

PART I**Task Control Block Linked List**

Following the previous homework (HW1), please add some code to the μ C/OS-II scheduler in the kernel level to observe the operations of the task control block (TCB) and TCB linked list.

The screenshot results. (10%)

```
D:\ntust\EmbeddingOS\RTOS_M11107309_PA1\RTOS_M1
OSTick    created, Thread ID 17316
Task[63] created, TCB address b2ffa0
-----After TCB[63] being linked-----
Previous TCB point to address      0
Current  TCB point to address      b2ffa0
Next     TCB point to address      0

Task[ 1] created, TCB address b2fffc
-----After TCB[ 1] being linked-----
Previous TCB point to address      0
Current  TCB point to address      b2fffc
Next     TCB point to address      b2ffa0

Task[ 2] created, TCB address b30058
-----After TCB[ 2] being linked-----
Previous TCB point to address      0
Current  TCB point to address      b30058
Next     TCB point to address      b2fffc

=====TCB linked list=====
Task   Prev_TCB_addr  TCB_addr  Next_TCB_addr
  2           0        b30058        b2fffc
  1       b30058        b2fffc        b2ffa0
 63       b2fffc        b2ffa0           0
```

A report that describes your implementation (please attach the screenshot of the code and MARK the modified part). (10%)

(如未標示，整段即為全新 code)

os_cpu.c

```
624
625  #if (OS_MSG_TRACE > 0u)
626      /// PA#1 part1
627      OS_Printf("Task[%2d] created, TCB address %6x\n", p_tcb->OSTCBPrio, p_tcb);
628  #endif
```

建立 task 時會進行 trace 並輸出訊息，將其更改為題目要求的 TCB address。

os_core.c

OSTCBInit()

```
2450      OSTCBList = ptcb;
2451
2452      /// PA#1 part1
2453      printf("-----After TCB[%2d] being linked-----\n", prio);
2454      printf("Previous TCB point to address\t%6x\n", ptcb->OSTCBPrev);
2455      printf("Current TCB point to address\t%6x\n", ptcb);
2456      printf("Next TCB point to address\t%6x\n\n", ptcb->OSTCBNext);
2457      /// PA#1 part1
2458
2459      OSRdyGrp |= ptcb->OSTCBBitY;          /* Make task ready to run
2460      OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
```

OSStart()

```
871  void OSStart (void)
872  {
873
874      if (OSRunning == OS_FALSE) {
875
876          OS_SchedNew();                      /* Find highest priority's task priority number */
877          OSPrioCur = OSPrioHighRdy;
878          OSTCBHighRdy = OSTCBPrioTbl[OSPrioHighRdy]; /* Point to highest priority task ready to run */
879          OSTCBCur = OSTCBHighRdy;
880
881          /// PA#1 part1
882          OS_TCB* ptcb;
883          ptcb = OSTCBList;
884
885          printf("=====TCB linked list=====\\n");
886          printf("Task Prev_TCB_addr TCB_addr Next_TCB_addr\\n");
887          while (ptcb->OSTCBPrio != OS_TASK_IDLE_PRIO) { /* Go through all TCBs in TCB list */
888              OS_ENTER_CRITICAL();
889              printf("%2d\t\t %6x\t %6x\t%6x\\n", ptcb->OSTCBPrio, ptcb->OSTCBPrev, ptcb, ptcb->OSTCBNext);
890              ptcb = ptcb->OSTCBNext; /* Point at next TCB in TCB list */
891              OS_EXIT_CRITICAL();
892          }
893          printf("%2d\t\t %6x\t %6x\t%6x\\n", ptcb->OSTCBPrio, ptcb->OSTCBPrev, ptcb, ptcb->OSTCBNext);
894          /// PA#1 part1
895      }
```

任務建立好後在 OSStart，呼叫 ptcb，輸出 TCB list，利用 ptcb->OSTCBNext 與 while，歷遍 TCB list。

PART II

RM Scheduler Implementation

To implement the Rate Monotonic (RM) scheduler for periodic tasks and observe the scheduling behaviors.

