Embedded OS Implementation

Project 3

學號 ： M11407517 黃志傑

[PART I] NPCS Implementation

Objective:

Implement the non-preemptible critical section (NPCS) based on the RM scheduler in uC/OS-II.

- ucos_ii.h:

```c
#include <string.h>
#define SYSTEM_END_TIME 100

#define INPUT_FILE_NAME "./TaskSet.txt"
#define OUTPUT_FILE_NAME "./Output.txt"
#define MAX 20
#define INFO 10
#define RM 0
#define FIFO 1
#define ALGORITHM RM
#define NPCS 0
#define CPP 1
#define PROTOCOL NPCS

#define MAX_PRINTF_BUFFER 16

    typedef struct task_para_set {
        INT16U TaskID;
        INT16U TaskArriveTime;
        INT16U TaskExecutionTime;
        INT16U TaskPeriodic;
        INT16U TaskNumber;
        INT16U TaskPriority;
        INT16U InitArriveTime;
        INT16U Ready;
        INT16U job_no;                  //Job number
        INT16U RemainTime;              //任務剩餘時間
        INT16U deadline;                //任務deadline
        INT16U ResponseTime;            //紀錄response time
        INT16U Delay;                   //任務需要被OSTimeDly多久
        INT16U BlockTime;
        INT16U PreemptiveTime;
        INT16U R1Lock;
        INT16U R1UnLock;
        INT16U R2Lock;
        INT16U R2UnLock;
    } task_para_set;
```

- 新增 task_para_set 的 member :
  - BlockTime : 任務被 Block 的時間
  - PreemptiveTime :任務被 Preemp 的時間
  - R1Lock、R1UnLock : Resoure1 被 Lock、Unlock 時間
  - R2Lock、R2UnLock : Resource2 被 Lock、Unlock 時間
- Define 使用的 PROCOCOL
  - NPCS(non-preemptible critical section)
  - CPP(Ceiling Priority Protocol)

- app_hooks.h :

```c
        while (ptr != NULL) {
            TaskInfo[i] = atoi(ptr);
            ptr = strtok_s(NULL, " ", &pTmp);
            if (i == 0) {
                TASK_NUMBER++;
                TaskParameter[j].TaskID = TASK_NUMBER;
            }
            else if (i == 1) {
                TaskParameter[j].InitArriveTime = TaskInfo[i];
                TaskParameter[j].TaskArriveTime = TaskInfo[i];
            }
            else if (i == 2) {
                TaskParameter[j].TaskExecutionTime = TaskInfo[i];
                TaskParameter[j].RemainTime = TaskInfo[i];
            }
            else if (i == 3) {
                TaskParameter[j].TaskPeriodic = TaskInfo[i];
            }
            else if (i == 4)
                TaskParameter[j].R1Lock = TaskInfo[i];
            else if (i == 5)
                TaskParameter[j].R1UnLock = TaskInfo[i];
            else if (i == 6)
                TaskParameter[j].R2Lock = TaskInfo[i];
            else if (i == 7)
                TaskParameter[j].R2UnLock = TaskInfo[i];
            TaskParameter[j].deadline = TaskParameter[j].TaskArriveTime + TaskParameter[j].TaskPeriodic;
            TaskParameter[j].BlockTime = 0;
            TaskParameter[j].PreemptiveTime = 0;
            TaskParameter[j].Ready = 0;
            i++;
        }

        TaskParameter[j].TaskPriority = j;
        j++;
    }
```

