6)	Task(2)(5)	
36	Preempte	Task(2)(5)	Task(1)(12)	
37	Complete	Task(1)(12)	Task(2)(5)	1
39	Complete	Task(2)(5)	Task(1)(13)	4
40	Complete	Task(1)(13)	Task(63)(7)	1
42	Complete	Task(63)(7)	Task(1)(14)	

2. Task set 2 模擬:

```
----------------Task Set 2 { t1(1,3) , t2(2,8) , t3(5,12) }--------------------
------------------------------------------------------------------------------
Current Time     Event        From Task ID        To Task ID       Response Time
------------------------------------------------------------------------------
        1        Complete        Task(1)(0)       Task(2)(0)       1
        3        Complete        Task(2)(0)       Task(1)(1)       3
        4        Complete        Task(1)(1)       Task(3)(0)       1
        6        Preempte        Task(3)(0)       Task(1)(2)
        7        Complete        Task(1)(2)       Task(3)(0)       1
        8        Preempte        Task(3)(0)       Task(2)(1)
        9        Preempte        Task(2)(1)       Task(1)(3)
        10       Complete        Task(1)(3)       Task(2)(1)       1
        11       Complete        Task(2)(1)       Task(3)(0)       3
        12       Miss Deadline   Task(3)(0)       Task(1)(4)
```

- **Schedulability Analysis**

1. RMA

The highest priority tasks have the smallest periods. Priorities are assigned off-line.

$r_0 = \sum_{\forall i} C_i$ ， r_0= response time ， C_i= 每個 task 的執行時間

==> $r_n = \sum_{\forall i} C_i \left\lceil \frac{r_{n-1}}{P_i} \right\rceil$ ， P_i= 每個 task 的 period

Task Set 1 ：$\tau 1(1,3)$、$\tau 2(3,7)$

T1：

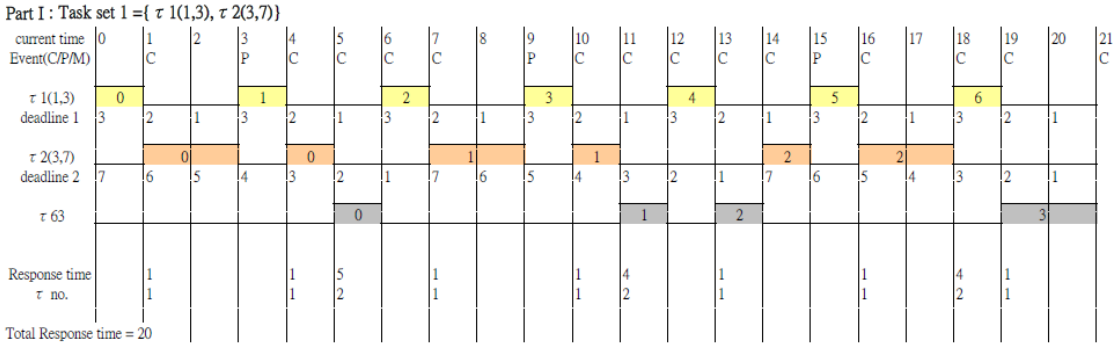
$r_0 = 1 \leqq 3$

T2：

$r_0 = 1+3 = 4 \leqq 7$

$r_1 = 1\times\left\lceil \frac{4}{3} \right\rceil + 3\times\left\lceil \frac{4}{7} \right\rceil = 5 \leqq 7$

$r_1 = 1\times\left\lceil \frac{5}{3} \right\rceil + 3\times\left\lceil \frac{5}{7} \right\rceil = 5 \leqq 7$ ， ok



Task Set 2 ：$\tau 1(1,3)$、$\tau 2(2,8)$、$\tau 3(5,12)$

T1：

$$r_0 = 1 \leqq 3 \text{ ， ok}$$

T2：

$$r_0 = 1+2 = 3 \leqq 8 \text{， ok}$$

$$r_1 = 1\times\left\lceil\frac{3}{3}\right\rceil + 2\times\left\lceil\frac{3}{8}\right\rceil = 3 \leqq 8 \text{ ， ok}$$

T3：

$$r_0 = 1+2+5 = 8 \leqq 12$$

$$r_1 = 1\times\left\lceil\frac{8}{3}\right\rceil + 2\times\left\lceil\frac{8}{8}\right\rceil + 5\times\left\lceil\frac{8}{12}\right\rceil = 10 \leqq 12$$

$$r_2 = 1\times\left\lceil\frac{10}{3}\right\rceil + 2\times\left\lceil\frac{10}{8}\right\rceil + 5\times\left\lceil\frac{10}{12}\right\rceil = 15 > 12$$

==> the task set failed

而 Utilization factor U = $\sum_{i=1}^{n}\frac{C_i}{P_i} \leqq U(n) = n(2^{1/n}-1)$

