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 02

班級　　：電機碩一
學生　　：江昱霖
學號　　：M10307431

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
```

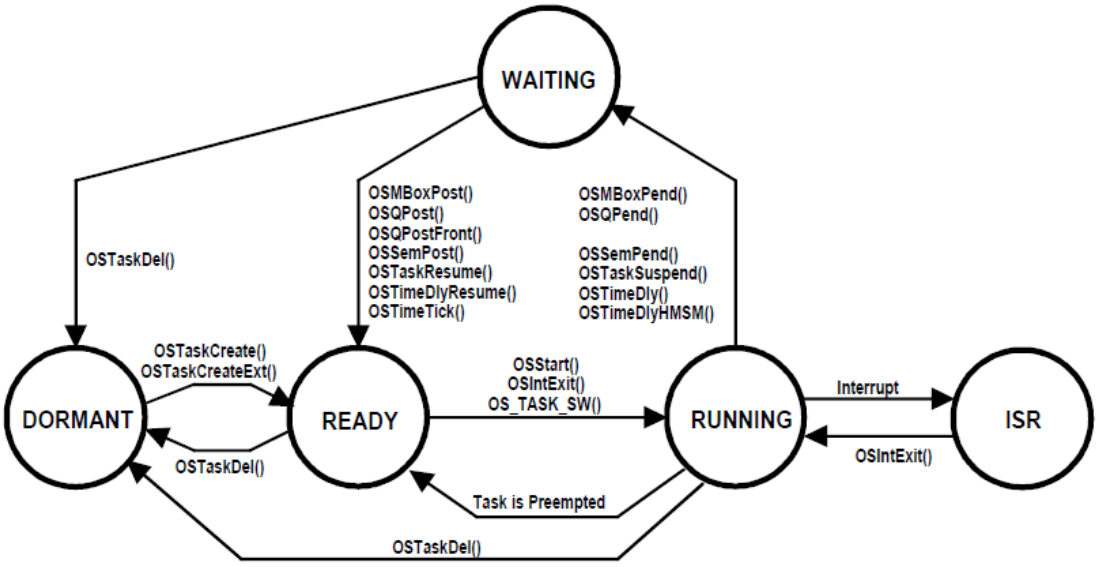
- 定義計時器倍率 SCALE，方便在硬體上執行的結果監控。

```
#ifndef __SYSTEM_H_
#define __SYSTEM_H_
#define SCALE   1
```

II. EDF (Earliest Deadline First Scheduling)

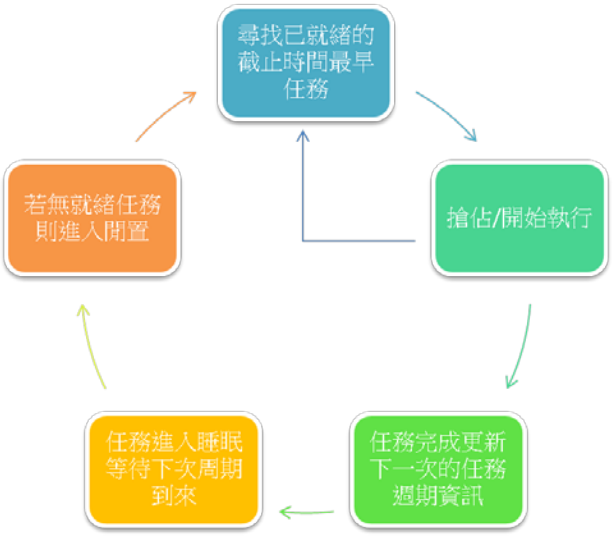
最早截止時間優先 EDF(Earliest Deadline First)是非常著名的實時排程演算法。從已就緒的任務中，優先執行截止時間最短的任務，每當有新的任務到達，必須立即計算目前截止時間最短的任務，決定新任務是否搶占當前任務，優先分配資源給截止時間最早也就是最急迫的任務，目前在單核心上為最佳的排程演算法。

- **uC/OS II 運作流程**



● 任務參數創建及初始化設定

由於 EDF 是根據截止時間最為優先權的判斷，因此在原架構下加入截止時間的相關參數及修改函式達到截止時間的設定及初始化。



1. 修改 TCB (ucos_ii.h) ，加入執行時間、週期、截止時間、響應時間、開始時間以及任務的計數，用來計算當前任務狀態及顯示的相關資訊。

```
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，用來初始化任務的設定。

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

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

Task Struct

3. 定義任務函式，以無限迴圈來表示週期性的任務，並判斷執行時間參數來得知是否已執行完畢，delay 函式來模擬任務等待下個週期的到達。

```
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);

  }
}
```

任務參數初始化

任務執行迴圈

4. 創建任務，傳入先前加入的任務資料結構，並修改該函式傳入截止時間做為初始化設定。

```
OSTaskCreateExt(task1,
                &t1,
                (void *)&task1_stk[TASK_STACKSIZE-1],
                TASK1_PRIORITY,
                TASK1_PRIORITY,
                task1_stk,
                TASK_STACKSIZE,
                NULL,
                0,
                t1.period          // deadline
                );
```

修改創建任務函式(ucos ii.h、os_task.c)，並在 OS_TCBinit(os_core.c)中做截止時間的初始。

```
#if OS_TASK_CREATE_EXT_EN > 0
INT8U          OSTaskCreateExt          (void          (*task)(void *p_arg),
                                         void           *p_arg,
                                         OS_STK         *ptos,
                                         INT8U           prio,
                                         INT16U          id,
                                         OS_STK         *pbos,
                                         INT32U          stk_size,
                                         void           *pext,
                                         INT16U          opt,
                                         INT32U          deadline);
```

```
INT8U  OS_TCBInit (INT8U prio, OS_STK *ptos, OS_STK *pbos, INT16U id, INT32U stk_size,
{
    OS_TCB     *ptcb;
#if OS_CRITICAL_METHOD == 3                                  /* Allocate storage for CPU
    OS_CPU_SR  cpu_sr = 0;
#endif

    ptcb->Deadline = deadline;                               // Deadline Initialization
```