Periodic Task Set = $\{\tau_{ID} (ID, \text{arrival time}, \text{execution time}, \text{period})\}$

Example Task Set 1 = $\{\tau_1 (1, 1, 2, 4), \tau_2 (2, 0, 4, 10)\}$

Example Task Set 2 = $\{\tau_1 (1, 3, 4, 14), \tau_2 (2, 0, 2, 8), \tau_3 (3, 0, 4, 10), \tau_4 (4, 24, 2, 12)\}$

Example Task Set 3 = $\{\tau_1 (1, 2, 2, 10), \tau_2 (2, 1, 1, 5), \tau_3 (3, 0, 8, 15)\}$

The correctness of schedule results of examples. Note the testing task set might not be the same as the given example task set. (40%)

Task Set 1

Tick	Event	CreateTask ID	NextTask ID	ResponseTime	#of ContextSwitch	PreemptionTime	OSTimeDly
1	Preemption	task(2)(0)	task(1)(0)				
3	Completion	task(1)(0)	task(2)(0)	2	2	0	2
5	Preemption	task(2)(0)	task(1)(1)				
7	Completion	task(1)(1)	task(2)(0)	2	2	0	2
8	Completion	task(2)(0)	task(63)	8	5	4	2
9	Preemption	task(63)	task(1)(2)				
11	Completion	task(1)(2)	task(2)(1)	2	2	0	2
13	Preemption	task(2)(1)	task(1)(3)				
15	Completion	task(1)(3)	task(2)(1)	2	2	0	2
17	Completion	task(2)(1)	task(1)(4)	7	4	2/t	3
19	Completion	task(1)(4)	task(63)	2	2	0	2
20	Preemption	task(63)	task(2)(2)				
21	Preemption	task(2)(2)	task(1)(5)				
23	Completion	task(1)(5)	task(2)(2)	2	2	0	2
25	Preemption	task(2)(2)	task(1)(6)				
27	Completion	task(1)(6)	task(2)(2)	2	2	0	2
28	Completion	task(2)(2)	task(63)	8	6	4	2
29	Preemption	task(63)	task(1)(7)				

Task Set 2

Tick	Event	CreateTask ID	NextTask ID	ResponseTime	#of ContextSwitch	PreemptionTime	OSTimeDly
2	Completion	task(2)(0)	task(3)(0)	2	1	0	6
6	Completion	task(3)(0)	task(4)(0)	6	2	0	4
6	Completion	task(4)(0)	task(1)(0)	6	2	0	6
8	Preemption	task(1)(0)	task(2)(1)				
10	Completion	task(2)(1)	task(3)(1)	2	2	0	6
14	Completion	task(3)(1)	task(1)(0)	4	2	0	6
16	Completion	task(1)(0)	task(2)(2)	16	4	6	-2
18	Completion	task(2)(2)	task(1)(1)	2	2	0	6
20	Preemption	task(1)(1)	task(3)(2)				
24	Completion	task(3)(2)	task(2)(3)	4	2	0	6
26	Completion	task(2)(3)	task(4)(1)	2	2	0	6
28	Completion	task(4)(1)	task(1)(1)	4	2	0	8
30	Completion	task(1)(1)	task(3)(3)	13	4	8	1

Task Set 3

Tick	Event	CreateTask ID	NextTask ID	ResponseTime	#of ContextSwitch	PreemptionTime	OSTimeDly
1	Preemption	task(3)(0)	task(2)(0)				
2	Completion	task(2)(0)	task(1)(0)	1	2	0	4
4	Completion	task(1)(0)	task(3)(0)	2	2	0	8
6	Preemption	task(3)(0)	task(2)(1)				
7	Completion	task(2)(1)	task(3)(0)	1	2	0	4
11	Preemption	task(3)(0)	task(2)(2)				
12	Completion	task(2)(2)	task(1)(1)	1	2	0	4
14	Completion	task(1)(1)	task(3)(0)	2	2	0	8
15	Completion	task(3)(0)	task(3)(1)	15	6	7	0
16	Preemption	task(3)(1)	task(2)(3)				
17	Completion	task(2)(3)	task(3)(1)	1	2	0	4
21	Preemption	task(3)(1)	task(2)(4)				
22	Completion	task(2)(4)	task(1)(2)	1	2	0	4
24	Completion	task(1)(2)	task(3)(1)	2	2	0	8
26	Preemption	task(3)(1)	task(2)(5)				
27	Completion	task(2)(5)	task(3)(1)	1	2	0	4
28	Completion	task(3)(1)	task(63)	13	7	5	2
30	Preemption	task(63)	task(3)(2)				