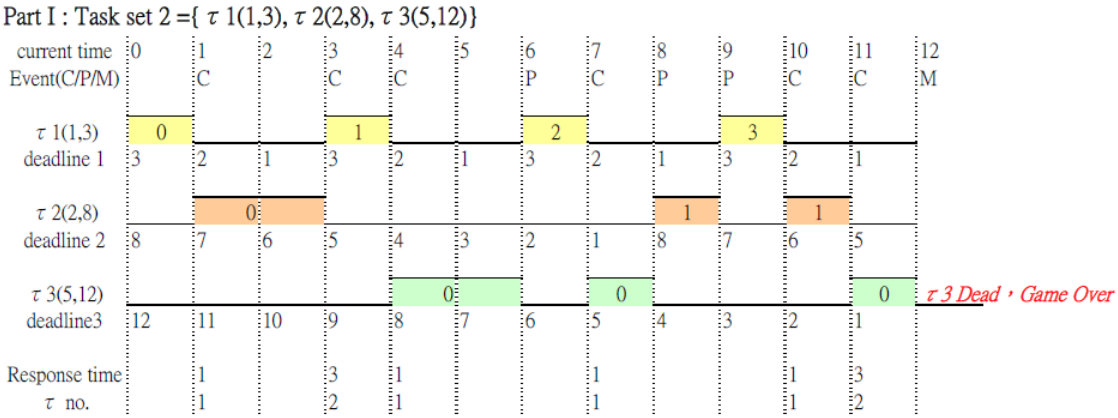
Task Set 1 ：$\tau 1(1,3)$、$\tau 2(3,7)$

$$U = \frac{1}{3}+\frac{3}{7} = 0.76 \leqq U(2) = 0.828$$

Task Set 2 ：$\tau 1(1,3)$、$\tau 2(2,8)$、$\tau 3(5,12)$

$$U = \frac{1}{3}+\frac{2}{8}+\frac{5}{12} = 1 > U(3) = 0.779$$

==> 不一定能排入，由前 RMA 分析結果已知不能被排程。



III. EDF (Earliest Deadline First Scheduling)

- **任務參數創建及初始化設定**

1. EDF 係使用截止時間作為動態優先權的設定依據，因此在 RM 的基礎上多增加了截止時間於 TCB 中(ucos_ii.h)。

```
typedef struct os_tcb {
    OS_STK          *OSTCBStkPtr;           /* Pointer to current top of stack
    INT16U          ExecTime;               // Remaining Execution Time
    INT16U          Period;
    INT16U          Deadline;
    INT16U          Resp;                   // Response Time
```

2. 定義任務函式

```
void task1(task* pdata)
{
  int start = 0;
  int end;
  int delay;
  OSTCBCur->NoJob = 0;
  OSTCBCur->Period = pdata->period;
  OSTCBCur->Exec = pdata->exec;
  OSTCBCur->ExecTime = pdata->exec;
  OSTCBCur->Start = 0;
  OSTCBCur->Deadline = pdata->period;
  while (1)
  {

    while(OSTCBCur->ExecTime){}

    end = OSTimeGet();
    if(end != OSTCBCur->Deadline-OSTCBCur->Period) {    // if finish time = deadline
        delay = pdata->period - (end-start);            // let it delay 1 timetick to avoid running the while loop
    }
    else
        delay =1;
    OSTCBCur->Resp = end-start;
    start += pdata->period;
    OSTimeDly(delay);

  }
}
```

- **uC/OS ii Kernel 修改**

EDF 以截止時間定義優先權，是一動態優先權的排程方式。因此在任務有異動(加入/完成)時就必須判斷目前哪個任務的截止時間最早，其優先權為最高，因此我們在 Time Tick 中判斷當 ExecTime 歸零時代表該任務即將完成，並更新其截止時間，再讓 Scheduler 去判斷最高優先權。在 Scheduler 部分摒棄了原查表的方式，利用一迴圈尋找存在的任務中截止時間最早的任務為優先權最高的任務，使其能夠馬上被執行。

1. 任務截止時間更新 (OS_TimeTick)

```
//printf("\n");
OSTCBCur->ExecTime--;
if(OSTCBCur->ExecTime==0) {
    OSTCBCur->Deadline += OSTCBCur->Period;
```

2. 修改 Scheduler 尋找最高優先權的部分，改以最早截止時間為優先 (OS_SchedNew)

```
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 UrgentPrio = OS_LOWEST_PRIO;          // the most urgent priority */
    INT16U UrgentDeadline = 65535u;
    INT8U i=0;
    while(i < OS_LOWEST_PRIO) {                                                 // search all priority table
        if(OSTCBPrioTbl[i]!=(OS_TCB*)0 && OSTCBPrioTbl[i]!=OS_TCB_RESERVED) {  // not an empty TCB (不為空||已存在)
            if(OSTCBPrioTbl[i]->OSTCBDly==0) {                                  // ready task
                // search shortest deadline (截止時間小||當截止時間相同以id小的優先)
                if((OSTCBPrioTbl[i]->Deadline < UrgentDeadline)||(OSTCBPrioTbl[i]->Deadline == UrgentDeadline && Ur
                   UrgentDeadline = OSTCBPrioTbl[i]->Deadline;
                   UrgentPrio = OSTCBPrioTbl[i]->OSTCBPrio;
                }
            }
        }
        i++;
    }
    OSPrioHighRdy = UrgentPrio;
```

- **問題與討論**

在 total utilization =1 的情況下，在 hyper period 的時間點，每個任務的 deadline 均相同，當下一定是最低優先權的任務完成，但由於完成的時間剛好壓在截止時間上，該任務的 Delay = 0，會直接重新執行 while 迴圈，導致一些參數的重複計算設定而出錯，後續的排程也為之大亂。

因此我們得知根本原因是會重複執行 while，所以當在這種狀況下強制使其 Delay 一個 TimeTick，使其在該時間點結束能夠保持正確的參數設定。因為這種狀況一定發生在 hyper period 上，而執行的任務一定是最低優先權的任務，接著執行的一定是最高優先權的任務，因此強制使最低優先權任務延遲一個 TimeTick 對整個系統排程是沒有影響的。

```
while(OSTCBCur->ExecTime){}

end = OSTimeGet();
if(end != OSTCBCur->Deadline-OSTCBCur->Period) {    // if finish time = deadline
    delay = pdata->period - (end-start);            // let it delay 1 timetick to avoid running the while loop
}
else
    delay =1;
OSTCBCur->Resp = end-start;
start += pdata->period;
OSTimeDly(delay);
```

由於是搜尋每個任務找尋最早截止時間的任務，因此其時間複雜度為 O(n)，而原本 kernel 的時間複雜度為 O(1)，但其是以空間換取時間，利用建表的方式來快速找尋最高優先權。