- **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。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. 模擬執行時間(OS_TimeTick)，更新當前任務的剩餘執行時間。

```
    }
    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**搶占**資訊 (OSIntExit)，為避免任務在完成設定 delay 前被其他任務搶占，因此在部分加入 ExecTime 不等於 0 的條件來阻擋，使任務能夠完成 delay 設定後在將下個任務調度進來。

```
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 顯示任務完成及失敗資訊 (OS_Sched)，與 OSIntExit 增加部分大致相同，因務任務已完成所以必須多加週期性任務的狀態更新，包含截止時間等等。

```
            OS_SchedNew();

            if(OSTimeGet() > OSTCBCur->Deadline)     /*超過截止時間則為 Miss deadline */
                printf("\t%d\tMiss Deadline\tTask(%d)(%d)\n",OSTimeGet()/SCALE,OSTCBCur->OSTCBId,OSTCBCur->

            if (OSPrioHighRdy != OSPrioCur) {          /* No Ctx Sw if current task is highest rdy     */
                OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy];

                printf("\t%d\tComplete\tTask(%d)(%d)\tTask(%d)(%d)\t%d\n",OSTimeGet()/SCALE,OSTCBCur->OSTCB
                /* 更新週期性任務的狀態及截止新間 */
                OSTCBCur->NoJob += 1;
                OSTCBCur->Start += OSTCBCur->Period;
                OSTCBCur->ExecTime = OSTCBCur->Exec;
                OSTCBCur->Deadline += OSTCBCur->Period;

#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*/
            }
```

4. 為實現 EDF 排程，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)，搜尋就緒任務中截止時間最早的任務優先排程，若截止時間相同，則讓周期較長的任務優先執行，會有較好的響應時間及較少的 context switch 次數。

```
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
            if(OSTCBPrioTbl[i]->OSTCBDly==0) {                                 // ready task

                if((OSTCBPrioTbl[i]->Deadline < UrgentDeadline)||
                (OSTCBPrioTbl[i]->Deadline == UrgentDeadline && OSTCBPrioTbl[UrgentPrio]->Period < OSTCBPrioTbl[i]->Period)) {
                    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。

EDF 與 RM 最大的不同在於 EDF 是動態優先權的排程演算法，而 RM 是靜態的演算法。RM 是依據任務的週期給予固定的優先權，EDF 則是依據當下截止時間的早晚來訂定優先權，因此任務在每個時間點的優先權是動態改變的。由於 RM 是靜態的演算法，其最大利用率大約趨近於 0.7 左右，而 EDF 能夠保證在利用率小於等於 1 皆可排程，達到最佳系統利用率的功能。

	Priority Type	Priority Definition	Utilization Bound
RM	Fixed	Period	0.7
EDF	Dynamic	Deadline	1

- **結果(Time Tick 0 - 70)**

1. Task set 1 模擬:

```
============ EDF Task Set 1 { t1(1,4) , t2(2,5) , t3(3,15) }==================
------------------------------------------------------------------------------
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(3)(0)      3
        4       Preempte        Task(3)(0)      Task(1)(1)
        5       Complete        Task(1)(1)      Task(2)(1)      1
        7       Complete        Task(2)(1)      Task(3)(0)      2
        8       Preempte        Task(3)(0)      Task(1)(2)
        9       Complete        Task(1)(2)      Task(3)(0)      1
        12      Complete        Task(3)(0)      Task(2)(2)      12
        14      Complete        Task(2)(2)      Task(1)(3)      4
        15      Complete        Task(1)(3)      Task(2)(3)      3
        17      Complete        Task(2)(3)      Task(1)(4)      2
        18      Complete        Task(1)(4)      Task(3)(1)      2
        20      Preempte        Task(3)(1)      Task(1)(5)
        21      Complete        Task(1)(5)      Task(2)(4)      1
        23      Complete        Task(2)(4)      Task(3)(1)      3
        24      Preempte        Task(3)(1)      Task(1)(6)
        25      Complete        Task(1)(6)      Task(3)(1)      1
        27      Complete        Task(3)(1)      Task(2)(5)      12
        29      Complete        Task(2)(5)      Task(1)(7)      4
        30      Complete        Task(1)(7)      Task(2)(6)      2
        32      Complete        Task(2)(6)      Task(1)(8)      2
        33      Complete        Task(1)(8)      Task(3)(2)      1
        35      Preempte        Task(3)(2)      Task(2)(7)
```

```
        35      Preempte        Task(3)(2)      Task(2)(7)
        37      Complete        Task(2)(7)      Task(1)(9)      2
        38      Complete        Task(1)(9)      Task(3)(2)      2
        40      Preempte        Task(3)(2)      Task(1)(10)
        41      Complete        Task(1)(10)     Task(3)(2)      1
        42      Complete        Task(3)(2)      Task(2)(8)      12
        44      Complete        Task(2)(8)      Task(1)(11)     4
        45      Complete        Task(1)(11)     Task(2)(9)      1
        47      Complete        Task(2)(9)      Task(3)(3)      2
        48      Preempte        Task(3)(3)      Task(1)(12)
        49      Complete        Task(1)(12)     Task(3)(3)      1
        50      Preempte        Task(3)(3)      Task(2)(10)
        52      Complete        Task(2)(10)     Task(1)(13)     2
        53      Complete        Task(1)(13)     Task(3)(3)      1
        56      Complete        Task(3)(3)      Task(2)(11)     11
        58      Complete        Task(2)(11)     Task(1)(14)     3
        59      Complete        Task(1)(14)     Task(63)(0)     3
        60      Complete        Task(63)(0)     Task(1)(15)
        61      Complete        Task(1)(15)     Task(2)(12)     1
        63      Complete        Task(2)(12)     Task(3)(4)      3
        64      Preempte        Task(3)(4)      Task(1)(16)
        65      Complete        Task(1)(16)     Task(2)(13)     1
        67      Complete        Task(2)(13)     Task(3)(4)      2
        68      Preempte        Task(3)(4)      Task(1)(17)
        69      Complete        Task(1)(17)     Task(3)(4)      1
        72      Complete        Task(3)(4)      Task(2)(14)     12
```

2. Task set 2 模擬:

