

# 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`:

```
66  #include <string.h>
67  #define SYSTEM_END_TIME 100
68
69  #define INPUT_FILE_NAME "./TaskSet.txt"
70  #define OUTPUT_FILE_NAME "./output.txt"
71  #define MAX 20
72  #define INFO 10
73  #define RM 0
74  #define FIFO 1
75  #define ALGORITHM RM
76  #define NPCS 0
77  #define CPP 1
78  #define PROTOCOL NPCS
79
80  #define MAX_PRINTF_BUFFER 16
81
82  typedef struct task_para_set {
83      INT16U TaskID;
84      INT16U TaskArriveTime;
85      INT16U TaskExecutionTime;
86      INT16U TaskPeriodic;
87      INT16U TaskNumber;
88      INT16U TaskPriority;
89      INT16U InitArriveTime;
90      INT16U Ready;
91      INT16U job_no;           //Job number
92      INT16U RemainTime;     //任務剩餘時間
93      INT16U deadline;       //任務deadline
94      INT16U ResponseTime;   //紀錄response time
95      INT16U Delay;          //任務需要被OSTimeDly多久
96      INT16U BlockTime;
97      INT16U PreemptiveTime;
98      INT16U R1Lock;
99      INT16U R1UnLock;
100     INT16U R2Lock;
101     INT16U R2UnLock;
102 } 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` : `Resource1` 被 `Lock`、`Unlock` 時間
  - `R2Lock`、`R2UnLock` : `Resource2` 被 `Lock`、`Unlock` 時間
- Define 使用的 PROCOOL
  - NPCS(non-preemptible critical section)
  - CPP(Ceiling Priority Protocol)

- app\_hooks.h :

```

105     while (ptr != NULL) {
106         TaskInfo[i] = atoi(ptr);
107         ptr = strtok_s(NULL, " ", &pTmp);
108         if (i == 0) {
109             TASK_NUMBER++;
110             TaskParameter[j].TaskID = TASK_NUMBER;
111         }
112         else if (i == 1) {
113             TaskParameter[j].InitArriveTime = TaskInfo[i];
114             TaskParameter[j].TaskArriveTime = TaskInfo[i];
115         }
116         else if (i == 2) {
117             TaskParameter[j].TaskExecutionTime = TaskInfo[i];
118             TaskParameter[j].RemainTime = TaskInfo[i];
119         }
120         else if (i == 3) {
121             TaskParameter[j].TaskPeriodic = TaskInfo[i];
122         }
123         else if (i == 4)
124             TaskParameter[j].R1Lock = TaskInfo[i];
125         else if (i == 5)
126             TaskParameter[j].R1Unlock = TaskInfo[i];
127         else if (i == 6)
128             TaskParameter[j].R2Lock = TaskInfo[i];
129         else if (i == 7)
130             TaskParameter[j].R2Unlock = TaskInfo[i];
131         TaskParameter[j].deadline = TaskParameter[j].TaskArriveTime + TaskParameter[j].TaskPeriodic;
132         TaskParameter[j].BlockTime = 0;
133         TaskParameter[j].PreemptiveTime = 0;
134         TaskParameter[j].Ready = 0;
135         i++;
136     }
137     TaskParameter[j].TaskPriority = j;
138     j++;
139 }
140 }
```

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

- App\_hooks.h(App\_TimeTickHook) :

```

369     void App_TimeTickHook (void)
370     {
371 #if (APP_CFG_PROBE_OS_PLUGIN_EN == DEF_ENABLED) && (OS_PROBE_HOOKS_EN > 0)
372     OSProbe_TickHook();
373 #endif
374
375     OS_TCB* ptcb;
376     task_para_set* pdata;
377     INT32U CurrentTime;
378     // 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
379     CurrentTime = OSTime + 1;
380     ptcb = OSTCBLlist;
381     while (ptcb != (OS_TCB*)0) {
382         if (ptcb->OSTCBExtPtr != (void*)0) {
383             pdata = (task_para_set*)ptcb->OSTCBExtPtr;
384             if (CurrentTime >= pdata->InitArriveTime) {
385                 INT32U time_diff = CurrentTime - pdata->InitArriveTime;
386
387                 if (pdata->TaskPeriodic > 0 && (time_diff % pdata->TaskPeriodic) == 0) {
388                     pdata->RemainTime = pdata->TaskExecutionTime;
389                     pdata->Ready = 1;
390                 }
391             }
392         }
393         // 檢查下一個 TCB
394         ptcb = ptcb->OSTCBNext;
395     }
}
```

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

➤ 檢查整個 OSTCBLlist

- main.c(task\_function) :