➢ 讀取 Taskset.txt 資料、並初始化 BlockTime、PreemptiveTime

- App_hooks.h(App_TimeTickHook) :

```c
void  App_TimeTickHook (void)
{
#if (APP_CFG_PROBE_OS_PLUGIN_EN == DEF_ENABLED) && (OS_PROBE_HOOKS_EN > 0)
    OSProbe_TickHook();
#endif

    OS_TCB* ptcb;
    task_para_set* pdata;
    INT32U CurrentTime;
    // 因為 Hook 是在 OSTime++ 之前執行的，所以我們要用 "下一刻的時間" 來判斷到達
    CurrentTime = OSTime + 1;
    ptcb = OSTCBList;
    while (ptcb != (OS_TCB*)0) {
        if (ptcb->OSTCBExtPtr != (void*)0) {
            pdata = (task_para_set*)ptcb->OSTCBExtPtr;
            if (CurrentTime >= pdata->InitArriveTime) {
                INT32U time_diff = CurrentTime - pdata->InitArriveTime;

                if (pdata->TaskPeriodic > 0 && (time_diff % pdata->TaskPeriodic) == 0) {
                    pdata->RemainTime = pdata->TaskExecutionTime;
                    pdata->Ready = 1;
                }
            }
        }
        // 檢查下一個 TCB
        ptcb = ptcb->OSTCBNext;
    }
```

➢ 將任務下次 ArriveTime 抵達時，補充 RemainTime=ExecutionTime

➢ 檢查整個 OSTCBList

- main.c(task_function) :

```c
void task(void* p_arg) {
    task_para_set* task_data = (task_para_set*)p_arg;
    INT32U current_time;
    INT32U executed_time;
    INT8U err;
    INT8U base_prio = task_data->TaskPriority;
    INT8U current_prio_log;
    INT8U target_prio_log;
    current_time = OSTimeGet();
    while (stopAlltask == 0) {
        task_data->Ready = 1;
        while (task_data->RemainTime > 0 && stopAlltask == 0) {
            if (stopAlltask == 1) {
                break;
            }

            executed_time = task_data->TaskExecutionTime - task_data->RemainTime;
            // 檢查 R1 Lock
            if ((executed_time == task_data->R1Lock) && (task_data->R1UnLock != 0)) {
#if PROTOCOL == NPCS
                LOG_print(3, "./Output.txt", "%d\tLockResource\ttask(%2d)(%2d)\t%s\n",
                    OSTime, task_data->TaskID, task_data->job_no, "R1");
                OSSchedLock(); // 禁止搶佔
```

```c
            // 檢查 R2 Lock
            if ((executed_time == task_data->R2Lock) && (task_data->R2UnLock != 0)) {

#if PROTOCOL == NPCS
                LOG_print(3, "./Output.txt", "%d\tLockResource\ttask(%2d)(%2d)\t%s\n",
                    OSTime, task_data->TaskID, task_data->job_no, "R2");
                OSSchedLock(); // 禁止搶佔
```

```c
            // 檢查 R1 Unlock
            if ((executed_time == task_data->R1UnLock) && (task_data->R1UnLock != 0)) {

#if PROTOCOL == NPCS
                LOG_print(3, "./Output.txt", "%d\tUnlockResource\ttask(%2d)(%2d)\t%s\t\n",
                    OSTime, task_data->TaskID, task_data->job_no, "R1");
                OSSchedUnlock(); // 恢復排程
```

```c
            // 檢查 R2 Unlock
            if ((executed_time == task_data->R2UnLock) && (task_data->R2UnLock != 0)) {

#if PROTOCOL == NPCS
                LOG_print(3, "./Output.txt", "%d\tUnlockResource\ttask(%2d)(%2d)\t%s\n",
                    OSTime, task_data->TaskID, task_data->job_no, "R2");
                OSSchedUnlock(); // 恢復排程
```

- 檢查 executed_time 是否為 Lock 或 UnLock 的時間
    - Lock : OSSchedLock() 禁止搶斷
    - UnLock : OSSchedUnlock() 恢復排程
- Print 出 Lock、UnLock 訊息