```
======= EDF Task Set 2 ( t1(1,5) , t2(2,6) , t3(3,10) , t4(5,15) )=============
-------------------------------------------------------------------------------
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(3)(0)      3
        6       Complete        Task(3)(0)      Task(1)(1)      6
        7       Complete        Task(1)(1)      Task(2)(1)      2
        9       Complete        Task(2)(1)      Task(4)(0)      3
        14      Complete        Task(4)(0)      Task(1)(2)      14
        15      Complete        Task(1)(2)      Task(2)(2)      5
        17      Complete        Task(2)(2)      Task(3)(1)      5
        20      Complete        Task(3)(1)      Task(1)(3)      10
        21      Miss Deadline   Task(1)(3)
```

- **SchedulabilityAnalysis**

Utilization Bound Test : U = $\sum_{i=1}^{n} \frac{C_i}{P_i} \leqq 1$

➢ Task Set 1 ：$\tau 1(1,4)$、$\tau 2(2,5)$ 、$\tau 3(3,15)$

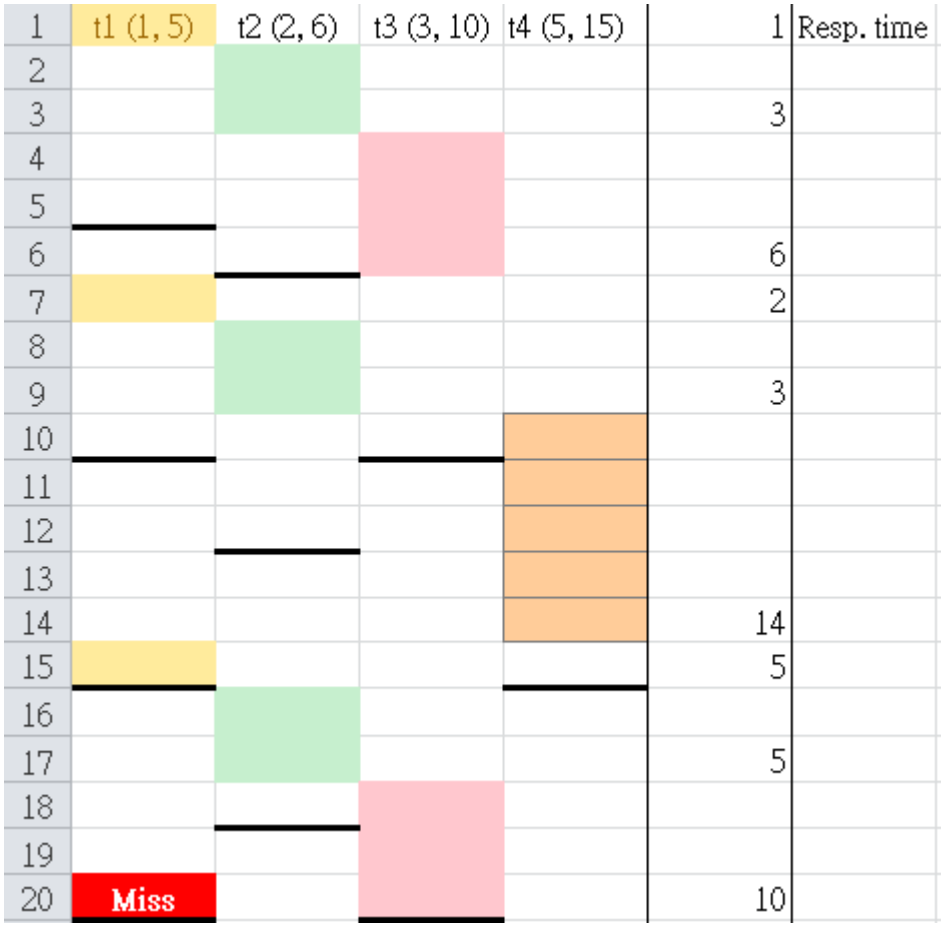
U = $\frac{1}{4}+\frac{2}{5}+\frac{3}{15}=0.85 \leqq 1$ → 可排程

	t1 (1, 4)	t2 (2, 5)	t3 (3, 15)	1	Resp. time
2					
3				3	
4					
5				1	
6					
7				2	
8					
9				1	
10					
11					
12				12	
13					
14				4	
15				3	
16					
17				2	
18				2	
19					
20					

➢ Task Set 2 ：$\tau 1(1,5)$、$\tau 2(2,6)$、$\tau 3(3,10)$、$\tau 4(5,15)$

U = $\frac{1}{5}+\frac{2}{6}+\frac{3}{10}+\frac{5}{15}=1.167 \geqq 1$ → 排程失敗

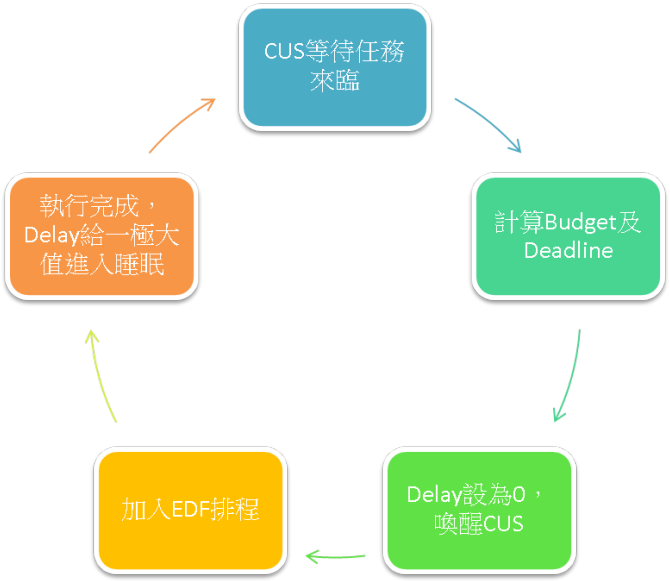
由結果分析可知，在時間點 20 時 t3 完成，但 t1 仍未執行，因此 t1 則 miss deadline。



III. CUS (Constant Utilization Server)

CUS 係用來處理非週期性任務的一種排程架構，參考週期性任務的利用率來設定 Server Size，保證一段時間內系統上的利用率能夠在要求內，並保證非週期性任務能夠在該時間內執行完成。每當非週期性任務來臨時，會依據該任務的執行執間及 Server Size 計算出 Deadline，及補充可執行的 Budget，並依據 EDF 排程，使非週期性任務能在不影響週期性任務下順利執行。

因此接續已完成的 EDF 部分，我們額外創建一個專門處理非週期性任務的任務，並在一開始給予極大的截止時間使系統不會將它納入排程。



- **任務參數創建及初始化設定**

1. 先創建模擬處理非週期性任務的任務函式並定義其優先權(hello_ucosii.c)及 CUS Server Size (os_core.c)，在任務創建之初設定其截止時間為一極大值。