```

110     void task(void* p_arg) {
111         task_para_set* task_data = (task_para_set*)p_arg;
112         INT32U current_time;
113         INT32U executed_time;
114         INT8U err;
115         INT8U base_prio = task_data->TaskPriority;
116         INT8U current_prio_log;
117         INT8U target_prio_log;
118         current_time = OSTimeGet();
119         while (stopAlltask == 0) {
120             task_data->Ready = 1;
121             while (task_data->RemainTime > 0 && stopAlltask == 0) {
122                 if (stopAlltask == 1) {
123                     break;
124                 }
125
126                 executed_time = task_data->TaskExecutionTime - task_data->RemainTime;
127                 // 檢查 R1 Lock
128                 if ((executed_time == task_data->R1Lock) && (task_data->R1UnLock != 0)) {
129 #if PROTOCOL == NPCS
130                     LOG_print(3, "./Output.txt", "%d\tLockResource\ttask(%2d)(%2d)\t%s\n",
131                             OSTime, task_data->TaskID, task_data->job_no, "R1");
132                     OSSchedLock(); // 禁止搶佔
133
134 // 檢查 R2 Lock
135                 if ((executed_time == task_data->R2Lock) && (task_data->R2UnLock != 0)) {
136 #if PROTOCOL == NPCS
137                     LOG_print(3, "./Output.txt", "%d\tLockResource\ttask(%2d)(%2d)\t%s\n",
138                             OSTime, task_data->TaskID, task_data->job_no, "R2");
139                     OSSchedLock(); // 禁止搶佔
140
141 // 檢查 R1 Unlock
142                 if ((executed_time == task_data->R1UnLock) && (task_data->R1UnLock != 0)) {
143 #if PROTOCOL == NPCS
144                     LOG_print(3, "./Output.txt", "%d\tUnlockResource\ttask(%2d)(%2d)\t%s\t\n",
145                         OSTime, task_data->TaskID, task_data->job_no, "R1");
146                     OSSchedUnlock(); // 恢復排程
147
148 // 檢查 R2 Unlock
149                 if ((executed_time == task_data->R2UnLock) && (task_data->R2UnLock != 0)) {
150 #if PROTOCOL == NPCS
151                     LOG_print(3, "./Output.txt", "%d\tUnlockResource\ttask(%2d)(%2d)\t%s\n",
152                         OSTime, task_data->TaskID, task_data->job_no, "R2");
153                     OSSchedUnlock(); // 恢復排程
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208

```

➤ 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) :

```

1979     static void OS_SchedNew(void)
1980     {
1981         #if OS_LOWEST_PRIO <= 63u                                /* See if we support up to 64 tasks */
1982         : INT8U y;
1983         static INT8U first_check = 0u;
1984
1985         if (first_check == 0u) {
1986             OS_TCB* ptcb;
1987             INT32U delay_ticks;
1988             first_check = 1u;
1989             ptcb = OSTCBList;
1990
1991             while (ptcb != (OS_TCB*)0 && (ptcb ->OSTCBExtPtr !=NULL)) {
1992                 delay_ticks = ((task_para_set*)(ptcb->OSTCBExtPtr)->InitArriveTime;
1993                 if (delay_ticks > 0) {
1994                     ptcb->OSTCBDly = delay_ticks;
1995                     // 從 Ready Table 中移除該任務
1996                     if ((OSRdyTbl[ptcb->OSTCBY] &= ~ptcb->OSTCBBitX) == 0u) {
1997                         OSRdyGrp &= ~ptcb->OSTCBBitY;
1998                     }
1999                 }
2000                 ptcb = ptcb->OSTCBNext; /* 移至下一個任務 */
2001             }
2002         }
2003         y = OSUnMapTbl[OSRdyGrp];
2004         OSPrhoHighRdy = (INT8U)(y << 3u) + OSUnMapTbl[OSRdyTbl[y]];

```

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

- os\_core.c(OS\_TimeTick):

