

# **110.1 Embedded OS Implementation**

## **PA#1 Report**

**四電機四乙 B10707119 陳俊宇**

**2021/11/10**

# [ PART I ] Task Control Block Linked List

## Screenshot of result

```
OSTick    created, Thread ID 53644
Task[ 63] created, TCB Address 0068EAC0
-----After TCB[63] being linked-----
Previous TCB pint to address 00000000
Current   TCB pint to address 0068EAC0
Next      TCB pint to address 00000000

Task[  1] created, TCB Address 0068EB18
-----After TCB[1] being linked-----
Previous TCB pint to address 00000000
Current   TCB pint to address 0068EB18
Next      TCB pint to address 0068EAC0

Task[  2] created, TCB Address 0068EB70
-----After TCB[2] being linked-----
Previous TCB pint to address 00000000
Current   TCB pint to address 0068EB70
Next      TCB pint to address 0068EB18

===== TCB linked list =====
Task   Prev_TCB_addr  TCB_addr      Next_TCB_addr
2      00000000        0068EB70      0068EB18
1      0068EB70        0068EB18      0068EAC0
63     0068EB18        0068EAC0      00000000
```

## Screenshot of modified code

OS\_TCBInit()

```
2272     OS_ENTER_CRITICAL();
2273     ptcb->OSTCBNext = OSTCBLst;
2274     ptcb->OSTCBPrev = (OS_TCB *)0;
2275     if (OSTCBLst != (OS_TCB *)0) {
2276         OSTCBLst->OSTCBPrev = ptcb;
2277     }
2278     OSTCBLst = ptcb;
2279
2280
2281     /// PA#1 part1
2282     printf("Task[%3.0d] created, TCB Address %9.0p\n", ptcb->OSTCBPrio, ptcb);
2283     printf("-----After TCB[%d] being linked-----\n", ptcb->OSTCBPrio);
2284     printf("Previous TCB pint to address %9.0p\n", ptcb->OSTCBPrev);    /// (OS_TCB *)0为pointer type, 用%p
2285     printf("Current   TCB pint to address %9.0p\n", ptcb);
2286     printf("Next      TCB pint to address %9.0p\n\n", ptcb->OSTCBNext);
2287     /// PA#1 part1
2288
2289
2290     OSRdyGrp      |= ptcb->OSTCBBitY;    /* Make task ready to run */
2291     OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
2292     OSTaskCtr++;    /* Increment the #tasks counter */
2293     OS_TRACE_TASK_READY(ptcb);
2294     OS_EXIT_CRITICAL();
2295     return (OS_ERR_NONE);
2296 }
2297 OS_EXIT_CRITICAL();
2298 return (OS_ERR_TASK_NO_MORE_TCB);
2299 )
```

main()

```
116 // RM scheduling rules : task with smallest time period will have highest priority
117 // insertion sort by TaskPeriodic => TaskParameter[0]為period最小的Task，升序排列
118 int i, j, key;
119 task_para_set tmp_TaskParameter;
120 for (j = 1; j < TASK_NUMBER; j++)
121 {
122     key = TaskParameter[j].TaskPeriodic;
123     tmp_TaskParameter = TaskParameter[j];
124     i = j - 1;
125     while (i >= 0 && TaskParameter[i].TaskPeriodic > key)
126     {
127         TaskParameter[i + 1] = TaskParameter[i];
128         i = i - 1;
129     }
130     TaskParameter[i + 1] = tmp_TaskParameter;
131 }
132
133 // PA#1，建立Task
134 for (n = 0; n < TASK_NUMBER; n++)
135 {
136     // printf("ID %d, prio %d\n", TaskParameter[n].TaskID, n+1);
137     OSTaskCreateExt(task1, // task function
138         &TaskParameter[n], // p_arg(給task function的參數)
139         &Task_STK[n][TASK_STACKSIZE - 1], // ptos
140         n+1, // prio, PA#1要求從1開始
141         TaskParameter[n].TaskID, // id
142         &Task_STK[n][0], // pbos
143         TASK_STACKSIZE, // stack size
144         &TaskParameter[n], // pext(TCB extensions的pointer)
145         (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR)); // opt
146 }
147
148 // PA#1 part1
149 printf("===== TCB linked list =====\n");
150 printf("Task\tPrev_TCB_addr\tTCB_addr\tNext_TCB_addr\n");
151 for (OS_TCB* ptcb = OSTCBLIST; ptcb != (OS_TCB*)0; ptcb = ptcb->OSTCBNext)
152 {
153     printf("%d\t%p\t%p\t%p\n", ptcb->OSTCBPrio, ptcb->OSTCBPrio, ptcb, ptcb->OSTCBNext);
154 }
155
156 // PA#1 part1
```