```
/* Definition of Task Priorities */
#define TASK1_PRIORITY      1
#define TASK2_PRIORITY      2
#define TASK3_PRIORITY      3
#define APTASK_PRIORITY     4
```

```
#define ServerSize_numerator 2
#define ServerSize_denominator 8
```

```
OSTaskCreateExt(aptask,
                NULL,
                (void *)&aptask_stk[TASK_STACKSIZE-1],
                APTASK_PRIORITY,
                APTASK_PRIORITY,
                aptask_stk,
                TASK_STACKSIZE,
                NULL,
                0,
                65535
                );
```

Deadline assign

2. 定義處理非週期性任務之任務函式，當任務完成時將截止時間設一極大值使之不會被排程，當有一新的非週期性任務到達時，會再將截止時間更新為計算的結果並使之重新納入排程比較。

```
void aptask(task* pdata)
{
  int start = 0;
  int end;
  int delay;

  while (1)
  {

    while(OSTCBCur->ExecTime){}

    end = OSTimeGet();
    delay = OSTCBCur->Deadline-end;
    OSTCBCur->Resp = end-start;
    OSTCBCur->Deadline = 65535;
    OSTimeDly(65535);
```

- **uC/OS ii Kernel 修改**

  CUS 是一考慮非週期性任務且基於 EDF 的動態優先權的排程方式。因此在非週期性任務加入時，會依據其執行時間來計算及設定 Budget 及 Deadline，並與原週期性任務一齊加入 EDF 排程。由於尚無非週期性任務時，該 CUS 被設定一極大的 Deadline 使之不會被排程，因此我們在 Time Tick 中判斷當有一非週期性任務來臨時，計算並設定其 Budget 及 Deadline，並將其 Delay 設為零使之立刻進入 ready list 開始運行，當非週期性任務完成時將 Delay 設一極大值使之進入 Sleep 等待下次任務的來臨。

1. 非週期性任務參數設定(OS_TimeTick)，判斷非週期性任務是否來臨，並根據其執行時間計算 Budget 及 Deadline，並將 Delay 設為 0 使之立即就緒加入排程。

```
/* Aperiodic Task1 (arrive:1,exec:3) */
if(OSTimeGet()==1*SCALE){
    printf(" \t%d====== Aperiodic Task Arrive (arrive:1,exec:3) =======\n ",OSTimeGet()/SCALE);
    OSTCBPrioTbl[APTASK_PRIORITY]->Start = OSTimeGet();
    OSTCBPrioTbl[APTASK_PRIORITY]->Exec = 3*SCALE;
    OSTCBPrioTbl[APTASK_PRIORITY]->ExecTime = OSTCBPrioTbl[APTASK_PRIORITY]->Exec;
    OSTCBPrioTbl[APTASK_PRIORITY]->Deadline = OSTimeGet()+OSTCBPrioTbl[APTASK_PRIORITY]->Execbp*ServerSize_denominator/ServerSize_numerator;
    OSTCBPrioTbl[APTASK_PRIORITY]->OSTCBDly = 0;
}
/* Aperiodic Task1 (arrive:13,exec:5) */
if(OSTimeGet()==13*SCALE){
    printf(" \t%d====== Aperiodic Task Arrive (arrive:13,exec:5) =======\n ",OSTimeGet()/SCALE);
    OSTCBPrioTbl[APTASK_PRIORITY]->Start = OSTimeGet();
    OSTCBPrioTbl[APTASK_PRIORITY]->Exec = 5*SCALE;
    OSTCBPrioTbl[APTASK_PRIORITY]->ExecTime = OSTCBPrioTbl[APTASK_PRIORITY]->Exec;
    OSTCBPrioTbl[APTASK_PRIORITY]->Deadline = OSTimeGet()+OSTCBPrioTbl[APTASK_PRIORITY]->Execbp*ServerSize_denominator/ServerSize_numerator;
    OSTCBPrioTbl[APTASK_PRIORITY]->OSTCBDly = 0;
}
```

2. 由於非週期性任務只來一次，所以在任務完成更新任務下次狀態時要將非週期性排除，才不會讓非週期性任務重複執行。