- os_core.c(OS_SchedNew) :

```c
static  void  OS_SchedNew(void)
{
#if OS_LOWEST_PRIO <= 63u                        /* See if we support up to 64 tasks                */
    INT8U   y;
    static INT8U first_check = 0u;

    if (first_check == 0u) {
        OS_TCB* ptcb;
        INT32U delay_ticks;
        first_check = 1u;
        ptcb = OSTCBList;

        while (ptcb != (OS_TCB*)0 && (ptcb ->OSTCBExtPtr !=NULL)) {
            delay_ticks = ((task_para_set*)(ptcb->OSTCBExtPtr))->InitArriveTime;
            if (delay_ticks > 0) {
                ptcb->OSTCBDly = delay_ticks;
                // 從 Ready Table 中移除該任務
                if ((OSRdyTbl[ptcb->OSTCBY] &= ~ptcb->OSTCBBitX) == 0u) {
                    OSRdyGrp &= ~ptcb->OSTCBBitY;
                }
            }
            ptcb = ptcb->OSTCBNext; /* 移至下一個任務 */
        }
    }

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

  - 原始設計為在 task_function 中檢查 InitalArriveTime，但會有低優先級任務來不及移除 RdyList 導致訊息印錯的狀況
  - InitalArriveTime != 0 的情況，OS_Start()後第一次呼叫 OS_SchedNew()搜尋 OSTCBList，如果 InitalArriveTime != 0，將其從 RdyList 中移除，並設定其 OSTCBDly

- os_core.c(OS_TimeTick):

```c
#if PROTOCOL == NPCS

            if (ptcb->OSTCBExtPtr != (void*)0) {
                task_para_set* p_task_data = (task_para_set*)(ptcb->OSTCBExtPtr);
                if (ptcb->OSTCBStat == OS_STAT_RDY && ptcb != OSTCBCur && p_task_data->RemainTime > 0 && (currentTime > p_task_data->InitArriveTime)
                    if (OSPrioCur != 63) {
                        // Blocking Time
                        if (ptcb->OSTCBPrio < OSTCBCur->OSTCBPrio) {
                            p_task_data->BlockTime++;
                            printf("%d\ttask(%2d)\tblockTime +1\n", OSTimeGet(), p_task_data->TaskID);
                        }
                        // Preemption Time
                        else {
                            p_task_data->PreemptiveTime++;
                            printf("%d\ttask(%2d)\tpreemptiveTime +1\n", OSTimeGet(), p_task_data->TaskID);
                        }
                    }
                }
            }
            ptcb = ptcb->OSTCBNext;
```

  - 搜尋 OSTCBList 當 ptcb 指向的 task 的 Priority 小於正在執行的 taskPrio 且 RemainTime>0(代表還沒做完)
    - 任務被 Block : BlockTime++
  - 搜尋 OSTCBList 當 ptcb 指向的 task 的 Priority 大於正在執行的 taskPrio 且 RemainTime>0(代表還沒做完)
    - 任務被 Preempt : PreemptTime++

[PART II] CPP Implementation

Objective:

Implement the ceiling- priority protocol (CPP) based on the RM scheduler in uC/OS-II.

- main.c(main_function) :

```c
#elif PROTOCOL == CPP
    INT8U R1_Prio = 255;
    INT8U R2_Prio = 255;
    for (int i = 0; i < TASK_NUMBER; i++) {
        TaskParameter[i].TaskPriority = 3*(i+1);
    }

    for (int i = 0; i < TASK_NUMBER; i++) {
        // 檢查該 Task 是否使用 R1
        if (TaskParameter[i].R1Lock < TaskParameter[i].R1UnLock) {
            if (TaskParameter[i].TaskPriority < R1_Prio) {
                R1_Prio = TaskParameter[i].TaskPriority;
            }
        }
        // 檢查該 Task 是否使用 R2
        if (TaskParameter[i].R2Lock < TaskParameter[i].R2UnLock) {
            if (TaskParameter[i].TaskPriority < R2_Prio) {
                R2_Prio = TaskParameter[i].TaskPriority;
            }
        }
    }

    R1_Ceiling = R1_Prio - 1;
    R2_Ceiling = R2_Prio - 2;