## Description of implementation

從 HW#1 進行修改，InputFile()讀取 TaskSet.txt 後，使用 for 迴圈建立 task，確保程式能夠應對數量不定的 TaskSet。

在 OS\_TCBInit() 中，完成 TCB 各項資料 (OSTCBPrio, OSTCBPrev, OSTCBNext.....) 初始化後，輸出 TCB linked list 的 pointer。所有 task 建立完成後，輸出完整的 TCB linked list。

此外，為了符合 PART II 的 RM scheduling，task 會先依照 TaskPeriod 進行排序，period 最短者有最高的 priority。

# [ PART II ] RM Scheduler Implementation

(未以紅色括弧標示之截圖表示全部為新程式碼)

## task function

```
181 void task1(void* p_arg) {
182     task_para_set* task_data;
183     task_data = p_arg;
184
185     int job_ready_time = task_data->TaskArriveTime; /// 第一個job什麼時候被排在schedule上
186     int next_job_ready_time;
187     int exe_time;
188     int job_start_time;
189
190     if (OSTimeGet() < task_data->TaskArriveTime)
191     {
192         OSTimeDly(task_data->TaskArriveTime - OSTimeGet());
193     }
194     while (1)
195     {
196         OSTCBCur->self_continue = 0;
197         next_job_ready_time = job_ready_time + task_data->TaskPeriodic;
198         OSTCBCur->next_job_time = next_job_ready_time;
199         exe_time = task_data->TaskExecutionTime;
200         OSTCBCur->remain_exe_time = exe_time;
201         job_start_time = OSTimeGet();
202
203         while (1)
204         {
205             if (OSTimeGet() != job_start_time)
206             {
207                 exe_time--;
208                 OSTCBCur->remain_exe_time = exe_time;
209                 ///printf("%2d\\TaskID %2d remain exe_time : %2d\\n", OSTimeGet(), task_data->TaskID, exe_time);
210                 if (exe_time == 0)
211                 {
212                     break;
213                 }
214                 job_start_time = OSTimeGet();
215             }
216         }
217
218         job_ready_time = next_job_ready_time; /// 預先更新
219
220         if (OSTCBCur->already_delay == 1)
221         {
222             OSTCBCur->already_delay = 0;
223             continue;
224         }
225
226         int delay_time = next_job_ready_time - OSTimeGet();
227         if (delay_time == 0)
228         {
229             OSTCBCur->self_continue = 1;
230             OS_Sched();
231         }
232         else if (delay_time < 0)
233         {
234             printf("TaskID %2d miss deadline at tick %2d !\\n", task_data->TaskID, OSTimeGet());
235             /// printf("TaskID : %2d\\tend exe at tick %2d, next job start at tick %2d\\n", task_data->TaskID, OSTimeGet(), OSTimeGet() + task_data->TaskPeriodic);
236             OSTimeDly(delay_time);
237         }
238     }
239 }
240
```

根據給定的 arrive time, execution time, period 進行運作的 task function。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。

## ucos\_ii.h

OS\_TCB 額外建立的變數

```
645     INT32U      OSTCBDly;          /* Nbr ticks to delay task or, timeout waiting for event */
646     INT8U       OSTCBStat;         /* Task status */
647     INT8U       OSTCBStatPend;     /* Task PEND status */
648     INT8U       OSTCBPrio;         /* Task priority (0 == highest) */
649
650     INT8U       OSTCBX;            /* Bit position in group corresponding to task priority */
651     INT8U       OSTCBY;            /* Index into ready table corresponding to task priority */
652     OS_PRIO     OSTCBBitX;         /* Bit mask to access bit position in ready table */
653     OS_PRIO     OSTCBBitY;         /* Bit mask to access bit position in ready group */
654
655     /// PA#1
656     INT8U       already_delay;
657     INT8U       remain_exe_time;
658     INT8U       next_job_time;
659     INT8U       self_continue;
660     /// PA#1
```