```
if(OSTCBCur->OSTCBPrio != APTASK_PRIORITY)
{
    OSTCBCur->Start += OSTCBCur->Period;
    OSTCBCur->ExecTime = OSTCBCur->Exec;
    OSTCBCur->Deadline += OSTCBCur->Period;
}
```

- **問題與討論**

雖然 DS 能夠處理非週期性任務，但由於它是固定優先權(Fixed Priority driven)的運作，因此若要搭配動態優先權的排程演算法(如 EDF)是不可行的，必須使用動態的 server(如 deadline driven)才能與 EDF 搭配運作。CUS 的基本精神在於給予非週期性任務一定固定的系統資源利用率，保證在依據其執行時間條件下計算出的截止時間內能夠被排程成功，雖能夠解決非週期性任務的排程問題，但在截止時間內完成且有另一個非週期性任務來臨，卻無法利用其中間的 idle time 使在等待的非週期性任務能夠先行執行。因此有另一個相似的演算法 TBS(Total Bandwidth Server)，與 CUS 最大不同在於當有頻寬可使用時則提供給予使用，計算依據當下時間及原可提供服務的時間得到兩個截止時間，取較大的截止時間並補充 Budget，使在等待的非週期性任務能夠有機會提早執行，這樣能夠有機會完整利用 CUS 尚未更新 Budget 這段時間的 idle time。

- **結果**

1. Task set :

   Server Size = 2/8

   Aperiodic job Arrival time Execution time

   1 1 3

   2 13 5