    R1 = OSMutexCreate(R1_Ceiling, &err);
    R2 = OSMutexCreate(R2_Ceiling, &err);

    for (int n = 0; n < TASK_NUMBER; n++) {
        Task_STK[n] = malloc(TASK_STACKSIZE * sizeof(int));
        OSTaskCreateExt(task,
            &TaskParameter[n],
            &Task_STK[n][TASK_STACKSIZE - 1],
            TaskParameter[n].TaskPriority,
            TaskParameter[n].TaskID,
            &Task_STK[n][0],
            TASK_STACKSIZE,
            &TaskParameter[n],
            (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR));
    }
#endif
```

- 在 CPP Protocol 中，當任務由 Periodic 由低排到高後(RM)，將任務的 Priority 設為 3*(i+1)，需要留 Priority 給 R1、R2
  - Task priority : 3、6、9...
- 檢查是否有用到 R1、R2，如果有，先將 R1、R2 Prio 設為當前 Task 的 Priority
  - 將 R1、R2 的 Prio – index 即為 R1、R2 的 Ceiling Priority
- 建立 Mutex 以及 periodic 任務

- main.c(task_function) :

```c
while (stopAlltask == 0) {
    task_data->Ready = 1;
    while (task_data->RemainTime > 0 && stopAlltask == 0) {
        if (stopAlltask == 1) {
            break;
        }

        executed_time = task_data->TaskExecutionTime - task_data->RemainTime;
        // 檢查 R1 Lock
        if ((executed_time == task_data->R1Lock) && (task_data->R1UnLock != 0)) {
#if PROTOCOL == NPCS
            LOG_print(3, "./Output.txt", "%d\tLockResource\ttask(%2d)(%2d)\t%s\n",
                OSTime, task_data->TaskID, task_data->job_no, "R1");
            OSSchedLock(); // 禁止搶佔
#elif PROTOCOL == CPP
            current_prio_log = base_prio;
            if (executed_time > task_data->R2Lock && executed_time < task_data->R2UnLock) {
                if (R2_Ceiling < current_prio_log) current_prio_log = R2_Ceiling;
            }
            target_prio_log = (R1_Ceiling < current_prio_log) ? R1_Ceiling : current_prio_log;
            LOG_print(3, "./Output.txt", "%d\tLockResource\ttask(%2d)(%2d)\t%s\t%d to %d\n",
                OSTime, task_data->TaskID, task_data->job_no, "R1", current_prio_log, target_prio_log);

            OSMutexPend(R1, 0, &err);
#endif
```

```c
        // 檢查 R2 Lock
        if ((executed_time == task_data->R2Lock) && (task_data->R2UnLock != 0)) {

#if PROTOCOL == NPCS
            LOG_print(3, "./Output.txt", "%d\tLockResource\ttask(%2d)(%2d)\t%s\n",
                OSTime, task_data->TaskID, task_data->job_no, "R2");
            OSSchedLock(); // 禁止搶佔
#elif PROTOCOL == CPP
            // Lock R2 前，檢查是否持有 R1
            current_prio_log = base_prio;
            if (executed_time > task_data->R1Lock && executed_time < task_data->R1UnLock) {
                if (R1_Ceiling < current_prio_log) current_prio_log = R1_Ceiling;
            }
            // Lock R2 後，再跟 R2 Ceiling 比
            target_prio_log = (R2_Ceiling < current_prio_log) ? R2_Ceiling : current_prio_log;
            LOG_print(3, "./Output.txt", "%d\tLockResource\ttask(%2d)(%2d)\t%s\t%d to %d\n",
                OSTime, task_data->TaskID, task_data->job_no, "R2", current_prio_log, target_prio_log);

            OSMutexPend(R2, 0, &err);
#endif
```

```c
        // 檢查 R1 Unlock
        if ((executed_time == task_data->R1UnLock) && (task_data->R1UnLock != 0)) {