```

#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 (OSPrhoCur != 63) {
                // Blocking Time
                if (ptcb->OSICBPrio < OSTCBCur->OSTCBPrio) {
                    p_task_data->blockTime++;
                    printf("%d\ttask(%d)\tblockTime +1\n", OSTimeGet(), p_task_data->TaskID);
                }
                // Preemption Time
                else {
                    p_task_data->PreemptiveTime++;
                    printf("%d\ttask(%d)\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) :

```
320     #elif PROTOCOL == CPP
321         INT8U R1_Prio = 255;
322         INT8U R2_Prio = 255;
323         for (int i = 0; i < TASK_NUMBER; i++) {
324             TaskParameter[i].TaskPriority = 3*(i+1);
325         }
326
327         for (int i = 0; i < TASK_NUMBER; i++) {
328             // 檢查該 Task 是否使用 R1
329             if (TaskParameter[i].R1Lock < TaskParameter[i].R1UnLock) {
330                 if (TaskParameter[i].TaskPriority < R1_Prio) {
331                     R1_Prio = TaskParameter[i].TaskPriority;
332                 }
333             }
334             // 檢查該 Task 是否使用 R2
335             if (TaskParameter[i].R2Lock < TaskParameter[i].R2UnLock) {
336                 if (TaskParameter[i].TaskPriority < R2_Prio) {
337                     R2_Prio = TaskParameter[i].TaskPriority;
338                 }
339             }
340         }
341
342         R1_Ceiling = R1_Prio - 1;
343         R2_Ceiling = R2_Prio - 2;
344
345         R1 = OSMutexCreate(R1_Ceiling, &err);
346         R2 = OSMutexCreate(R2_Ceiling, &err);
347
348         for (int n = 0; n < TASK_NUMBER; n++) {
349             Task_STK[n] = malloc(TASK_STACKSIZE * sizeof(int));
350             OTaskCreateExt(task,
351                             &TaskParameter[n],
352                             &Task_STK[n][TASK_STACKSIZE - 1],
353                             TaskParameter[n].TaskPriority,
354                             TaskParameter[n].TaskID,
355                             &Task_STK[n][0],
356                             TASK_STACKSIZE,
357                             &TaskParameter[n],
358                             (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR));
359         }
360     #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 紿 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) :