A report that describes your implementation (please attach the screenshot of the code and MARK the modified part). (40%)

(如未標示，整段即為全新 code)

ucos ii.h

global variable

```
654  
655     /// PA#1 part2  
656     INT8U already_delay;  
657     INT8U remain_exe_time;  
658     INT8U next_job_time;  
659     INT8U self_continue;  
660     /// PA#1 part2  
661
```

建立全域變數，用於後面計算 response time。

Main function

```
113 // PA#1 part2
114 // RM scheduling rules : task with smallest time period will have highest priority
115 // insertion sort by TaskPeriodic
116 int i, j, key;
117 task_para_set tmp_TaskParameter;
118 for (j = 1; j < TASK_NUMBER; j++)
119 {
120     key = TaskParameter[j].TaskPeriodic;
121     tmp_TaskParameter = TaskParameter[j];
122     i = j - 1;
123     while (i >= 0 && TaskParameter[i].TaskPeriodic > key)
124     {
125         TaskParameter[i + 1] = TaskParameter[i];
126         i = i - 1;
127     }
128     TaskParameter[i + 1] = tmp_TaskParameter;
129 }
130
131 // create n tasks
132 for (n = 0; n < TASK_NUMBER; n++)
133 {
134     // printf("ID %d, prio %d\n", TaskParameter[n].TaskID, n+1);
135     OSTaskCreateExt(task1, // task function
136                     &TaskParameter[n], // p_arg(task function)
137                     &Task_STK[n][TASK_STACKSIZE - 1], // ptos
138                     n+1, // prio, PA#1 priority start from 1.
139                     TaskParameter[n].TaskID, // id
140                     &Task_STK[n][0], // pbos
141                     TASK_STACKSIZE, // stack size
142                     &TaskParameter[n], // pext(TCB extension pointer)
143                     (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR)); // opt
144 }
145 // PA#1 part2
```

依照題目要求，需要能夠根據輸入的任務數量，建立任務，所以利用 for loop 進行 n 個 task 的建立。

因為要進行 RM scheduling，所以必須將 task 的 priority 根據 period 進行排列(成反比)，因此針對輸入的 period 將任務重新排列。

Task function

```
172 void task1(void* p_arg) {
173     task_para_sct* task_data;
174     task_data = p_arg;
175
176     /// PA#1 part2
177     int job_ready_time = task_data->TaskArriveTime;
178     int ncxt_job_ready_time;
179     int cxc_time;
180     int job_start_time;
181
182     if (OSTimeGet() < task_data->TaskArriveTime)
183     {
184         OSTimeDly(task_data->TaskArriveTime - OSTimeGet());
185     }
186     while (1)
187     {
188         OSTCECur->self_continuc = 0;
189         ncxt_job_ready_time = job_ready_time + task_data->TaskPeriodic;
190         OSTCECur->ncxt_job_time = ncxt_job_ready_time;
191         cxc_time = task_data->TaskExecutionTime;
192         OSTCECur->remain_cxc_time = cxc_time;
193         job_start_time = OSTimeGet();
194
195         while (1)
196         {
197             if (OSTimeGet() != job_start_time)
198             {
199                 cxc_time--;
200                 OSTCECur->remain_cxc_time = cxc_time;
201                 ///printf("%2d\tTaskID %2d remain cxc_time : %2d\n", OSTimeGet(), task_data->TaskID, cxc_time);
202                 if (cxc_time == 0)
203                 {
204                     break;
205                 }
206                 job_start_time = OSTimeGet();
207             }
208         }
209
210         job_ready_time = ncxt_job_ready_time; /// Pre-update
211
212         if (OSTCECur->alrcady_dclay == 1)
213         {
214             OSTCECur->alrcady_dclay = 0;
215             continuoc;
216         }
217
218         int dclay_time = ncxt_job_ready_time - OSTimeGet();
219         if (dclay_time == 0)
220         {
221             OSTCECur->self_continuc = 1;
222             OS_Sched();
223         }
224         OSTimeDly(dclay_time);
225     }
226 }
227
228 /// PA#1 part2
```