#if PROTOCOL == NPCS
            LOG_print(3, "./Output.txt", "%d\tUnlockResource\ttask(%2d)(%2d)\t%s\t\n",
                OSTime, task_data->TaskID, task_data->job_no, "R1");
            OSSchedUnlock(); // 恢復排程
#elif PROTOCOL == CPP
            // 檢查是否持有 R2
            current_prio_log = base_prio;
            if (R1_Ceiling < current_prio_log) current_prio_log = R1_Ceiling; // 因為正在解鎖 R1，肯定持有它
            if (executed_time > task_data->R2Lock && executed_time < task_data->R2UnLock) {
                if (R2_Ceiling < current_prio_log) current_prio_log = R2_Ceiling;
            }
            // Unlock 後，R1 移除，檢查 R2
            target_prio_log = base_prio;
            if (executed_time > task_data->R2Lock && executed_time < task_data->R2UnLock) {
                if (R2_Ceiling < target_prio_log) target_prio_log = R2_Ceiling;
            }
            LOG_print(3, "./Output.txt", "%d\tUnlockResource\ttask(%2d)(%2d)\t%s\t%d to %d\n",
                OSTime, task_data->TaskID, task_data->job_no, "R1", current_prio_log, target_prio_log);

            OSMutexPost(R1);
#endif
        }
```

```c
        // 檢查 R2 Unlock
        if ((executed_time == task_data->R2UnLock) && (task_data->R2UnLock != 0)) {

#if PROTOCOL == NPCS
            LOG_print(3, "./Output.txt", "%d\tUnlockResource\ttask(%2d)(%2d)\t%s\n",
                OSTime, task_data->TaskID, task_data->job_no, "R2");
            OSSchedUnlock(); // 恢復排程
#elif PROTOCOL == CPP
            // 檢查 R1
            current_prio_log = base_prio;
            if (R2_Ceiling < current_prio_log) current_prio_log = R2_Ceiling; // 肯定持有 R2
            if (executed_time > task_data->R1Lock && executed_time < task_data->R1UnLock) {
                if (R1_Ceiling < current_prio_log) current_prio_log = R1_Ceiling;
            }
            // Unlock 後，R2 移除 檢查有沒有 R1
            target_prio_log = base_prio;
            if (executed_time > task_data->R1Lock && executed_time < task_data->R1UnLock) {
                if (R1_Ceiling < target_prio_log) target_prio_log = R1_Ceiling;
            }
            LOG_print(3, "./Output.txt", "%d\tUnlockResource\ttask(%2d)(%2d)\t%s\t%d to %d\n",
                OSTime, task_data->TaskID, task_data->job_no, "R2", current_prio_log, target_prio_log);

            OSMutexPost(R2);
#endif
```

- 變數定義:
    - base_prio: Task 本身的原始優先權
    - current_prio_log: 動作前的當下優先權
    - target_prio_log: 動作後的目標優先權
    - Ceiling: 資源的最高優先權上限

- Lock Resource： 準備鎖定 Resource 時，優先權會從目前的狀態提升到包含 Resource Ceiling 的狀態
    - 計算 Current：檢查目前是否已經持有其他資源->若有，Current = max_prio(Base, R2_Ceiling)
    - 計算 Target：比較 Current 與 R1 的 Ceiling ->Target = max_prio(Current, R1_Ceiling)
- Unlock Resource： 準備釋放 R1 時，優先權會從目前的高優先權降回剩餘資源的最高 Ceiling 或是原始優先權
    - 計算 Current：因為正在持有 R1（且可能持有 R2），Current 是所有持有資源中最高的 Ceiling
    - 計算 Target：釋放 R1 後，回歸 Base，再檢查是否還持有 R2。若還持有 R2，Target = max_prio(Base, R2_Ceiling)

os_core.c(OS_TimeTick) :

```c
#elif PROTOCOL == CPP
                if (ptcb->OSTCBExtPtr != (void*)0) {
                    task_para_set* p_task_data = (task_para_set*)(ptcb->OSTCBExtPtr);
                    if (ptcb != OSTCBCur && p_task_data->RemainTime > 0 &&
                        (currentTime > p_task_data->InitArriveTime) &&
                        p_task_data->Ready == 1 &&
                        currentTime > p_task_data->TaskArriveTime) {