```

119
120     while (stopAlltask == 0) {
121         task_data->Ready = 1;
122         while (task_data->RemainTime > 0 && stopAlltask == 0) {
123             if (stopAlltask == 1) {
124                 break;
125             }
126
127             executed_time = task_data->TaskExecutionTime - task_data->RemainTime;
128             // 檢查 R1 Lock
129             if ((executed_time == task_data->R1Lock) && (task_data->R1UnLock != 0)) {
130                 LOG_print(3, "./Output.txt", "%d\tLockResource\ttask(%2d)(%2d)\t%s\n",
131                         OSTime, task_data->TaskID, task_data->job_no, "R1");
132                 OSSchedLock(); // 禁止搶佔
133             #elif PROTOCOL == CPP
134                 current_prio_log = base_prio;
135                 if (executed_time > task_data->R2Lock && executed_time < task_data->R2Unlock) {
136                     if (R2_Ceiling < current_prio_log) current_prio_log = R2_Ceiling;
137                 }
138
139                 target_prio_log = (R1_Ceiling < current_prio_log) ? R1_Ceiling : current_prio_log;
140                 LOG_print(3, "./Output.txt", "%d\tLockResource\ttask(%2d)(%2d)\t%s\n",
141                         OSTime, task_data->TaskID, task_data->job_no, "R1", current_prio_log, target_prio_log);
142
143                 OSMutexPend(R1, 0, &err);
144             #endif
145
146             // 檢查 R2 Lock
147             if ((executed_time == task_data->R2Lock) && (task_data->R2UnLock != 0)) {
148
149             #if PROTOCOL == NPCS
150                 LOG_print(3, "./Output.txt", "%d\tLockResource\ttask(%2d)(%2d)\t%s\n",
151                         OSTime, task_data->TaskID, task_data->job_no, "R2");
152                 OSSchedLock(); // 禁止搶佔
153             #elif PROTOCOL == CPP
154                 // Lock R2 前，檢查是否持有 R1
155                 current_prio_log = base_prio;
156                 if (executed_time > task_data->R1Lock && executed_time < task_data->R1Unlock) {
157                     if (R1_Ceiling < current_prio_log) current_prio_log = R1_Ceiling;
158                 }
159
160                 // Lock R2 後，再跟 R2 Ceiling 比
161                 target_prio_log = (R2_Ceiling < current_prio_log) ? R2_Ceiling : current_prio_log;
162                 LOG_print(3, "./Output.txt", "%d\tLockResource\ttask(%2d)(%2d)\t%s\t%d to %d\n",
163                         OSTime, task_data->TaskID, task_data->job_no, "R2", current_prio_log, target_prio_log);
164
165                 OSMutexPend(R2, 0, &err);
166             #endif
167
168             // 檢查 R1 Unlock
169             if ((executed_time == task_data->R1Unlock) && (task_data->R1UnLock != 0)) {
170
171             #if PROTOCOL == NPCS
172                 LOG_print(3, "./Output.txt", "%d\tUnlockResource\ttask(%2d)(%2d)\t%s\t%\n",
173                         OSTime, task_data->TaskID, task_data->job_no, "R1");
174                 OSSchedUnlock(); // 復復排程
175             #elif PROTOCOL == CPP
176                 // 檢查是否持有 R2
177                 current_prio_log = base_prio;
178                 if (R1_Ceiling < current_prio_log) current_prio_log = R1_Ceiling; // 因為正在解鎖 R1，肯定持有 R2
179                 if (executed_time > task_data->R2Lock && executed_time < task_data->R2Unlock) {
180                     if (R2_Ceiling < current_prio_log) current_prio_log = R2_Ceiling;
181                 }
182
183                 // Unlock 後，R1 移除，檢查 R2
184                 target_prio_log = base_prio;
185                 if (executed_time > task_data->R2Lock && executed_time < task_data->R2Unlock) {
186                     if (R2_Ceiling < target_prio_log) target_prio_log = R2_Ceiling;
187                 }
188                 LOG_print(3, "./Output.txt", "%d\tUnlockResource\ttask(%2d)(%2d)\t%s\t%d to %d\n",
189                         OSTime, task_data->TaskID, task_data->job_no, "R1", current_prio_log, target_prio_log);
190
191                 OSMutexPost(R1);
192             #endif
193
194             // 檢查 R2 Unlock
195             if ((executed_time == task_data->R2Unlock) && (task_data->R2UnLock != 0)) {
196
197             #if PROTOCOL == NPCS
198                 LOG_print(3, "./Output.txt", "%d\tUnlockResource\ttask(%2d)(%2d)\t%s\n",
199                         OSTime, task_data->TaskID, task_data->job_no, "R2");
200                 OSSchedUnlock(); // 復復排程
201             #elif PROTOCOL == CPP
202                 // 檢查 R1
203                 current_prio_log = base_prio;
204                 if (R2_Ceiling < current_prio_log) current_prio_log = R2_Ceiling; // 肯定持有 R2
205                 if (executed_time > task_data->R1Lock && executed_time < task_data->R1Unlock) {
206                     if (R1_Ceiling < current_prio_log) current_prio_log = R1_Ceiling;
207                 }
208
209                 // Unlock 後，R2 移除，檢查有沒有 R1
210                 target_prio_log = base_prio;
211                 if (executed_time > task_data->R1Lock && executed_time < task_data->R1Unlock) {
212                     if (R1_Ceiling < target_prio_log) target_prio_log = R1_Ceiling;
213                 }
214
215                 LOG_print(3, "./Output.txt", "%d\tUnlockResource\ttask(%2d)(%2d)\t%s\t%d to %d\n",
216                         OSTime, task_data->TaskID, task_data->job_no, "R2", current_prio_log, target_prio_log);
217
218                 OSMutexPost(R2);
219             #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) :

```

1110     v #elif PROTOCOL == CPP
1111     v if (ptcb->OSTCBExtPtr != (void*)0) {
1112         v     task_para_set* p_task_data = (task_para_set*)(ptcb->OSTCBExtPtr);
1113         v     if (ptcb != OSTCBCur && p_task_data->RemainTime > 0 &&
1114             (currentTime > p_task_data->InitArriveTime) &&
1115             p_task_data->Ready == 1 &&
1116             currentTime > p_task_data->TaskArriveTime) {
1117
1118         v         task_para_set* p_running_data = (task_para_set*)0;
1119         v         if (OSTCBCur->OSTCBExtPtr != (void*)0) {
1120             v             p_running_data = (task_para_set*)(OSTCBCur->OSTCBExtPtr);
1121         }
1122
1123         v         if (p_running_data != (task_para_set*)0) {
1124             v             INT8U my_base_prio = p_task_data->TaskPriority;
1125             v             INT8U running_base_prio = p_running_data->TaskPriority;
1126             v             if (ptcb->OSTCBStat == OS_STAT_RDY) {
1127                 // Blocking
1128                 v                 if (running_base_prio > my_base_prio) {
1129                     v                     p_task_data->BlockTime++;
1130                     v                     printf("%d\ttask(%d)\tblockTime +1 (Prio Inv)\n", OSTimeGet(), p_task_data->TaskID);
1131                 }
1132                 // Preemption
1133                 v                 else {
1134                     v                     p_task_data->PreemptiveTime++;
1135                     v                     printf("%d\ttask(%d)\tpreemptiveTime +1\n", OSTimeGet(), p_task_data->TaskID);
1136                 }
1137             }
1138         }
1139     }
1140 }
1141 #endif
1142         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 BlockTime。
- 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):