- **結果**

1. Task set 1 :

   - 模擬

```
================= EDF Task Set 1 { t1(1,3) , t2(3,7) }=========================
--------------------------------------------------------------------------------
Current Time     Event       From Task ID        To Task ID      Response Time
--------------------------------------------------------------------------------
        1        Complete        Task(1)(0)      Task(2)(0)      1
        3        Preempte        Task(2)(0)      Task(1)(1)
        4        Complete        Task(1)(1)      Task(2)(0)      1
        5        Complete        Task(2)(0)      Task(63)(0)     5
        6        Complete        Task(63)(0)     Task(1)(2)
        7        Complete        Task(1)(2)      Task(2)(1)      1
        9        Preempte        Task(2)(1)      Task(1)(3)
        10       Complete        Task(1)(3)      Task(2)(1)      1
        11       Complete        Task(2)(1)      Task(63)(1)     4
        12       Complete        Task(63)(1)     Task(1)(4)
        13       Complete        Task(1)(4)      Task(63)(2)     1
        14       Complete        Task(63)(2)     Task(2)(2)
        15       Preempte        Task(2)(2)      Task(1)(5)
        16       Complete        Task(1)(5)      Task(2)(2)      1
        18       Complete        Task(2)(2)      Task(1)(6)      4
        19       Complete        Task(1)(6)      Task(63)(3)     1
        21       Complete        Task(63)(3)     Task(1)(7)

        21       Complete        Task(63)(3)     Task(1)(7)
        22       Complete        Task(1)(7)      Task(2)(3)      1
        24       Preempte        Task(2)(3)      Task(1)(8)
        25       Complete        Task(1)(8)      Task(2)(3)      1
        26       Complete        Task(2)(3)      Task(63)(4)     5
        27       Complete        Task(63)(4)     Task(1)(9)
        28       Complete        Task(1)(9)      Task(2)(4)      1
        30       Preempte        Task(2)(4)      Task(1)(10)
        31       Complete        Task(1)(10)     Task(2)(4)      1
        32       Complete        Task(2)(4)      Task(63)(5)     4
        33       Complete        Task(63)(5)     Task(1)(11)
        34       Complete        Task(1)(11)     Task(63)(6)     1
        35       Complete        Task(63)(6)     Task(2)(5)
        36       Preempte        Task(2)(5)      Task(1)(12)
        37       Complete        Task(1)(12)     Task(2)(5)      1
        39       Complete        Task(2)(5)      Task(1)(13)     4
        40       Complete        Task(1)(13)     Task(63)(7)     1
        42       Complete        Task(63)(7)     Task(1)(14)
```

➢ 硬體上運行

Problems | Console | Properties

<terminated> hello_ucosii_0 Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (2015/11/6 下午 8:34)

```
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

================= EDF Task Set 1 ( t1(1,3) , t2(3,7) )=========================
--------------------------------------------------------------------------------
Current Time      Event        From Task ID         To Task ID       Response Time
--------------------------------------------------------------------------------
          1       Complete        Task(1)(0)       Task(2)(0)       1
          3       Preempte        Task(2)(0)       Task(1)(1)
          4       Complete        Task(1)(1)       Task(2)(0)       1
          5       Complete        Task(2)(0)       Task(63)(0)      5
          6       Complete        Task(63)(0)      Task(1)(2)
          7       Complete        Task(1)(2)       Task(2)(1)       1
          9       Preempte        Task(2)(1)       Task(1)(3)
          10      Complete        Task(1)(3)       Task(2)(1)       1
          11      Complete        Task(2)(1)       Task(63)(1)      4
          12      Complete        Task(63)(1)      Task(1)(4)
          13      Complete        Task(1)(4)       Task(63)(2)      1
          14      Complete        Task(63)(2)      Task(2)(2)
          15      Preempte        Task(2)(2)       Task(1)(5)
          16      Complete        Task(1)(5)       Task(2)(2)       1
          18      Complete        Task(2)(2)       Task(1)(6)       4
          19      Complete        Task(1)(6)       Task(63)(3)      1
          21      Complete        Task(63)(3)      Task(1)(7)
          22      Complete        Task(1)(7)       Task(2)(3)       1
          24      Preempte        Task(2)(3)       Task(1)(8)
          25      Complete        Task(1)(8)       Task(2)(3)       1
          26      Complete        Task(2)(3)       Task(63)(4)      5
          27      Complete        Task(63)(4)      Task(1)(9)
          28      Complete        Task(1)(9)       Task(2)(4)       1
          30      Preempte        Task(2)(4)       Task(1)(10)
          31      Complete        Task(1)(10)      Task(2)(4)       1
          32      Complete        Task(2)(4)       Task(63)(5)      4
          33      Complete        Task(63)(5)      Task(1)(11)
          34      Complete        Task(1)(11)      Task(63)(6)      1
          35      Complete        Task(63)(6)      Task(2)(5)
          36      Preempte        Task(2)(5)       Task(1)(12)
          37      Complete        Task(1)(12)      Task(2)(5)       1
          39      Complete        Task(2)(5)       Task(1)(13)      4
          40      Complete        Task(1)(13)      Task(63)(7)      1
          42      Complete        Task(63)(7)      Task(1)(14)
```