根據給定的 arrive time, execution time, period 進行運作的 task funtion。execution 期間卡在 while 迴圈中不斷檢查執行時間是否結束,結束後則使用 OSTimeDly() 進入 waiting 狀態,直到下個周期開始。

os_core.c

OSStart()

```
895 |  
896 |  
897 |  
898 |  
899 |  
900 |  
901 |
```

```
    /// PA#1 part2  
    printf("\nTick\t Event\t\t\t CreateTask ID\t NextTask ID\t ResponseTime #of ContextSwitch PreemptionTime OSTimeDly\n");  
    /// PA#1 part2  
    OSStartHighRdy();  
    /* Execute target specific code to start task */
```

根據題目要求，輸出標題。

variable

```
959 |  
960 |  
961 |  
962 |  
963 |  
964 |  
965 |  
966 |  
967 |  
968 |  
969 |  
970 |  
971 |  
972 |
```

```
    /*  
    /// PA#1 part2  
    int job_number[64] = { 0 };  
    INT8U OSMaTbl[8] = { 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80 };  
    int task_start_time[64] = { 0 };  
    int task_preempt_count[64] = { 0 };  
    int task_preempt_time[64] = { 0 };  
    int task_preempt_time_acc[64] = { 0 };  
    /// PA#1 part2  
void OSTimeTick (void)  
{
```

建立變數，等會用於計算 response time、preemption time、job number 與 CtxSwitch number。

OSTimeTick()

```
1036 |  
1037 |  
1038 |  
1039 |  
1040 |  
1041 |  
1042 |  
1043 |  
1044 |  
1045 |  
1046 |  
1047 |
```

```
    if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) == OS_STAT_RDY) { /* Is task suspended? */  
        OSRdyGrp |= ptcb->OSTCBBitY; /* No, Make ready */  
        OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;  
    }  
    /// PA#1 part2  
    task_start_time[ptcb->OSTCBPrio] = OSTimeGet();  
    /// PA#1 part2  
    OS_TRACE_TASK_READY(ptcb);
```