                        task_para_set* p_running_data = (task_para_set*)0;
                        if (OSTCBCur->OSTCBExtPtr != (void*)0) {
                            p_running_data = (task_para_set*)(OSTCBCur->OSTCBExtPtr);
                        }

                        if (p_running_data != (task_para_set*)0) {
                            INT8U my_base_prio = p_task_data->TaskPriority;
                            INT8U running_base_prio = p_running_data->TaskPriority;
                            if (ptcb->OSTCBStat == OS_STAT_RDY) {
                                //  Blocking
                                if (running_base_prio > my_base_prio) {
                                    p_task_data->BlockTime++;
                                    printf("%d\ttask(%2d)\tblockTime +1 (Prio Inv)\n", OSTimeGet(), p_task_data->TaskID);
                                }
                                //  Preemption
                                else {
                                    p_task_data->PreemptiveTime++;
                                    printf("%d\ttask(%2d)\tpreemptiveTime +1\n", OSTimeGet(), p_task_data->TaskID);
                                }
                            }
                        }
                    }
                }
    #endif
                ptcb = ptcb->OSTCBNext;                     /* Point at next TCB in TCB list              */
```

- 判斷 Block Time 與 PreemptiveTime
    - Running_Base_Prio < My_Base_Prio (執行任務的原始優先級比我低)。代表執行任務是因為持有資源，優先權被暫時提升才擋住=>這屬於優先權反轉，計入 BlockiTime。
    - Running_Base_Prio > My_Base_Prio (執行任務的原始優先級本來就比較高)。代表這是正常的排程搶佔，計入 Preemptive Time。

- os_mutex.c(OSMutexPend):

```c
    OS_ENTER_CRITICAL();
    pcp = (INT8U)(pevent->OSEventCnt >> 8u);


    if ((INT8U)(pevent->OSEventCnt & OS_MUTEX_KEEP_LOWER_8) == OS_MUTEX_AVAILABLE) {
        pevent->OSEventCnt &= OS_MUTEX_KEEP_UPPER_8;
        pevent->OSEventCnt |= OSTCBCur->OSTCBPrio;
        pevent->OSEventPtr = (void*)OSTCBCur;
        if (pcp != OS_PRIO_MUTEX_CEIL_DIS) {
            if (OSTCBCur->OSTCBPrio > pcp) {
                if ((OSRdyTbl[OSTCBCur->OSTCBY] & OSTCBCur->OSTCBBitX) != 0) {
                    OSRdyTbl[OSTCBCur->OSTCBY] &= (OS_PRIO)~OSTCBCur->OSTCBBitX;
                    if (OSRdyTbl[OSTCBCur->OSTCBY] == 0) {
                        OSRdyGrp &= (OS_PRIO)~OSTCBCur->OSTCBBitY;
                    }
                }
                OSTCBCur->OSTCBPrio = pcp;


#if OS_LOWEST_PRIO <= 63u
                OSTCBCur->OSTCBY = (INT8U)(OSTCBCur->OSTCBPrio >> 3u);
                OSTCBCur->OSTCBX = (INT8U)(OSTCBCur->OSTCBPrio & 0x07u);
#else
                OSTCBCur->OSTCBY = (INT8U)((INT8U)(OSTCBCur->OSTCBPrio >> 4u) & 0xFFu);
                OSTCBCur->OSTCBX = (INT8U)(OSTCBCur->OSTCBPrio & 0x0Fu);
#endif
                OSTCBCur->OSTCBBitY = (OS_PRIO)(1uL << OSTCBCur->OSTCBY);
                OSTCBCur->OSTCBBitX = (OS_PRIO)(1uL << OSTCBCur->OSTCBX);
                OSRdyGrp |= OSTCBCur->OSTCBBitY;
                OSRdyTbl[OSTCBCur->OSTCBY] |= OSTCBCur->OSTCBBitX;