2. Task set 2 :

- 模擬

========== EDF Task Set 2 { t1(1,3) , t2(2,8) , t3(5,12) }==========

Current Time	Event	From Task ID	To Task ID	Response Time
1	Complete	Task(1)(0)	Task(2)(0)	1
3	Complete	Task(2)(0)	Task(1)(1)	3
4	Complete	Task(1)(1)	Task(3)(0)	1
6	Preempte	Task(3)(0)	Task(1)(2)	
7	Complete	Task(1)(2)	Task(3)(0)	1
9	Preempte	Task(3)(0)	Task(1)(3)	
10	Complete	Task(1)(3)	Task(3)(0)	1
11	Complete	Task(3)(0)	Task(2)(1)	11
12	Preempte	Task(2)(1)	Task(1)(4)	
13	Complete	Task(1)(4)	Task(2)(1)	1
14	Complete	Task(2)(1)	Task(3)(1)	6
15	Preempte	Task(3)(1)	Task(1)(5)	
16	Complete	Task(1)(5)	Task(2)(2)	1
18	Complete	Task(2)(2)	Task(1)(6)	2
19	Complete	Task(1)(6)	Task(3)(1)	1
21	Preempte	Task(3)(1)	Task(1)(7)	
22	Complete	Task(1)(7)	Task(3)(1)	1
24	Complete	Task(3)(1)	Task(1)(8)	12
25	Complete	Task(1)(8)	Task(2)(3)	1
27	Complete	Task(2)(3)	Task(1)(9)	3
28	Complete	Task(1)(9)	Task(3)(2)	1
30	Preempte	Task(3)(2)	Task(1)(10)	
31	Complete	Task(1)(10)	Task(3)(2)	1
33	Preempte	Task(3)(2)	Task(1)(11)	
34	Complete	Task(1)(11)	Task(3)(2)	1
35	Complete	Task(3)(2)	Task(2)(4)	11
36	Preempte	Task(2)(4)	Task(1)(12)	
37	Complete	Task(1)(12)	Task(2)(4)	1
38	Complete	Task(2)(4)	Task(3)(3)	6
39	Preempte	Task(3)(3)	Task(1)(13)	
40	Complete	Task(1)(13)	Task(2)(5)	1
42	Complete	Task(2)(5)	Task(1)(14)	2
43	Complete	Task(1)(14)	Task(3)(3)	1
45	Preempte	Task(3)(3)	Task(1)(15)	
46	Complete	Task(1)(15)	Task(3)(3)	1
48	Complete	Task(3)(3)	Task(1)(16)	12

➢ 硬體上運行

Problems | Console | Properties

<terminated> hello_ucosii_0 Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (2015/11/6 下午 8:21)

```
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

============ EDF Task Set 2 { t1(1,3) , t2(2,8) , t3(5,12) }==================
------------------------------------------------------------------------------
Current Time      Event        From Task ID         To Task ID       Response Time
------------------------------------------------------------------------------
        1       Complete        Task(1)(0)       Task(2)(0)       1
        3       Complete        Task(2)(0)       Task(1)(1)       3
        4       Complete        Task(1)(1)       Task(3)(0)       1
        6       Preempte        Task(3)(0)       Task(1)(2)
        7       Complete        Task(1)(2)       Task(3)(0)       1
        9       Preempte        Task(3)(0)       Task(1)(3)
        10      Complete        Task(1)(3)       Task(3)(0)       1
        11      Complete        Task(3)(0)       Task(2)(1)       11
        12      Preempte        Task(2)(1)       Task(1)(4)
        13      Complete        Task(1)(4)       Task(2)(1)       1
        14      Complete        Task(2)(1)       Task(3)(1)       6
        15      Preempte        Task(3)(1)       Task(1)(5)
        16      Complete        Task(1)(5)       Task(2)(2)       1
        18      Complete        Task(2)(2)       Task(1)(6)       2
        19      Complete        Task(1)(6)       Task(3)(1)       1
        21      Preempte        Task(3)(1)       Task(1)(7)
        22      Complete        Task(1)(7)       Task(3)(1)       1
        24      Complete        Task(3)(1)       Task(1)(8)       12
        25      Complete        Task(1)(8)       Task(2)(3)       1
        27      Complete        Task(2)(3)       Task(1)(9)       3
        28      Complete        Task(1)(9)       Task(3)(2)       1
        30      Preempte        Task(3)(2)       Task(1)(10)
        31      Complete        Task(1)(10)      Task(3)(2)       1
        33      Preempte        Task(3)(2)       Task(1)(11)
        34      Complete        Task(1)(11)      Task(3)(2)       1
        35      Complete        Task(3)(2)       Task(2)(4)       11
        36      Preempte        Task(2)(4)       Task(1)(12)
        37      Complete        Task(1)(12)      Task(2)(4)       1
        38      Complete        Task(2)(4)       Task(3)(3)       6
        39      Preempte        Task(3)(3)       Task(1)(13)
        40      Complete        Task(1)(13)      Task(2)(5)       1
        42      Complete        Task(2)(5)       Task(1)(14)      2
        43      Complete        Task(1)(14)      Task(3)(3)       1
        45      Preempte        Task(3)(3)       Task(1)(15)
        46      Complete        Task(1)(15)      Task(3)(3)       1
        48      Complete        Task(3)(3)       Task(1)(16)      12
```

- **Schedulability Analysis**

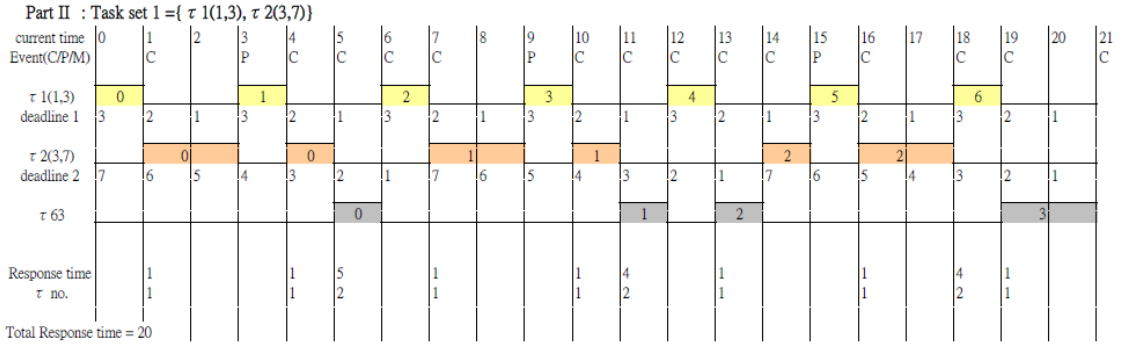
  A job having an earlier deadline is assigned to a higher priority. A set of tasks is schedulable by EDF if and only if its total CPU utilization is no more than 1.

  若每個 task 工作時間都 < T 而加起來 U>1 則不能排!