在 Time tick 中，設定 task 的啟動時間。

OS_SchedNew()

```

1814
1815 // PA#1 part2
1816 if ((Output_err = fopen_s(&Output_fp, "../Output.txt", "a")) != 0)
1817 {
1818     printf("can't open Output.txt");
1819     exit(0);
1820 }
1821
1822 // =====special case completion=====
1823 // Next task and current task are same
1824 if (OSPrioCur == OSPrioHighRdy)
1825 {
1826     if (OSTCBCur->self_continue == 1)
1827     {
1828         printf("%2d\t", OSTimeGet());
1829         fprintf(Output_fp, "%2d\t", OSTimeGet());
1830
1831         printf(" Completion\t");
1832         fprintf(Output_fp, " Completion\t");
1833         // CurrentTask
1834         printf(" task(%2d)(%2d)", OSTCBPrioTbl[OSPrioCur]->OSTCBId, job_number[OSPrioCur]);
1835         fprintf(Output_fp, " task(%2d)(%2d)", OSTCBPrioTbl[OSPrioCur]->OSTCBId, job_number[OSPrioCur]);
1836         // NextTask
1837         job_number[OSPrioCur]++;
1838         if (OSPrioHighRdy == 63)
1839         {
1840             printf(" task(%2d) \t\t", OSPrioHighRdy);
1841             fprintf(Output_fp, " task(%2d) \t\t", OSPrioHighRdy);
1842         }
1843         else
1844         {
1845             printf(" task(%2d)(%2d)\t\t", OSTCBPrioTbl[OSPrioHighRdy]->OSTCBId, job_number[OSPrioHighRdy]);
1846             fprintf(Output_fp, " task(%2d)(%2d)\t\t", OSTCBPrioTbl[OSPrioHighRdy]->OSTCBId, job_number[OSPrioHighRdy]);
1847         }
1848         // ResponseTime
1849         printf("%4d\t\t", (OSTimeGet() - task_start_time[OSPrioCur]));
1850         fprintf(Output_fp, "%4d\t\t", (OSTimeGet() - task_start_time[OSPrioCur]));
1851         // # of ContextSwitch
1852         // task_preempt_count[OSPrioCur]++; // same task
1853         printf("%7d\t\t", task_preempt_count[OSPrioCur]);
1854         fprintf(Output_fp, "%7d\t\t", task_preempt_count[OSPrioCur]);
1855         // PreemptionTime
1856         printf("%4d\t", task_preempt_time_acc[OSPrioCur]);
1857         fprintf(Output_fp, "%4d\t", task_preempt_time_acc[OSPrioCur]);
1858         // DelayTime: Period - ResponseTime
1859         printf("%11d\n", (TaskParameter[OSPrioCur - 1].TaskPeriodic - (OSTimeGet() - task_start_time[OSPrioCur])));
1860         fprintf(Output_fp, "%11d\n", (TaskParameter[OSPrioCur - 1].TaskPeriodic - (OSTimeGet() - task_start_time[OSPrioCur])));
1861
1862         task_preempt_time[OSPrioCur] = 0;
1863         task_preempt_count[OSPrioCur] = 0;
1864         task_preempt_time_acc[OSPrioCur] = 0;
1865
1866         if (task_preempt_time[OSPrioHighRdy] != 0) // if the next task has been preempted
1867         {
1868             task_preempt_time_acc[OSPrioHighRdy] = task_preempt_time_acc[OSPrioHighRdy] + (OSTimeGet() - task_preempt_time[OSPrioHighRdy]);
1869         }
1870
1871         // task_preempt_count[OSPrioHighRdy]++; // same task
1872         task_start_time[OSTCBCur->OSTCBPrio] = OSTimeGet();
1873     }
1874 }
1875 // =====Special Case completion=====
1876
1877 else if ((OSPrioCur != OSPrioHighRdy && OSTimeGet() != 0))
1878 {
1879     if (OSPrioCur == 0)
1880     {
1881         task_start_time[OSPrioHighRdy] = OSTimeGet();
1882     }
1883     else
1884     {
1885         printf("%2d\t", OSTimeGet());
1886         fprintf(Output_fp, "%2d\t", OSTimeGet());
1887
1888         // =====Completion=====
1889         if ((OSRdyTbl[OSPrioCur >> 3] & OSMapTbl[OSPrioCur & 0x07]) == 0)
1890         {
1891             printf(" Completion\t");
1892             fprintf(Output_fp, " Completion\t");
1893             // CurrentTask
1894             printf(" task(%2d)(%2d)", OSTCBPrioTbl[OSPrioCur]->OSTCBId, job_number[OSPrioCur]);
1895             fprintf(Output_fp, " task(%2d)(%2d)", OSTCBPrioTbl[OSPrioCur]->OSTCBId, job_number[OSPrioCur]);
1896             // NextTask
1897             if (OSPrioHighRdy == 63)
1898             {
1899                 printf(" task(%2d) \t\t", OSPrioHighRdy);
1900                 fprintf(Output_fp, " task(%2d) \t\t", OSPrioHighRdy);
1901             }