Task function 執行時的某些資訊(如：task 剩餘的 execution time)需要儲存到 TCB 中，才有辦法在 kernel level 讀取。

## os\_core.c

額外建立的變數

```
938     /// PA#1, 紀錄每個task已執行幾次
939     int job_number[64] = { 0 };
940
941     /// PA#1, 用於確認OSRdyTbl中的某個priority是否為1
942     INT8U OSMaTbl[8] = { 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80 };
943
944     /// PA#1, 用於計算各個Task的response time、紀錄被preempt的次數、時間(# of ContextSwitch, PreemptionTime)
945     int task_start_time[64] = { 0 }; /// 在OSTimeDly()把task設為ready時紀錄
946     int task_preempt_count[64] = { 0 };
947     int task_preempt_time[64] = { 0 };
948     int task_preempt_time_acc[64] = { 0 };
```

OSMaTbl 用於查詢 OSRdyTbl, 確認某個 priority 的 task 是否為 ready 狀態, 其餘 array 則用於記錄需要輸出的資料。

OSTimeTick()

```
1015     if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) == OS_STAT_RDY) { /* Is task suspended?
1016         OSRdyGrp |= ptcb->OSTCBBitY; /* No, Make ready
1017         OSRdyTbl[ptcb->OSTCBY] |= ptcb->OSTCBBitX;
1018
1019         /// PA#1
1020         task_start_time[ptcb->OSTCBPrio] = OSTimeGet();
1021         /// PA#1
1022
1023         OS_TRACE_TASK_READY(ptcb);
1024     }
```

記錄 task 被設為 ready 的時間，用於後續計算 response time。



## OS\_SchedNew()

```

1785     /// PA#1: 控制輸出第一列
1786     int first_line = 1;
1787     static void OS_SchedNew (void)
1788     {
1789         #if OS_LOWEST_PRIO <= 63u                                     /* See if we support up to 64 tasks */
1790             INT8U y;
1791             y = OSUnMapTbl[OSRdyGrp];
1792
1793             OSPrioHighRdy = (INT8U)((y << 3u) + OSUnMapTbl[OSRdyTbl[y]]);
1794
1795             /// PA#1
1796             if ((Output_err = fopen_s(&Output_fp, ".\\Output.txt", "a")) != 0)
1797             {
1798                 printf("Can't open Output.txt");
1799                 exit(0);
1800             }
1801
1802             if (first_line == 1) /// PA#1: 輸出第一列
1803             {
1804                 printf("\nTick\\tEvent\\t\\tCurrentTask ID\\tNextTask ID\\tResponseTime\\t# of ContextSwitch\\tPreemptionTime\\n");
1805                 first_line = 0;
1806             }
1807
1808             /// =====特殊Completion=====
1809             /// task的job結束後，緊接著又是同個task的下個job，如Task Set 3 tick 15
1810             if (OSPrioCur == OSPrioHighRdy)
1811             {
1812                 if (OSTCBCur->self_contime == 1)
1813                 {
1814                     printf("%2d\\t", OSTimeGet());
1815                     fprintf(Output_fp, "%2d\\t", OSTimeGet());
1816
1817                     printf("Completion\\t");
1818                     fprintf(Output_fp, "Completion\\t");
1819                     /// CurrentTask
1820                     if (OSPrioCur == 63) /// 其實不會發生這種情況
1821                     {
1822                         printf("Task(%2d)\\t", OSPrioCur);
1823                         fprintf(Output_fp, "Task(%2d)\\t", OSPrioCur);
1824                     }
1825                     else
1826                     {
1827                         printf("Task(%2d)(%2d)\\t", OSTCBPrioTbl[OSPrioCur]->OSTCBId, job_number[OSPrioCur]);
1828                         fprintf(Output_fp, "Task(%2d)(%2d)\\t", OSTCBPrioTbl[OSPrioCur]->OSTCBId, job_number[OSPrioCur]);
1829                     }
1830                     /// NextTask
1831                     job_number[OSPrioCur]++;
1832                     if (OSPrioHighRdy == 63)
1833                     {
1834                         printf("Task(%2d)\\t", OSPrioHighRdy);
1835                         fprintf(Output_fp, "Task(%2d)\\t", OSPrioHighRdy);
1836                     }
1837                     else
1838                     {
1839                         printf("Task(%2d)(%2d)\\t", OSTCBPrioTbl[OSPrioHighRdy]->OSTCBId, job_number[OSPrioHighRdy]);
1840                         fprintf(Output_fp, "Task(%2d)(%2d)\\t", OSTCBPrioTbl[OSPrioHighRdy]->OSTCBId, job_number[OSPrioHighRdy]);
1841                     }
1842                     /// ResponseTime
1843                     printf("%5d\\t\\t", (OSTimeGet() - task_start_time[OSPrioCur]));
1844                     fprintf(Output_fp, "%5d\\t\\t", (OSTimeGet() - task_start_time[OSPrioCur]));
1845                     /// # of ContextSwitch
1846                     /// task_preempt_count[OSPrioCur]++; /// same task
1847                     printf("%5d\\t\\t\\t", task_preempt_count[OSPrioCur]);
1848                     fprintf(Output_fp, "%5d\\t\\t\\t", task_preempt_count[OSPrioCur]);
1849                     /// PreemptionTime
1850                     printf("%5d\\n", task_preempt_time_acc[OSPrioCur]);
1851                     fprintf(Output_fp, "%5d\\n", task_preempt_time_acc[OSPrioCur]);
1852
1853                     task_preempt_time[OSPrioCur] = 0;
1854                     task_preempt_count[OSPrioCur] = 0;
1855                     task_preempt_time_acc[OSPrioCur] = 0;
1856