  即 $U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leqq 1$

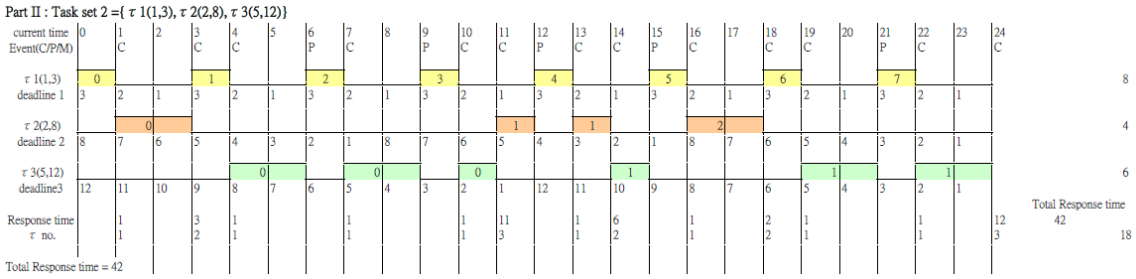
  Task Set 1 ：$\tau 1(1,3)$、$\tau 2(3,7)$

  $U = \frac{1}{3} + \frac{3}{7} = 0.76 \leqq 1$ ==> 可排程



  Task Set 2 ：$\tau 1(1,3)$、$\tau 2(2,8)$、$\tau 3(5,12)$

  $U = \frac{1}{3} + \frac{2}{8} + \frac{5}{12} = 1 \leqq 1$ ==> 可排程



IV. LSTR (Least Slack Time Rate Scheduling)

- **構想:**

事有緩急輕重，現實生活中對於事情的安排我們可能會考慮最急迫及最重要的為優先，因此我們構想出能夠兼具事情急迫性與重要性的演算法，初步稱之為最大壓力排程法，剩餘工作量與剩餘可執行時間的比例為之壓力:

$$\text{Stress} = \frac{Remaining\ Time}{Deadline - Current\ Time}$$

以壓力大的任務為優先,在每個時間點都會檢查所有任務的壓力，動態調整優先權，與EDF最大的不同為EDF只考慮截止時間，不能確認每個當下刻不容緩的任務，一旦EDF執行高優先權時有一段時間其他任務不被考慮，因此我們的方式能夠確認每個當下最急迫的任務進行排程，其utilization bound也與EDF相同。

後來開始著手尋找相關演算法確認是否已被提出，結果找到了一篇2010年提出與我們構想一模一樣的演算法叫做LSTR(Least Slack Time Rate Scheduling)，其是在EDF及LST的基礎下發展而來，運行在多核的環境下，因每個時間點都去檢查任務狀況，雖然多花了點時間去做檢查任務狀態，但能夠減少其他處理器閒置的情形。

- **任務參數創建及初始化設定**

1. 此方法係使用壓力作為動態優先權的設定依據，因此在EDF的基礎上多增加了壓力於TCB中(ucos_ii.h)。

```
INT16U          Start;
INT16U          NoJob;
INT16U          Stress;                 // Stress = Remaining Exec / Remaining Time
```

2. 我們將所有任務參數初始化都放進了TCB初始化的函式中(TCBinit),由於使用整數計算，因此在**計算Stress時乘上100來修正小數的情形**。

```
#if OS_TASK_CREATE_EXT_EN > 0
        ptcb->OSTCBExtPtr        = pext;                /* Store pointer to TCB extension          */
        ptcb->OSTCBStkSize       = stk_size;            /* Store stack size                        */
        ptcb->OSTCBStkBottom     = pbos;                /* Store pointer to bottom of stack        */
        ptcb->OSTCBOpt           = opt;                 /* Store task options                      */
        ptcb->OSTCBId            = id;                  /* Store task ID                           */
        ptcb->Start              = 0;
        ptcb->NoJob              = 0;
        ptcb->Exec               = t->exec;
        ptcb->ExecTime           = t->exec;
        ptcb->Period             = t->period;
        ptcb->Deadline           = t->period;
        ptcb->Stress             = (100*t->exec) / t->period;
```

- uC/OS ii Kernel 修改

1. Scheduler與EDF類似，只是改以尋找壓力最大的任務進行排程。

```
INT8U UrgentPrio = OS_LOWEST_PRIO;          // the most urgent priority */
INT16U UrgentStr = 0;
INT8U i=0;
while(i < OS_LOWEST_PRIO) {                                    // search all priority table
    if(OSTCBPrioTbl[i]!=(OS_TCB*)0 && OSTCBPrioTbl[i]!=OS_TCB_RESERVED) {  // not an empty TCB
        if(OSTCBPrioTbl[i]->OSTCBDly==0) {                                 // ready task
            if((OSTCBPrioTbl[i]->Stress > UrgentStr)||(OSTCBPrioTbl[i]->Stress == UrgentStr && t
                UrgentStr = OSTCBPrioTbl[i]->Stress;
                UrgentPrio = OSTCBPrioTbl[i]->OSTCBPrio;
            }
        }
    }
    i++;
}
OSPrioHighRdy = UrgentPrio;
```

2. 每個任務的壓力在每個TimeTick均會更新 (OS_TimeTick)

```
/*  Stress = Remaining Exec / Remaining Time  (100 -> scale)  */
ptcb->Stress = (100*ptcb->ExecTime) / (ptcb->Deadline - OSTimeGet()+1);

ptcb = ptcb->OSTCBNext;                    /* Point at next TCB in TCB list              */
OS_EXIT_CRITICAL();
```

- 問題與討論

我們的構想已存在於2010 年JCUS論文中[1]，此論文是探討在多核的環境下，LSTR相較EDF能夠大幅減少其他核心閒置的情形，且可排程性與EDF相同，能夠在total utilization小於1的情形下排程，具有與EDF相似的特性。

其缺點就是必須在每個時間點多花一些時間計算每個任務的執行速率，但也因為這樣能夠及時知道哪個任務是刻不容緩的，從實驗數據中能夠看出當task越多，其overhead越大，為O(n)線性成長的趨勢。

Num. of tasks	3 tasks		5 tasks		7 tasks		9 tasks	
Method	EDF	LSTR	EDF	LSTR	EDF	LSTR	EDF	LSTR
Avg. ET	186	281	518	762	1,074	1,436	1,970	2,419
Worst ET	761	910	1,685	2,321	2,886	3,160	6,928	7,571
Difference (LSTR-EDF)	95 (0.0095 ms)		254 (0.0254 ms)		363 (0.0363 ms)		449 (0.0449 ms)	