```



```

1902 else
1903 {
1904     printf(" task(%2d)(%2d)\t\t", OSTCBPrioTbl[OSPrioHighRdy]->OSTCBId, job_number(OSPrioHighRdy));
1905     fprintf(Output_fp, " task(%2d)(%2d)\t\t", OSTCBPrioTbl[OSPrioHighRdy]->OSTCBId, job_number(OSPrioHighRdy));
1906 }
1907
1908 // ResponseTime
1909 printf("%4d\t\t", (OSTimeGet() - task_start_time[OSPrioCur]));
1910 fprintf(Output_fp, "%4d\t\t", (OSTimeGet() - task_start_time[OSPrioCur]));
1911 // # of ContextSwitch
1912 task_preempt_count[OSPrioCur]++;
1913 printf("%7d\t\t", task_preempt_count[OSPrioCur]);
1914 fprintf(Output_fp, "%7d\t\t", task_preempt_count[OSPrioCur]);
1915 // PreemptionTime
1916 printf("%4d\t", task_preempt_time_acc[OSPrioCur]);
1917 fprintf(Output_fp, "%4d\t", task_preempt_time_acc[OSPrioCur]);
1918 // DelayTime: Period - ResponseTime
1919 printf("%11d\n", (TaskParameter[OSPrioCur - 1].TaskPeriodic - (OSTimeGet() - task_start_time[OSPrioCur])));
1920 fprintf(Output_fp, "%11d\n", (TaskParameter[OSPrioCur - 1].TaskPeriodic - (OSTimeGet() - task_start_time[OSPrioCur])));
1921
1922 task_preempt_time[OSPrioCur] = 0;
1923 task_preempt_count[OSPrioCur] = 0;
1924 task_preempt_time_acc[OSPrioCur] = 0;
1925
1926 if (task_preempt_time[OSPrioHighRdy] != 0)
1927 {
1928     task_preempt_time_acc[OSPrioHighRdy] = task_preempt_time_acc[OSPrioHighRdy] + (OSTimeGet() - task_preempt_time[OSPrioHighRdy]);
1929 }
1930
1931 task_preempt_count[OSPrioHighRdy]++;
1932
1933 job_number[OSPrioCur]++;
1934 }
1935 // =====Completion=====
1936
1937 else
1938 {
1939     // =====Special Case Preemption=====
1940     // After OSTimeTick(), the task should be completed(i.e. remain_exe_time == 1)
1941     if (OSTCBCur->remain_exe_time == 1)
1942     {
1943         OSTCBCur->remain_exe_time = 0;
1944
1945         INTRU ticks = OSTCBCur->next_job_time - OSTimeGet();
1946         if (OSIntNesting > 0u) { /* See if trying to call from an ISR */
1947             return;
1948         }
1949         if (OSLockNesting > 0u) { /* See if called with scheduler locked */
1950             return;
1951         }
1952         if (ticks > 0u) { /* 0 means no delay! */
1953             OS_ENTER_CRITICAL();
1954             y = OSTCBCur->OSTCBY; /* Delay current task */
1955             OSRdyTbl[y] &= (OS_PRIO)-OSTCBCur->OSTCBBitX;
1956             OS_TRACE_TASK_SUSPENDED(OSTCBCur);
1957             if (OSRdyTbl[y] == 0u) {
1958                 OSRdyGrp &= (OS_PRIO)-OSTCBCur->OSTCBBitY;
1959             }
1960             OSTCBCur->OSTCBY = ticks; /* Load ticks in TCb */
1961             OS_TRACE_TASK_DLY(ticks);
1962             OS_EXIT_CRITICAL();
1963         }
1964         OSTCBCur->already_delay = 1;
1965
1966         printf(" Completion\t");
1967         fprintf(Output_fp, " Completion\t");
1968         // CurrentTask
1969         printf(" task(%2d)(%2d)", OSTCBPrioTbl[OSPrioCur]->OSTCBId, job_number(OSPrioCur));
1970         fprintf(Output_fp, " task(%2d)(%2d)", OSTCBPrioTbl[OSPrioCur]->OSTCBId, job_number(OSPrioCur));
1971         // NextTask
1972         if (OSPrioHighRdy == 63)
1973         {
1974             printf(" task(%2d) \t\t", OSPrioHighRdy);
1975             fprintf(Output_fp, " task(%2d) \t\t", OSPrioHighRdy);
1976         }
1977         else
1978         {
1979             printf(" task(%2d)(%2d)\t\t", OSTCBPrioTbl[OSPrioHighRdy]->OSTCBId, job_number(OSPrioHighRdy));
1980             fprintf(Output_fp, " task(%2d)(%2d)\t\t", OSTCBPrioTbl[OSPrioHighRdy]->OSTCBId, job_number(OSPrioHighRdy));
1981         }
1982         // ResponseTime
1983         printf("%4d\t\t", (OSTimeGet() - task_start_time[OSPrioCur]));
1984         fprintf(Output_fp, "%4d\t\t", (OSTimeGet() - task_start_time[OSPrioCur]));
1985         // # of ContextSwitch
1986         task_preempt_count[OSPrioCur]++;
1987         printf("%7d\t\t", task_preempt_count[OSPrioCur]);
1988         fprintf(Output_fp, "%7d\t\t", task_preempt_count[OSPrioCur]);
1989         // PreemptionTime