```
============ CUS Task Set 1 ( t1(1,5) , t2(2,8) , t3(3,10) )==================
========================== CUS Server Size = 2/8 ==============================
------------------------------------------------------------------------------
Current Time     Event       From Task ID        To Task ID      Response Time
------------------------------------------------------------------------------
        1====== Aperiodic Task Arrive (arrive:1,exec:3,deadline:13) =======
        1       Complete        Task(1)(0)      Task(2)(0)      1
        3       Complete        Task(2)(0)      Task(3)(0)      3
        6       Complete        Task(3)(0)      Task(1)(1)      6
        7       Complete        Task(1)(1)      Task(4)(0)      2
        10      Complete        Task(4)(0)      Task(1)(2)      9
        11      Complete        Task(1)(2)      Task(2)(1)      1
        13====== Aperiodic Task Arrive (arrive:13,exec:5,deadline:33) =======
        13      Complete        Task(2)(1)      Task(3)(1)      5
        16      Complete        Task(3)(1)      Task(1)(3)      6
        17      Complete        Task(1)(3)      Task(2)(2)      2
        19      Complete        Task(2)(2)      Task(4)(1)      3
        20      Preempte        Task(4)(1)      Task(1)(4)
        21      Complete        Task(1)(4)      Task(3)(2)      1
        24      Complete        Task(3)(2)      Task(2)(3)      4
        25      Preempte        Task(2)(3)      Task(1)(5)
        26      Complete        Task(1)(5)      Task(2)(3)      1
        27      Complete        Task(2)(3)      Task(4)(1)      3
        31      Complete        Task(4)(1)      Task(1)(6)      18
        32      Complete        Task(1)(6)      Task(3)(3)      2
        35      Complete        Task(3)(3)      Task(2)(4)      5
        37      Complete        Task(2)(4)      Task(1)(7)      5
        38      Complete        Task(1)(7)      Task(63)(0)     3
        40      Complete        Task(63)(0)     Task(1)(8)
        41      Complete        Task(1)(8)      Task(2)(5)      1
        43      Complete        Task(2)(5)      Task(3)(4)      3
        46      Complete        Task(3)(4)      Task(1)(9)      6
        47      Complete        Task(1)(9)      Task(63)(1)     2
        48      Complete        Task(63)(1)     Task(2)(6)
        50      Complete        Task(2)(6)      Task(1)(10)     2
        51      Complete        Task(1)(10)     Task(3)(5)      1
        54      Complete        Task(3)(5)      Task(63)(2)     4
        55      Complete        Task(63)(2)     Task(1)(11)
        56      Complete        Task(1)(11)     Task(2)(7)      1
        58      Complete        Task(2)(7)      Task(63)(3)     2
        60      Complete        Task(63)(3)     Task(1)(12)
        61      Complete        Task(1)(12)     Task(3)(6)      1
        64      Complete        Task(3)(6)      Task(2)(8)      4
        65      Preempte        Task(2)(8)      Task(1)(13)
        66      Complete        Task(1)(13)     Task(2)(8)      1
        67      Complete        Task(2)(8)      Task(63)(4)     3
        70      Complete        Task(63)(4)     Task(1)(14)
```

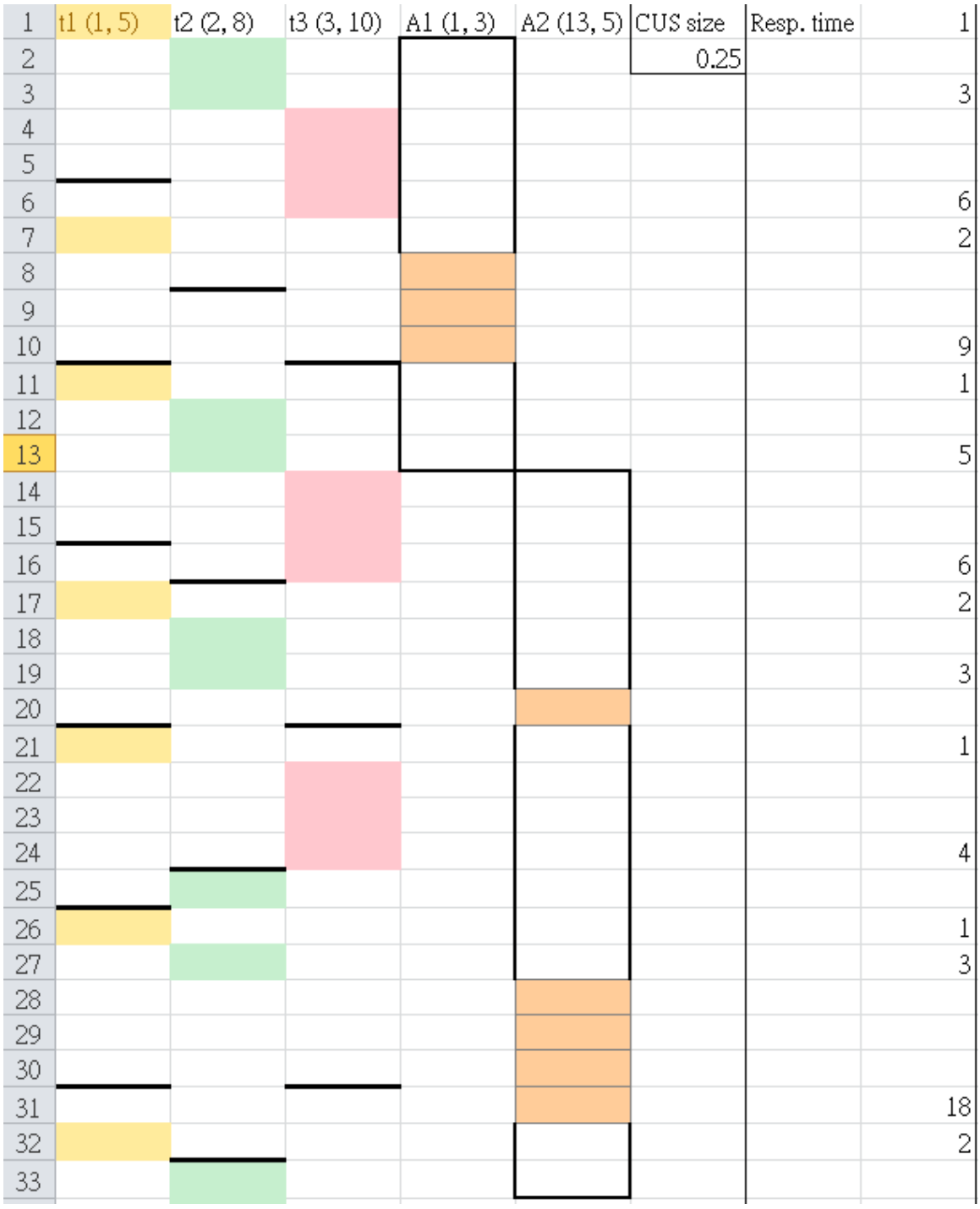
- **SchedulabilityAnalysis**

Server Size = 2/8,

Initially, $e_s = 0$, d=0

當 TimeTick=1 時，A1 來臨，執行時間為 3，$e_s = 3$，d=1+3*4=13。

當 TimeTick=13 時，A2 來臨，執行時間為 5，$e_s = 5$，d=13+5*4=33。。



IV. CUS& Button-triggered aperiodic job

此部分是將 CUS 實現在硬體上，利用硬體上的按鈕觸發來模擬非週期性任務的到來。由於使用到硬體的部分，所以需將一些硬體的設定寫入程式，由於硬體上會依照其真實時脈運作，為方便結果的監看及操作上的方便，因此會去調整先前環境設定上的時間倍率，使任務的時間相關參數皆放大倍數，並設定按鈕及 LCD 螢幕的 IO 設定，在短固定時間內去檢查按鈕是否被按下，判斷非週期性任務的到來。

- **任務參數創建及初始化設定**

1. 首先調整時間倍率，使能夠方便操作及觀察。(system.h)

```
#ifndef __SYSTEM_H_
#define __SYSTEM_H_
#define SCALE   100
```

2. 定義處理非週期性任務之任務函式，一開始 LCD 上會顯示”Press the button”，只要按下按鈕就會依據按下時刻創建一個非週期性任務，並顯示該任務的到達時間及執行時間。非週期性任務執行中，LCD 會即時顯示剩餘時間及截止時間。當任務完成時將截止時間設一極大值使之不會被排程，LCD 顯示完成的時間，當有一新的非週期性任務到達時，會再將截止時間更新為計算的結果並使之重新納入排程比較。

```
void aptask(task* pdata)
{
  FILE  *lcd;
  lcd = fopen("/dev/lcd", "w");
  fprintf(lcd, "Press the button\n\n");

  while (1)
  {
    while(OSTCBCur->ExecTime)
    {
        fprintf(lcd, "Remain /Deadline\n%6d / %7d\n", OSTCBCur->ExecTime/SCALE, OSTCBCur->Deadline/SCALE);
    }
    fprintf(lcd, "Complete %lu \n\n", OSTime/SCALE);