```

```

1857 if (task_preempt_time[OSPrioHighRdy] != 0) /// 有被preempt的紀錄
1858 {
1859     task_preempt_time_acc[OSPrioHighRdy] = task_preempt_time_acc[OSPrioHighRdy] + (OSTimeGet() - task_preempt_
1860 }
1861
1862 /// task_preempt_count[OSPrioHighRdy]++; /// same task
1863
1864 task_start_time[OSTCBCur->OSTCBPrio] = OSTimeGet();
1865 }
1866
1867 /// =====特殊Completion=====
1868
1869 /// Task變了 => completed or preempted。 tick = 0時會有arrive time != 0的task completion，不需要輸出
1870 else if ((OSPrioCur != OSPrioHighRdy && OSTimeGet() != 0))
1871 {
1872     if (OSPrioCur == 0)
1873     {
1874         task_start_time[OSPrioHighRdy] = OSTimeGet();
1875     }
1876     else
1877     {
1878         printf("%d\t", OSTimeGet());
1879         fprintf(Output_fp, "%d\t", OSTimeGet());
1880
1881         /// =====一般Completion=====
1882         if ((OSRdyTbl[OSPrioCur >> 3] & OSMapTbl[OSPrioCur & 0x07]) == 0) /// 表示原本的Task已完成，進入等待狀態
1883         {
1884             printf("Completion\t");
1885             fprintf(Output_fp, "Completion\t");
1886             /// CurrentTask
1887             if (OSPrioCur == 63) /// 其實不會發生這種情況
1888             {
1889                 printf("Task(%d)\t", OSPrioCur);
1890                 fprintf(Output_fp, "Task(%d)\t", OSPrioCur);
1891             }
1892             else
1893             {
1894                 printf("Task(%d)(%d)\t", OSTCBCPrioTbl[OSPrioCur]->OSTCBId, job_number[OSPrioCur]);
1895                 fprintf(Output_fp, "Task(%d)(%d)\t", OSTCBCPrioTbl[OSPrioCur]->OSTCBId, job_number[OSPrioCur]);
1896             }
1897             /// NextTask
1898             if (OSPrioHighRdy == 63)
1899             {
1900                 printf("Task(%d)\t", OSPrioHighRdy);
1901                 fprintf(Output_fp, "Task(%d)\t", OSPrioHighRdy);
1902             }
1903             else
1904             {
1905                 printf("Task(%d)(%d)\t", OSTCBCPrioTbl[OSPrioHighRdy]->OSTCBId, job_number[OSPrioHighRdy]);
1906                 fprintf(Output_fp, "Task(%d)(%d)\t", OSTCBCPrioTbl[OSPrioHighRdy]->OSTCBId, job_number[OSPrioHighRdy]);
1907             }
1908             /// ResponseTime
1909             printf("%5d\t\t", (OSTimeGet() - task_start_time[OSPrioCur]));
1910             fprintf(Output_fp, "%5d\t\t", (OSTimeGet() - task_start_time[OSPrioCur]));
1911             /// # of ContextSwitch
1912             task_preempt_count[OSPrioCur]++;
1913             printf("%5d\t\t\t", task_preempt_count[OSPrioCur]);
1914             fprintf(Output_fp, "%5d\t\t\t", task_preempt_count[OSPrioCur]);
1915             /// PreemptionTime
1916             printf("%5d\n", task_preempt_time_acc[OSPrioCur]);
1917             fprintf(Output_fp, "%5d\n", task_preempt_time_acc[OSPrioCur]);
1918
1919             task_preempt_time[OSPrioCur] = 0;
1920             task_preempt_count[OSPrioCur] = 0;
1921             task_preempt_time_acc[OSPrioCur] = 0;
1922
1923             if (task_preempt_time[OSPrioHighRdy] != 0) /// 有被preempt的紀錄
1924             {
1925                 task_preempt_time_acc[OSPrioHighRdy] = task_preempt_time_acc[OSPrioHighRdy] + (OSTimeGet() - task_pree
1926             }
1927
1928             task_preempt_count[OSPrioHighRdy]++; /// 換過去第一次也算，才會符合PA#1範例