```

```

1988 // PreemptionTime
1989 printf("%d\t", task_preempt_time_acc[OSPrioCur]);
1990 fprintf(Output_fp, "%d\t", task_preempt_time_acc[OSPrioCur]);
1991 // DelayTime: Period - ResponseTime
1992 printf("%ld\n", (TaskParameter[OSPrioCur - 1].TaskPeriodic - (OSTimeGet() - task_start_time[OSPrioCur]));
1993 fprintf(Output_fp, "%ld\n", (TaskParameter[OSPrioCur - 1].TaskPeriodic - (OSTimeGet() - task_start_time[OSPrioCur]));
1994
1995 task_preempt_time[OSPrioCur] = 0;
1996 task_preempt_count[OSPrioCur] = 0;
1997 task_preempt_time_acc[OSPrioCur] = 0;
1998
1999 if (task_preempt_time[OSPrioHighRdy] != 0) // if the next task has been preempted
2000 {
2001     task_preempt_time_acc[OSPrioHighRdy] = task_preempt_time_acc[OSPrioHighRdy] + (OSTimeGet() - task_preempt_time[OSPrioHighRdy]);
2002 }
2003
2004 task_preempt_count[OSPrioHighRdy]++;
2005 job_number[OSPrioCur]++;
2006 }
2007 // =====Special Case Preemption=====
2008
2009
2010 // =====Preemption=====
2011 else
2012 {
2013     printf(" Preemption\t");
2014     fprintf(Output_fp, " Preemption\t");
2015     // CurrentTask
2016     if (OSPrioCur == 63)
2017     {
2018         printf(" task(%2d) ", OSPrioCur);
2019         fprintf(Output_fp, " task(%2d) ", OSPrioCur);
2020     }
2021     else
2022     {
2023         printf(" task(%2d)(%2d)", OSTCBPrioTbl[OSPrioCur]->OSTCBId, job_number[OSPrioCur]);
2024         fprintf(Output_fp, " task(%2d)(%2d)", OSTCBPrioTbl[OSPrioCur]->OSTCBId, job_number[OSPrioCur]);
2025     }
2026     // NextTask
2027     if (OSPrioHighRdy == 63)
2028     {
2029         printf(" task(%2d) \n", OSPrioHighRdy);
2030         fprintf(Output_fp, " task(%2d) \n", OSPrioHighRdy);
2031     }
2032     else
2033     {
2034         printf(" task(%2d)(%2d)\n", OSTCBPrioTbl[OSPrioHighRdy]->OSTCBId, job_number[OSPrioHighRdy]);
2035         fprintf(Output_fp, " task(%2d)(%2d)\n", OSTCBPrioTbl[OSPrioHighRdy]->OSTCBId, job_number[OSPrioHighRdy]);
2036     }
2037 }
2038 task_preempt_time[OSPrioCur] = OSTimeGet(); // if the next task has been preempted
2039 task_preempt_count[OSPrioCur]++;
2040
2041 // task_preempt_time[OSPrioHighRdy]
2042 task_preempt_count[OSPrioHighRdy]++;
2043 }
2044 // =====Preemption=====
2045
2046 }
2047
2048 fclose(Output_fp);
2049 // PA#1 part2

```

透過 OSRdyTbl 去確認 task 是否 ready 並進入 waiting，來區分任務為 completion 與 preemption，其中加入 OSPrioCur 與 OSPrioHighRdy，可以協助判斷 task 的切換。

資料	判斷、記錄方式
Event	Task a 切換至 Task b 時,若 Task a 在 OSRdyTbl 上仍處於 ready 狀態(==1),則此情況為 preemption;反之則為 completion。
Response Time	「Completion 的時間點」減掉「task 被設為 ready 的時間點(記錄於 task_start_time)」,即為 response time。
# of ContextSwitch	記錄於 task_preempt_count,每當 task 切換則+1,completion 輸出後歸零。
Preemption Time	發生 preemption 時,記錄「task 被 preempt 的時間點」,與「後續返回到該 task 的時間點」之差距,即為 preemption time。再 task complete 之前有多次 preemption 則進行累加。

Report

這次的 PA1 主要為 RM Scheduling 的實作,首先透過熟悉 TCB 的運作過程,再到如何計算 Response Time 等等工作時間,過程挫折感滿重的,需要將紙本的知識應用於實作上,常常因為不夠清楚,導致一直踩坑。

雖然沒有做出完整的功能,但卻更加熟悉 RM 的設計流程,非常有趣,也透過這次作業知道了生活上有許事情要安排,就好像這次作業一樣,要如何評估是非常重要的,才不會因為期中考與作業同時來的時候,導致 Miss Deadline。