                OSTCBPrioTbl[pcp] = OSTCBCur;
            }
        }
//      if (pcp != OS_PRIO_MUTEX_CEIL_DIS) {
//          mprio = (INT8U)(pevent->OSEventCnt & OS_MUTEX_KEEP_LOWER_8); /*  Get priority of mutex owner   */
//          ptcb = (OS_TCB*)(pevent->OSEventPtr);                        /*     Point to TCB of mutex owner   */
//          if (ptcb->OSTCBPrio > pcp) {                                  /*     Need to promote prio of owner?*/
//              if (mprio > OSTCBCur->OSTCBPrio) {
//                  y = ptcb->OSTCBY;
```

  - 獲取 Mutex 時立即提升優先級
    - 成功獲取 Mutex 的當下，強制將當前任務的優先級提升至該 Mutex 定義的 Ceiling Priority，確保任務以最高優先級運行
  - 原有的優先級繼承 (Priority Inheritance) 偵測。由於任務持有資源時已處於最高優先級，因此不再需要當 Mutex 發生競爭時才動態提升優先級的邏輯。

- os_mutex.c(OSMutexPost) :

```c
INT8U  OSMutexPost(OS_EVENT* pevent)
{
    INT8U       pcp;                                    /* Priority ceiling priority                     */
    INT8U       prio;
    BOOLEAN     prio_restored = OS_FALSE;               /* [新增] 標記是否發生了優先權還原                  */
#if OS_CRITICAL_METHOD == 3u                            /* Allocate storage for CPU status register      */
    OS_CPU_SR  cpu_sr = 0u;
#endif

    if (OSIntNesting > 0u) {                            /* See if called from ISR ...                    */
        return (OS_ERR_POST_ISR);                       /* ... can't POST mutex from an ISR              */
    }
#if OS_ARG_CHK_EN > 0u
    if (pevent == (OS_EVENT*)0) {                      /* Validate 'pevent'                             */
        return (OS_ERR_PEVENT_NULL);
    }
#endif

    OS_TRACE_MUTEX_POST_ENTER(pevent);

    if (pevent->OSEventType != OS_EVENT_TYPE_MUTEX) {   /* Validate event block type                     */
        OS_TRACE_MUTEX_POST_EXIT(OS_ERR_EVENT_TYPE);
        return (OS_ERR_EVENT_TYPE);
    }
    OS_ENTER_CRITICAL();
    pcp = (INT8U)(pevent->OSEventCnt >> 8u);           /* Get priority ceiling priority of mutex        */
    prio = (INT8U)(pevent->OSEventCnt & OS_MUTEX_KEEP_LOWER_8);  /* Get owner's original priority      */
    if (OSTCBCur != (OS_TCB*)pevent->OSEventPtr) {     /* See if posting task owns the MUTEX            */
        OS_EXIT_CRITICAL();
        OS_TRACE_MUTEX_POST_EXIT(OS_ERR_NOT_MUTEX_OWNER);
        return (OS_ERR_NOT_MUTEX_OWNER);
    }
    if (pcp != OS_PRIO_MUTEX_CEIL_DIS) {
        if (OSTCBCur->OSTCBPrio == pcp) {
            OS_TRACE_MUTEX_TASK_PRIO_DISINHERIT(OSTCBCur, prio);
            OSMutex_RdyAtPrio(OSTCBCur, prio);         /* Restore the task's original priority          */

            prio_restored = OS_TRUE;                   /* 降低了自己的優先權                 */
        }
        OSTCBPrioTbl[pcp] = OS_TCB_RESERVED;           /* Reserve table entry                           */
    }
    if (pevent->OSEventGrp != 0u) {
```