```

```

1929     job_number[OSPrioCur]++;
1930 }
1931
1932 /// =====一般Completion=====
1933
1934
1935 else /// 表示原本的Task尚未完成，preempt(必定是低prio換到高prio)
1936 {
1937     /// =====特殊Preemption=====
1938     /// task在OSTimeTick()之後應complete(i.e. remain_exe_time == 1)，但先被preempt掉了，如Task Set 2 tick 12
1939     if (OSTCBCur->remain_exe_time == 1)
1940     {
1941         OSTCBCur->remain_exe_time = 0;
1942
1943         INT8U ticks = OSTCBCur->next_job_time - OSTimeGet();
1944         if (OSIntNesting > 0u) { /* See if trying to call from an ISR */ /*
1945             return;
1946         }
1947         if (OSLockNesting > 0u) { /* See if called with scheduler locked */ /*
1948             return;
1949         }
1950         if (ticks > 0u) { /* 0 means no delay! */ /*
1951             OS_ENTER_CRITICAL();
1952             y = OSTCBCur->OSTCBY; /* Delay current task */ /*
1953             OSRdyTbl[y] &= (OS_PRIO)~OSTCBCur->OSTCBBitX;
1954             OS_TRACE_TASK_SUSPENDED(OSTCBCur);
1955             if (OSRdyTbl[y] == 0u) {
1956                 OSRdyGrp &= (OS_PRIO)~OSTCBCur->OSTCBBitY;
1957             }
1958             OSTCBCur->OSTCBEDly = ticks; /* Load ticks in TCB */ /*
1959             OS_TRACE_TASK_DLY(ticks);
1960             OS_EXIT_CRITICAL();
1961         }
1962         OSTCBCur->already_delay = 1;
1963
1964         printf("Completion\t");
1965         fprintf(Output_fp, "Completion\t");
1966         /// CurrentTask
1967         if (OSPrioCur == 63) /// 其實不會發生這種情況
1968         {
1969             printf("Task(%2d)\t", OSPrioCur);
1970             fprintf(Output_fp, "Task(%2d)\t", OSPrioCur);
1971         }
1972         else
1973         {
1974             printf("Task(%2d)(%2d)\t", OSTCBPrioTbl[OSPrioCur]->OSTCBId, job_number[OSPrioCur]);
1975             fprintf(Output_fp, "Task(%2d)(%2d)\t", OSTCBPrioTbl[OSPrioCur]->OSTCBId, job_number[OSPrioCur]);
1976         }
1977         /// NextTask
1978         if (OSPrioHighRdy == 63)
1979         {
1980             printf("Task(%2d)\t", OSPrioHighRdy);
1981             fprintf(Output_fp, "Task(%2d)\t", OSPrioHighRdy);
1982         }
1983         else
1984         {
1985             printf("Task(%2d)(%2d)\t", OSTCBPrioTbl[OSPrioHighRdy]->OSTCBId, job_number[OSPrioHighRdy]);
1986             fprintf(Output_fp, "Task(%2d)(%2d)\t", OSTCBPrioTbl[OSPrioHighRdy]->OSTCBId, job_number[OSPrioHighRdy]);
1987         }
1988         /// ResponseTime
1989         printf("%5d\t\t", (OSTimeGet() - task_start_time[OSPrioCur]));
1990         fprintf(Output_fp, "%5d\t\t", (OSTimeGet() - task_start_time[OSPrioCur]));
1991         /// # of ContextSwitch
1992         task_preempt_count[OSPrioCur]++;
1993         printf("%5d\t\t\t", task_preempt_count[OSPrioCur]);
1994         fprintf(Output_fp, "%5d\t\t\t", task_preempt_count[OSPrioCur]);
1995         /// PreemptionTime
1996         printf("%5d\n", task_preempt_time_acc[OSPrioCur]);
1997         fprintf(Output_fp, "%5d\n", task_preempt_time_acc[OSPrioCur]);
1998
1999         task_preempt_time[OSPrioCur] = 0;
2000         task_preempt_count[OSPrioCur] = 0;