    OSTCBCur->Deadline = 65535;
    OSTimeDly(65535);
```

- **uC/OS ii Kernel 修改**

  繼承先前完成的 CUS 架構，利用按鈕觸發來模擬非週期性任務的加入，會依據其執行時間來計算及設定 Budget 及 Deadline，並與原週期性任務一齊加入 EDF 排程。因此我們在 Time Tick 中判斷每隔一小段時間就檢查按鈕是否有被觸發，當有一非週期性任務來臨時，計算並設定其 Budget 及 Deadline，並將其 Delay 設為零使之立刻進入 ready list 開始運行，當非週期性任務完成時將 Delay 設一極大值使之進入 Sleep 等待下次任務的來臨。

1. 宣告一 bp 全域變數紀錄隨機產生的非週期性任務執行時間。(ucosii.h)

```
*********************************************************************
*                                                  GLOBAL VARIABLES
*********************************************************************
*/

OS_EXT  INT32U             OSCtxSwCtr;                /* Counter
OS_EXT  INT32U             bp;
```

2. 定義並將控制硬體的相關資源包含進來。(os_core.c)

```
#ifndef  OS_MASTER_FILE
#define  OS_GLOBALS
#include <ucos_ii.h>
#include <stdio.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include <stdlib.h>
#define NONE_PRESSED 0xF
alt_u8  buttons;
alt_u8  led = 0x01;
int counter = 1;
#endif
#define APTASK_PRIORITY     4

#define ServerSize_numerator 2
#define ServerSize_denominator 8
```

3. 在 OS_TimeTick 中每隔 100 個時間單位就會檢查按鈕是否被觸發，當按鈕(bp)被觸發代表非週期性任務來臨，會隨機產生一執行時間，並根據其執行時間計算 Budget 及 Deadline，並將 Delay 設為 0 使之立即就緒加入排程，且在 LCD 上顯示任務來臨的時間及執行時間。

```
OSTCBCur->ExecTime--;
srand(OSTime);
counter = (counter+1)%100;
if(OSTime%100 == 0)
{
    IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, led);
    buttons = IORD_ALTERA_AVALON_PIO_DATA(BUTTON_PIO_BASE);
    //printf("%d\n", OSTCBPrioTbl[APTASK_PRIORITY]->ExecTime/SCALE);
    if(buttons != NONE_PRESSED)
    {
        if(led >= 0x80)
            led = 0x01;
        else
            led = led<<1;
        IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, led);
        bp = rand()%2000+SCALE;
        printf("Aperiodic task arrives at time %d", OSTime/SCALE);
        printf(" the computation time is %d.\n", bp/SCALE);
        FILE  *lcd;
        lcd = fopen("/dev/lcd", "w");
        fprintf(lcd, "Arrive / Exec.  \n%6d / %7d\n", OSTime/SCALE, bp/SCALE);
        buttons = NONE_PRESSED;
        OSTCBPrioTbl[APTASK_PRIORITY]->Start = OSTime;
        OSTCBPrioTbl[APTASK_PRIORITY]->ExecTime = bp;
        OSTCBPrioTbl[APTASK_PRIORITY]->Deadline = OSTime+ bp*ServerSize_denominator/ServerSize_numerator;
        OSTCBPrioTbl[APTASK_PRIORITY]->OSTCBDly = 0;
```

按鈕偵測

隨機產生執行時間

設定任務參數

4. 由於非週期性任務只來一次，所以在任務完成更新任務下次狀態時要將非週期性排除，才不會讓非週期性任務重複執行。

```
if(OSTCBCur->OSTCBPrio != APTASK_PRIORITY)
{
    OSTCBCur->Start += OSTCBCur->Period;
    OSTCBCur->ExecTime = OSTCBCur->Exec;
    OSTCBCur->Deadline += OSTCBCur->Period;
}
```

- **結果**

1. Task set :

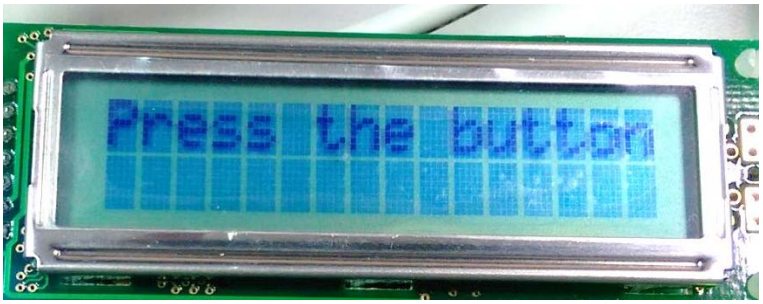
   Server Size = 2/8

Aperiodic job	Arrival time	Execution time	Deadline
A1	56	7	84

➢ 軟體資訊顯示

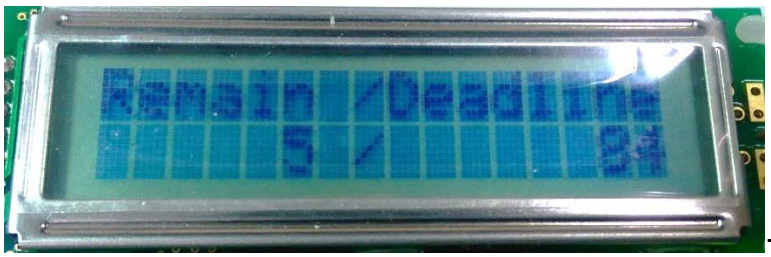
```
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