Table 12: Scheduling process using EDF algorithm (ET: Elapsed Time)

- **結果**

1. Task set 1 模擬:

```
================ MSF Task Set 1 { t1(1,3) , t2(3,7) }=======================
-----------------------------------------------------------------------------
Current Time     Event       From Task ID        To Task ID      Response Time
-----------------------------------------------------------------------------
          1      Preempte      Task(2)(0)       Task(1)(0)
          2      Complete      Task(1)(0)       Task(2)(0)      2
          3      Preempte      Task(2)(0)       Task(1)(1)
          4      Complete      Task(1)(1)       Task(2)(0)      1
          5      Complete      Task(2)(0)       Task(63)(0)     5
          6      Complete      Task(63)(0)      Task(1)(2)
          7      Complete      Task(1)(2)       Task(2)(1)      1
          9      Preempte      Task(2)(1)       Task(1)(3)
          10     Complete      Task(1)(3)       Task(2)(1)      1
          11     Complete      Task(2)(1)       Task(63)(1)     4
          12     Complete      Task(63)(1)      Task(1)(4)
          13     Complete      Task(1)(4)       Task(63)(2)     1
          14     Complete      Task(63)(2)      Task(2)(2)
          16     Preempte      Task(2)(2)       Task(1)(5)
          17     Complete      Task(1)(5)       Task(2)(2)      2
          18     Complete      Task(2)(2)       Task(1)(6)      4
          19     Complete      Task(1)(6)       Task(63)(3)     1
          21     Complete      Task(63)(3)      Task(2)(3)
          22     Preempte      Task(2)(3)       Task(1)(7)
          23     Complete      Task(1)(7)       Task(2)(3)      2
          24     Preempte      Task(2)(3)       Task(1)(8)
          25     Complete      Task(1)(8)       Task(2)(3)      1
          26     Complete      Task(2)(3)       Task(63)(4)     5
          27     Complete      Task(63)(4)      Task(1)(9)
          28     Complete      Task(1)(9)       Task(2)(4)      1
          30     Preempte      Task(2)(4)       Task(1)(10)
          31     Complete      Task(1)(10)      Task(2)(4)      1
          32     Complete      Task(2)(4)       Task(63)(5)     4
          33     Complete      Task(63)(5)      Task(1)(11)
          34     Complete      Task(1)(11)      Task(63)(6)     1
          35     Complete      Task(63)(6)      Task(2)(5)
          37     Preempte      Task(2)(5)       Task(1)(12)
          38     Complete      Task(1)(12)      Task(2)(5)      2
          39     Complete      Task(2)(5)       Task(1)(13)     4
          40     Complete      Task(1)(13)      Task(63)(7)     1
          42     Complete      Task(63)(7)      Task(2)(6)
```

2. Task set 2 模擬:

========== MSF Task Set 2 { t1(1,3) , t2(2,8) , t3(5,12) }==========

Current Time	Event	From Task ID	To Task ID	Response Time
1	Preempte	Task(3)(0)	Task(1)(0)	
2	Complete	Task(1)(0)	Task(3)(0)	2
3	Preempte	Task(3)(0)	Task(2)(0)	
4	Preempte	Task(2)(0)	Task(1)(1)	
5	Complete	Task(1)(1)	Task(3)(0)	2
6	Preempte	Task(3)(0)	Task(2)(0)	
7	Complete	Task(2)(0)	Task(1)(2)	7
8	Complete	Task(1)(2)	Task(3)(0)	2
9	Preempte	Task(3)(0)	Task(1)(3)	
10	Complete	Task(1)(3)	Task(3)(0)	1
11	Complete	Task(3)(0)	Task(2)(1)	11
12	Preempte	Task(2)(1)	Task(3)(1)	
13	Preempte	Task(3)(1)	Task(1)(4)	
14	Complete	Task(1)(4)	Task(3)(1)	2
15	Preempte	Task(3)(1)	Task(2)(1)	
16	Complete	Task(2)(1)	Task(1)(5)	8
17	Complete	Task(1)(5)	Task(3)(1)	2
18	Preempte	Task(3)(1)	Task(2)(2)	
19	Preempte	Task(2)(2)	Task(1)(6)	
20	Complete	Task(1)(6)	Task(3)(1)	2
21	Preempte	Task(3)(1)	Task(1)(7)	
22	Complete	Task(1)(7)	Task(2)(2)	1
23	Complete	Task(2)(2)	Task(3)(1)	7
24	Complete	Task(3)(1)	Task(1)(8)	12
25	Complete	Task(1)(8)	Task(3)(2)	1
27	Preempte	Task(3)(2)	Task(2)(3)	
28	Preempte	Task(2)(3)	Task(1)(9)	
29	Complete	Task(1)(9)	Task(3)(2)	2
30	Preempte	Task(3)(2)	Task(2)(3)	
31	Complete	Task(2)(3)	Task(1)(10)	7
32	Complete	Task(1)(10)	Task(3)(2)	2
33	Preempte	Task(3)(2)	Task(1)(11)	
34	Complete	Task(1)(11)	Task(3)(2)	1
35	Complete	Task(3)(2)	Task(2)(4)	11
36	Preempte	Task(2)(4)	Task(3)(3)	
37	Preempte	Task(3)(3)	Task(1)(12)	
38	Complete	Task(1)(12)	Task(3)(3)	2
39	Preempte	Task(3)(3)	Task(2)(4)	
40	Complete	Task(2)(4)	Task(1)(13)	8
41	Complete	Task(1)(13)	Task(3)(3)	2
42	Preempte	Task(3)(3)	Task(2)(5)	
43	Preempte	Task(2)(5)	Task(1)(14)	
44	Complete	Task(1)(14)	Task(3)(3)	2
45	Preempte	Task(3)(3)	Task(1)(15)	
46	Complete	Task(1)(15)	Task(2)(5)	1
47	Complete	Task(2)(5)	Task(3)(3)	7
48	Complete	Task(3)(3)	Task(1)(16)	12
49	Complete	Task(1)(16)	Task(3)(4)	1