```



```

2001 task_preempt_time_acc[OSPrioCur] = 0;
2002
2003 if (task_preempt_time[OSPrioHighRdy] != 0) /// 有被preempt的紀錄
2004 {
2005     task_preempt_time_acc[OSPrioHighRdy] = task_preempt_time_acc[OSPrioHighRdy] + (OSTimeGet() - task_preempt_time[OSPrioHighRdy]);
2006 }
2007
2008 task_preempt_Count[OSPrioHighRdy]++; /// 換過去第一次也算，才會符合PA#1範例
2009
2010 job_number[OSPrioCur]++;
2011 }
2012 /// =====特殊Preemption=====
2013
2014
2015 /// =====一般Preemption=====
2016 else
2017 {
2018     printf("Preemption\t");
2019     fprintf(Output_fp, "Preemption\t");
2020     /// CurrentTask
2021     if (OSPrioCur == 63)
2022     {
2023         printf("Task(%2d)\t", OSPrioCur);
2024         fprintf(Output_fp, "Task(%2d)\t", OSPrioCur);
2025     }
2026     else
2027     {
2028         printf("Task(%2d)(%2d)\t", OSTCEPrioTbl[OSPrioCur]->OSTCBId, job_number[OSPrioCur]);
2029         fprintf(Output_fp, "Task(%2d)(%2d)\t", OSTCEPrioTbl[OSPrioCur]->OSTCBId, job_number[OSPrioCur]);
2030     }
2031     /// NextTask
2032     if (OSPrioHighRdy == 63)
2033     {
2034         printf("Task(%2d)\t", OSPrioHighRdy);
2035         fprintf(Output_fp, "Task(%2d)\t", OSPrioHighRdy);
2036     }
2037     else
2038     {
2039         printf("Task(%2d)(%2d)\n", OSTCEPrioTbl[OSPrioHighRdy]->OSTCBId, job_number[OSPrioHighRdy]);
2040         fprintf(Output_fp, "Task(%2d)(%2d)\n", OSTCEPrioTbl[OSPrioHighRdy]->OSTCBId, job_number[OSPrioHighRdy]);
2041     }
2042
2043     task_preempt_time[OSPrioCur] = OSTimeGet(); /// 先記錄何時被preempt
2044     task_preempt_Count[OSPrioCur]++;
2045
2046     /// task_preempt_time[OSPrioHighRdy] 不動
2047     task_preempt_Count[OSPrioHighRdy]++;
2048 }
2049 /// =====一般Preemption=====
2050 }
2051 }
2052
2053 fclose(Output_fp);
2054 /// PA#1

```

從 OS\_SchedNew() 中，觀察 task 之間的切換情形，取得需要的輸出資訊。

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 preemptpion；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 preemptpion 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 preemptpion 則進行累加。