```c
        OSTCBPrioTbl[pcp] = OS_TCB_RESERVED;          /* Reserve table entry                   */
    }
    if (pevent->OSEventGrp != 0u) {

        prio                = OS_EventTaskRdy(pevent, (void*)0, OS_STAT_MUTEX, OS_STAT_PEND_OK);
        pevent->OSEventCnt &= OS_MUTEX_KEEP_UPPER_8;
        pevent->OSEventCnt |= prio;
        pevent->OSEventPtr = OSTCBPrioTbl[prio];
        if ((pcp != OS_PRIO_MUTEX_CEIL_DIS) && (prio <= pcp)) {
            OS_EXIT_CRITICAL();
            OS_Sched();
            OS_TRACE_MUTEX_POST_EXIT(OS_ERR_PCP_LOWER);
            return (OS_ERR_PCP_LOWER);
        }
        else {
            OS_EXIT_CRITICAL();
            OS_Sched();                                    /* Find highest priority task ready to run   */
            OS_TRACE_MUTEX_POST_EXIT(OS_ERR_NONE);
            return (OS_ERR_NONE);
        }
    }
    pevent->OSEventCnt |= OS_MUTEX_AVAILABLE;
    pevent->OSEventPtr = (void*)0;
    OS_EXIT_CRITICAL();
    if (prio_restored == OS_TRUE) {
        OS_Sched();
    }

    OS_TRACE_MUTEX_POST_EXIT(OS_ERR_NONE);
    return (OS_ERR_NONE);
}
```

- 新增變數 prio_restored 來追蹤優先級變化
  - 修改內容：宣告了一個 BOOLEAN prio_restored = OS_FALSE;。當檢測到任務需要將優先級從 Ceiling Priority 還原回原始優先級時，將此旗標設為 OS_TRUE
- 釋放 Mutex 後觸發排程 (OS_Sched)
  - 當你優先級降低時，可能有其他沒有在等待此 Mutex，但優先級比原始優先級高的任務（原本被 CPP 壓制的任務）現在應該要執行了。 因此，必須加入 OS_Sched()，確保優先級降低後，系統能立即將 CPU 交給更高優先級的就緒任務。

[PART III] Performance Analysis [10%]

- Explain the difference between NPCS and CPP, and how each mechanism avoids the deadlock problem.
  - Describe how it works、Advantage、Disadvantage

CPP(Ceiling Priority Protocol):

- 允許搶佔，但受到嚴格限制。當任務進入 Critical Section 時，會立即將優先級提升至該資源的 Ceiling Priority。這只會阻止會使用該資源或優先級較低的任務進行搶佔，但仍允許那些不使用該資源且優先級更高的任務搶佔，保留了系統的反應能力
- 如何避免 Deadlock :
  - 由於任務一旦取得資源，優先級就會被提升至最高潛在競爭者的等級，系統保證了當前任務在持有資源期間，不會被任何可能請求同一資源的任務搶佔，確保任務能順利執行完畢並釋放資源，不會卡在中間等待更高優先級的任務釋放資源
- Advantage:
  - 允許優先級高於 Ceiling Priority(即不使用該資源) 的任務進行搶佔。這確保了系統能對緊急且不衝突的事件做出即時反應

        - 高優先級任務在執行過程中，最多只會被阻塞一次，避免了連鎖阻塞 Transitive blocking 的發生。
    - Disadvantages :
        - 動態調整優先級並操作 Ready Table，程式邏輯較為繁瑣
        - 涉及優先級計算與 Ready Queue 的改變，CPU 使用週期高於 NPCS
- NPCS(non-preemptible critical section) :
    - 將 Critical Section 視為不可分割的。透過關閉中斷或鎖住排程器，禁止任何 Context Switch。這意味著執行中的任務不能被任何其他任務搶佔，無論對方的優先級多高
    - 如何避免 Deadlock:
        - 因為持有資源的任務獨佔 CPU，其他任務無法執行，無法形成資源依賴的循環，直到當前任務釋放資源為止
    - Advantage :
        - 邏輯簡單，不需要複雜的優先級計算或修改 OS 核心
        - 進入與退出僅較少的 CPU 指令，減少系統時間
    - Disadvantage :
        - 它會阻塞所有任務，包含那些完全不需要此資源的最高優先級緊急任務。在 Real-time 系統中，這可能導致危急任務無法及時執行
        - 中斷延遲 ：若實作方式為關閉中斷，系統在 Critical Section 期間將無法響應硬體訊號。