============ CUS Task Set 1 ( t1(1,5) , t2(2,8) , t3(3,10) )==================
========================= CUS Server Size = 2/8 ==============================
------------------------------------------------------------------------------
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(3)(0)       3
          6      Complete       Task(3)(0)       Task(1)(1)       6
          7      Complete       Task(1)(1)       Task(63)(0)      2
          8      Complete       Task(63)(0)      Task(2)(1)
          10     Complete       Task(2)(1)       Task(1)(2)       2
          11     Complete       Task(1)(2)       Task(3)(1)       1
          14     Complete       Task(3)(1)       Task(63)(1)      4
          15     Complete       Task(63)(1)      Task(1)(3)
          16     Complete       Task(1)(3)       Task(2)(2)       1
          18     Complete       Task(2)(2)       Task(63)(2)      2
          20     Complete       Task(63)(2)      Task(1)(4)
          21     Complete       Task(1)(4)       Task(3)(2)       1
          24     Complete       Task(3)(2)       Task(2)(3)       4
          25     Preempte       Task(2)(3)       Task(1)(5)
          26     Complete       Task(1)(5)       Task(2)(3)       1
          27     Complete       Task(2)(3)       Task(63)(3)      3
          30     Complete       Task(63)(3)      Task(1)(6)
          31     Complete       Task(1)(6)       Task(3)(3)       1
          34     Complete       Task(3)(3)       Task(2)(4)       4
          36     Complete       Task(2)(4)       Task(1)(7)       4
          37     Complete       Task(1)(7)       Task(63)(4)      2
          40     Complete       Task(63)(4)      Task(1)(8)
          34     Complete       Task(3)(3)       Task(2)(4)       4
          36     Complete       Task(2)(4)       Task(1)(7)       4
          37     Complete       Task(1)(7)       Task(63)(4)      2
          40     Complete       Task(63)(4)      Task(1)(8)
          41     Complete       Task(1)(8)       Task(2)(5)       1
          43     Complete       Task(2)(5)       Task(3)(4)       3
          46     Complete       Task(3)(4)       Task(1)(9)       6
          47     Complete       Task(1)(9)       Task(63)(5)      2
          48     Complete       Task(63)(5)      Task(2)(6)
          50     Complete       Task(2)(6)       Task(1)(10)      2
          51     Complete       Task(1)(10)      Task(3)(5)       1
          54     Complete       Task(3)(5)       Task(63)(6)      4
          55     Complete       Task(63)(6)      Task(1)(11)
Aperiodic task arrives at time 56 the computation time is 7.
          56     Complete       Task(1)(11)      Task(2)(7)       1
          58     Complete       Task(2)(7)       Task(4)(0)       2
          60     Preempte       Task(4)(0)       Task(1)(12)
          61     Complete       Task(1)(12)      Task(3)(6)       1
          64     Complete       Task(3)(6)       Task(2)(8)       4
          65     Preempte       Task(2)(8)       Task(1)(13)
          66     Complete       Task(1)(13)      Task(2)(8)       1
          67     Complete       Task(2)(8)       Task(4)(0)       3
          70     Preempte       Task(4)(0)       Task(1)(14)
          71     Complete       Task(1)(14)      Task(3)(7)       1
          74     Complete       Task(3)(7)       Task(2)(9)       4
          76     Complete       Task(2)(9)       Task(1)(15)      4
          77     Complete       Task(1)(15)      Task(4)(0)       2
          79     Complete       Task(4)(0)       Task(63)(7)      23
          80     Complete       Task(63)(7)      Task(1)(16)
          81     Complete       Task(1)(16)      Task(2)(10)      1
          83     Complete       Task(2)(10)      Task(3)(8)       3
```

➢ 硬體資訊顯示

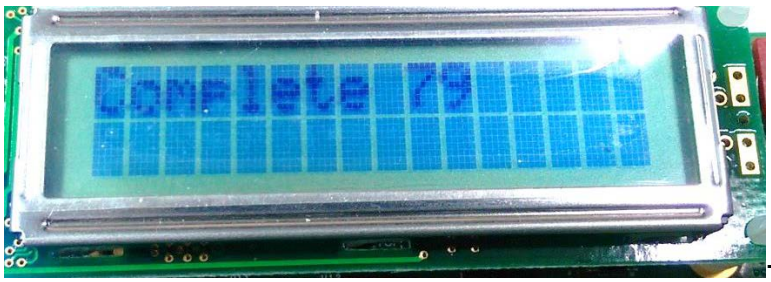


→開始畫面

→任務來臨



→任務執行



→任務完成

→硬體端顯示與軟體端結果相吻合。

- **SchedulabilityAnalysis**

  Server Size = 2/8,
  Initially, $e_s = 0$, d=0
  當 TimeTick=56 時，非週期性任務來臨，執行時間為 7，
  $e_s = 7$ ，d=56+7*4= 84。
  加入 EDF 排程，TimeTick=79 時完成任務。

V. 心得

本次project又再一次的實作EDF的部分，有了上次的基礎，在此部分只需將輸入修改一下即可完成。由於上次的任務型態都是週期性任務，而這次加入了處理非週期性任務的機制，使用了CUS的方式來針對不可預期的非週期性任務的排程進行處理。由於之前有額外建一個資料結構將任務的參數結構化傳入創建任務的函式中，剛開始不想要修改創任務的函式，因此在截止時間的初始化部分想了很久，最後仍是放棄決定使用修改任務創建函式的方式，將截止時間一開始就傳進函式在TCB初始化實設定好截止時間，過了一關又一關，結果發現非週期性任務不會動，發現原來在做非週期性任務設定時忘記給他執行時間也就是budget的設定，如此任務當然不會動，最後也順利完成CUS的部分。這次比較有趣的地方是能夠將CUS實作在開發版上，實際的透過按鈕觸發去隨機創建一非週期性任務。透過程式的撰寫以及實際開發版的操作，當中不僅能夠以自我思考的方式以及實際碰到的問題，對於課堂上的理論有更深刻的理解及內化，也能夠發現自己在理解上的盲點是甚麼，課堂上的理論實現在實際的操作上，不僅僅是身為理工人應有的能力，也是教育最初希望達到的目標與宗旨。