```

495     OS_ENTER_CRITICAL();
496     pcp = (INT8U)(pevent->OSEventCnt >> 8u);
497
498     if (((INT8U)(pevent->OSEventCnt & OS_MUTEX_KEEP_LOWER_8) == OS_MUTEX_AVAILABLE) {
499         if ((OSRdyTbl[OSTCBCur->OSTCBY] & OSTCBCur->OSTCBBitX) != 0) {
500             OSRdyTbl[OSTCBCur->OSTCBY] &= ~(OS_PRIO)-OSTCBCur->OSTCBBitX;
501             if (OSRdyTbl[OSTCBCur->OSTCBY] == 0) {
502                 OSRdyGrp &= ~(OS_PRIO)-OSTCBCur->OSTCBBitY;
503             }
504         }
505         if (OSTCBCur->OSTCBPrio > pcp) {
506             if ((OSRdyTbl[OSTCBCur->OSTCBY] & OSTCBCur->OSTCBBitY) != 0) {
507                 OSRdyTbl[OSTCBCur->OSTCBY] &= ~(OS_PRIO)-OSTCBCur->OSTCBBitX;
508                 if (OSRdyTbl[OSTCBCur->OSTCBY] == 0) {
509                     OSRdyGrp &= ~(OS_PRIO)-OSTCBCur->OSTCBBitY;
510                 }
511             }
512             OSTCBCur->OSTCBPrio = pcp;
513
514         #if OS_LOWEST_PRIO <= 63u
515             OSTCBCur->OSTCBY = (INT8U)(OSTCBCur->OSTCBPrio >> 3u);
516             OSTCBCur->OSTCBX = (INT8U)(OSTCBCur->OSTCBPrio & 0x07u);
517         #else
518             OSTCBCur->OSTCBY = (INT8U)((INT8U)(OSTCBCur->OSTCBPrio >> 4u) & 0xFFu);
519             OSTCBCur->OSTCBX = (INT8U)(OSTCBCur->OSTCBPrio & 0xFu);
520         #endif
521         OSTCBCur->OSTCBBitY = (OS_PRIO)(1uL << OSTCBCur->OSTCBY);
522         OSTCBCur->OSTCBBitX = (OS_PRIO)(1uL << OSTCBCur->OSTCBX);
523         OSRdyGrp |= OSTCBCur->OSTCBBitY;
524         OSRdyTbl[OSTCBCur->OSTCBY] |= OSTCBCur->OSTCBBitX;
525
526         OSTCBPrioTbl[pcp] = OSTCBCur;
527     }
528
529     // if (pcp != OS_PRIO_MUTEX_CEIL_DIS) {
530     //     mprio = (INT8U)(pevent->OSEventCnt & OS_MUTEX_KEEP_LOWER_8); /* Get priority of mutex owner */
531     //     ptcb = (OS_TCB*)(pevent->OSEventPtr); /* Point to TCB of mutex owner */
532     //     if (ptcb->OSTCBPrio > pcp) { /* Need to promote prio of owner */
533     //         if (mprio > OSTCBCur->OSTCBPrio) {
534     //             y = ptcb->OSTCBY;
535     //         }
536     //     }
537 }

```

➤ 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):