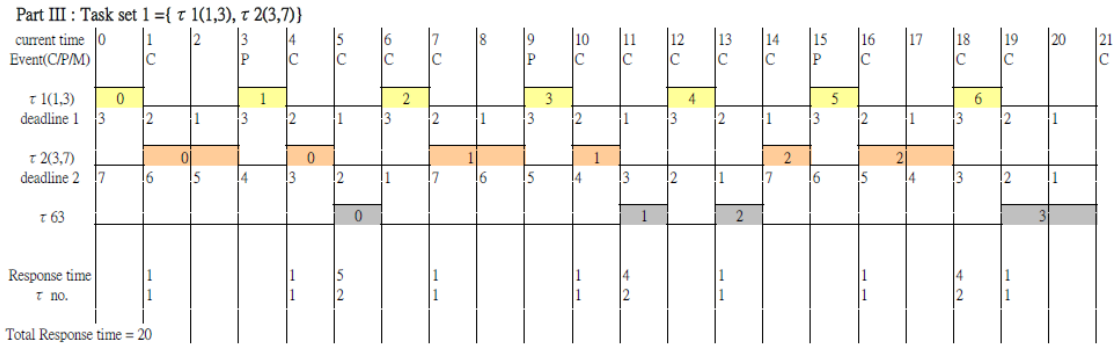
- **Schedulability Analysis**

  一個 Task 是否可以排入，同 EDF：A set of tasks is schedulable by EDF if and only if its total CPU utilization is no more than 1.
  若每個 task 工作時間都 < T 而加起來 U>1 則不能排!

  即 U = $\sum_{i=1}^{n} \frac{C_i}{P_i}$ ≦ 1

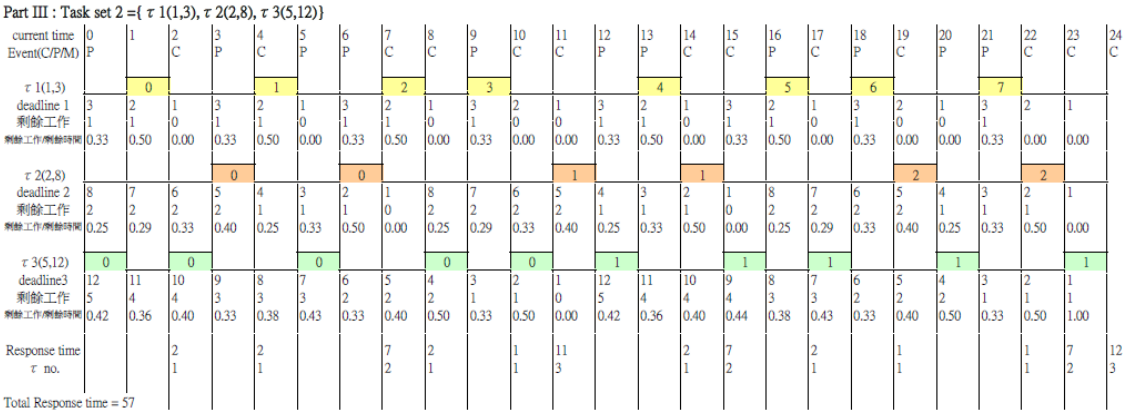
  Task Set 1 ：τ 1(1, 3)、τ 2(3, 7)

  U = $\frac{1}{3}+\frac{3}{7}=0.76$ ≦ 1 ==> 可排程



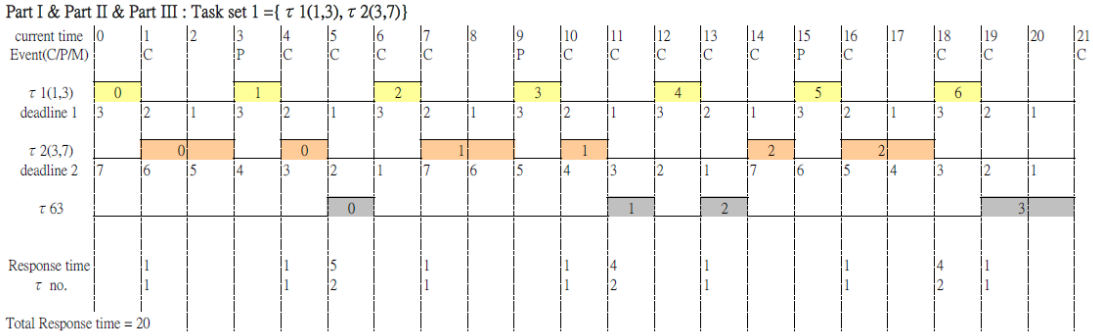
  Task Set 2 ：τ 1(1, 3)、τ 2(2, 8)、τ 3(5, 12)