```

647     INT8U OSMutexPost(OS_EVENT* pevent)
648     {
649         INT8U pcp;                                /* Priority ceiling priority */          */
650         INT8U prio;                             /* CPU status register */                */
651         BOOLEAN prio_restored = OS_FALSE;        /* [新增] 標記是否發生了優先權還原 */    */
652         #if OS_CRITICAL_METHOD == 3u              /* Allocate storage for CPU status register */
653             OS_CPU_SR cpu_sr = 0u;
654         #endif
655
656         if (OSIntNesting > 0u) {                  /* See if called from ISR ... */        */
657             return (OS_ERR_POST_ISR);            /* ... can't POST mutex from an ISR */   */
658         }
659         #if OS_ARG_CHK_EN > 0u
660             if (pevent == (OS_EVENT*)0) {           /* Validate 'pevent' */                  */
661                 return (OS_ERR_PEVENT_NULL);
662             }
663         #endif
664
665         OS_TRACE_MUTEX_POST_ENTER(pevent);
666
667         if (pevent->OSEventType != OS_EVENT_TYPE_MUTEX) { /* Validate event block type */      */
668             OS_TRACE_MUTEX_POST_EXIT(OS_ERR_EVENT_TYPE);
669             return (OS_ERR_EVENT_TYPE);
670         }
671         OS_ENTER_CRITICAL();
672         pcp = (INT8U)(pevent->OSEventCnt >> 8u); /* Get priority ceiling priority of mutex */
673         prio = (INT8U)(pevent->OSEventCnt & OS_MUTEX_KEEP_LOWER_8); /* Get owner's original priority */
674         if (OSTCBCur != (OS_TCB*)pevent->OSEventPtr) { /* See if posting task owns the MUTEX */
675             OS_EXIT_CRITICAL();
676             OS_TRACE_MUTEX_POST_EXIT(OS_ERR_NOT_MUTEX_OWNER);
677             return (OS_ERR_NOT_MUTEX_OWNER);
678         }
679         if (pcp != OS_PRIO_MUTEX_CEIL_DIS) {
680             if (OSTCBCur->OSTCBPrio == pcp) {
681                 OS_TRACE_MUTEX_TASK_PRIO_DISINHERIT(OSTCBCur, prio);
682                 OSMutex_RdyAtPrio(OSTCBCur, prio); /* Restore the task's original priority */
683
684                 prio_restored = OS_TRUE;          /* 降低了自己的優先權 */               */
685             }
686             OSTCBPrioTbl[pcp] = OS_TCB_RESERVED; /* Reserve table entry */                */
687         }
688         if (pevent->OSEventGrp != 0u) {

```

```

686     OSTCBPrioTbl[pcp] = OS_TCB_RESERVED;           /* Reserve table entry */      */
687 }
688 if (pevent->OSEventGrp != 0) {
689
690     prio          = OS_EventTaskRdy(pevent, (void*)0, OS_STAT_MUTEX, OS_STAT_PEND_OK);
691     pevent->OSEventCnt &= OS_MUTEX_KEEP_UPPER_8;
692     pevent->OSEventCnt |= prio;
693     pevent->OSEventPtr = OSTCBPrioTbl[prio];
694     if ((pcp != OS_PRIO_MUTEX_CEIL_DIS) && (prio <= pcp)) {
695         OS_EXIT_CRITICAL();
696         OS_Sched();
697         OS_TRACE_MUTEX_POST_EXIT(OS_ERR_PCP_LOWER);
698         return (OS_ERR_PCP_LOWER);
699     }
700     else {
701         OS_EXIT_CRITICAL();
702         OS_Sched();                                /* Find highest priority task ready to run */
703         OS_TRACE_MUTEX_POST_EXIT(OS_ERR_NONE);
704         return (OS_ERR_NONE);
705     }
706 }
707 pevent->OSEventCnt |= OS_MUTEX_AVAILABLE;
708 pevent->OSEventPtr = (void*)0;
709 OS_EXIT_CRITICAL();
710 if (prio_restored == OS_TRUE) {
711     OS_Sched();
712 }
713 OS_TRACE_MUTEX_POST_EXIT(OS_ERR_NONE);
714 return (OS_ERR_NONE);
715
716 }

```

- 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。