  U = $\frac{1}{3}+\frac{2}{8}+\frac{5}{12}=1$ ≦ 1 ==> 可排程



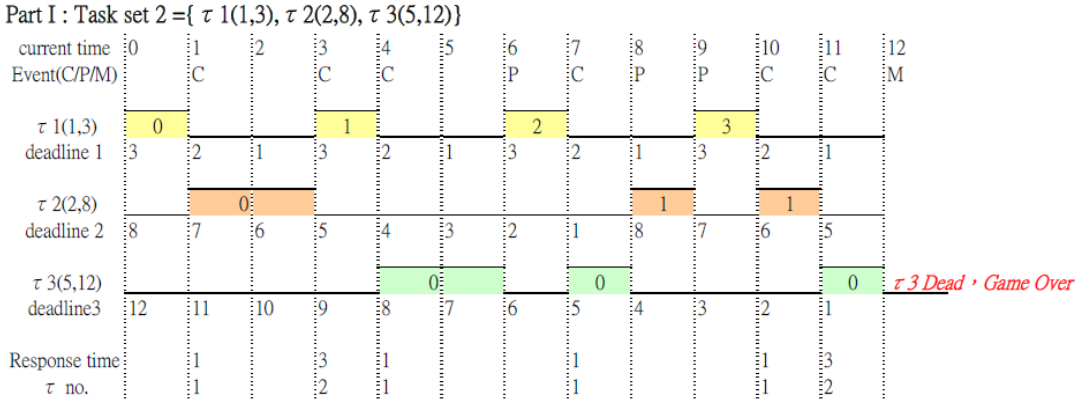
V. 心得

本次project最難的地方是第一部分，不是因為要做的事情難，而是初次接觸這個OS，對其不熟悉，需要花很多的時間去trace，也必須了解整個系統流程，才能夠知道需要修改甚麼地方、要在哪裡得到所需的資訊以及如何找出問題發生的位置，這些都花了好一段時間才逐漸地掌握，通盤了解系統流程後，後續的部分也不是甚麼大問題了。最有趣的是第三部分，藉由小組的方式能夠互相討論，每個人所看到的問題點都不同，更能夠激發出一些創意的想法來設計我們自己的排程演算法。藉由尋找相關學術論文及資訊，也能夠認識更多不同用途的演算法，甚至發現我們的演算法在幾年前剛剛被發表出來,證明我們也是有能力想出一個不錯的排程演算法。本次project中取RM、EDF、LSTR三種不同的排程法進行工作排序，並將比較三種不同排程的差異。



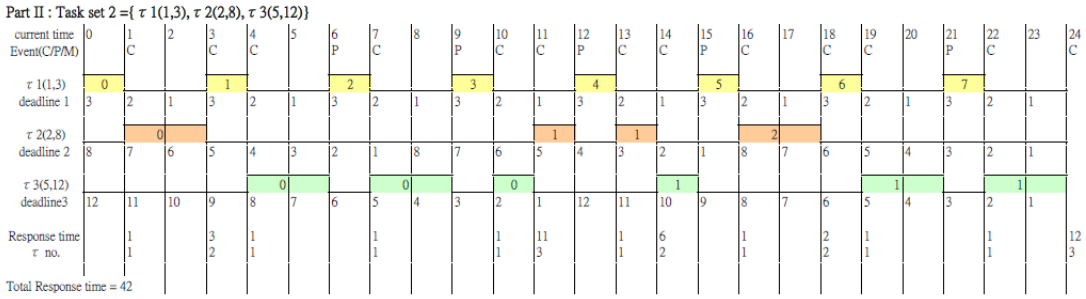
由 Task Set 1 對三種不同排程法的測試結果發現，三種不同的排程法所得到的排程結果相同 ： 每21個時間單位即可完成一次循環,其中可執行7個τ1以及3個τ2，並且將τ1與τ2分成13個小區塊完成，其中執行了16次Context Switch(考慮常態任務轉換及任務搶佔)且Total Response Time=20。
而 Task Set 2 的情況則可以分別出三種排程法不同的效能。

首先是 RM:



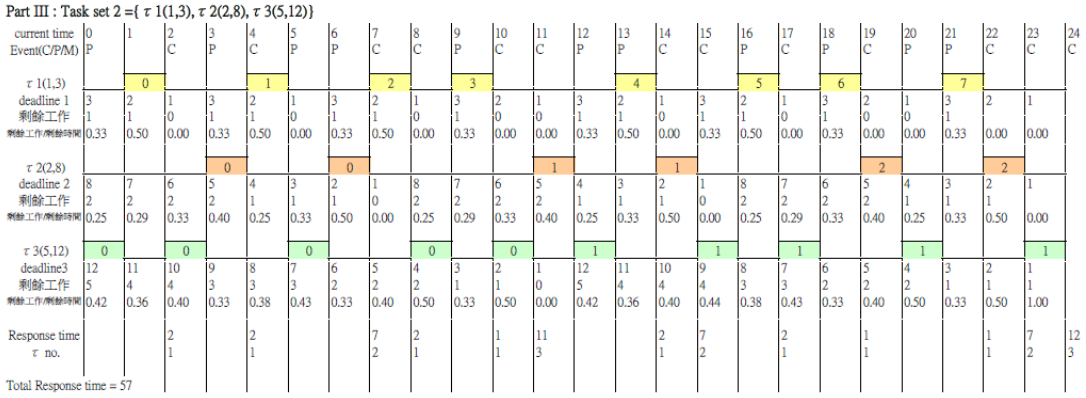
在 current time=12 時 miss deadline，從計算中也可以看出此種排程法無法將工作排入。

EDF:



每 24 個時間單位即可完成一次循環，其中可執行 8 個 τ 1、3 個 τ 2 以及 2 個 τ 3，並且將工作分成 18 個小區塊完成(分別為 8 個 τ 1 片段、4 個 τ 2 片段以及 6 個 τ 3)，其中執行了 18 次 Context Switch 且 Total Response Time=42，最長的 Response Time 為 12。

LSTR:



每 24 個時間單位即可完成一次循環，其中可執行 8 個 τ 1、3 個 τ 2 以及 2 個 τ 3，並且將工作分成 24 個小區塊完成(分別為 8 個 τ 1 片段、6 個 τ 2 片段以及 10 個 τ 3)，其中執行了 24 次 Context Switch 且 Total Response Time=57，最長的 Response Time 為 12。

從以上結果可以整理出此表格 ：

	EDF	LSTR
完成 τ 1數	8	8
完成 τ 2數	3	3
完成 τ 3數	2	2
切分個數	18	24
Context Switch數	18	24
Total Response Time	42	57
最長Response Time	12	12

由此表可發覺 EDF 是較有效率排程方法，然而 LSTR 則可兼顧工作的重要性與事情急迫性且 LSTR 可以讓每一個 Task 的進度都隨著時間的經過，而有均勻的進展。

VI. Reference

[1] Myunggwon Hwang, Dongjin Choi, Pankoo Kim, " **Least Slack Time Rate First: an Efficient Scheduling Algorithm for Pervasive Computing Environment**", Journal of Universal Computer Science, 2010