

[PART II] RM Scheduler Implementation.



Fig 4. The schedule of each task.

1. Screenshot results (with the given format) of four task sets. (Time ticks 0-30 or miss deadline).

Tick	Event	CurrentTask ID	NextTask ID	ResponseTime	# of ContextWswitch
1	Completion	task(1)(0)	task(2)(0)	1	1
3	Preemption	task(2)(0)	task(1)(1)		
4	Completion	task(1)(1)	task(2)(0)	1	2
5	Completion	task(2)(0)	task(63)	5	4
6	Preemption	task(63)	task(1)(2)		
7	Completion	task(1)(2)	task(2)(1)	1	2
9	Preemption	task(2)(1)	task(1)(3)		
10	Completion	task(1)(3)	task(2)(1)	1	2
11	Completion	task(2)(1)	task(63)	5	4
12	Preemption	task(63)	task(1)(4)		
13	Completion	task(1)(4)	task(2)(2)	1	2
15	Preemption	task(2)(2)	task(1)(5)		
16	Completion	task(1)(5)	task(2)(2)	1	2
17	Completion	task(2)(2)	task(63)	5	4
18	Preemption	task(63)	task(1)(6)		
19	Completion	task(1)(6)	task(2)(3)	1	2
21	Preemption	task(2)(3)	task(1)(7)		
22	Completion	task(1)(7)	task(2)(3)	1	2
23	Completion	task(2)(3)	task(63)	5	4
24	Preemption	task(63)	task(1)(8)		
25	Completion	task(1)(8)	task(2)(4)	1	2
27	Preemption	task(2)(4)	task(1)(9)		
28	Completion	task(1)(9)	task(2)(4)	1	2
29	Completion	task(2)(4)	task(63)	5	4
30	Preemption	task(63)	task(1)(10)		

Fig 5. Result of Task set 1

Tick	Event	CurrentTask ID	NextTask ID	ResponseTime	# of ContextWswitch
2	Completion	task(2)(0)	task(1)(0)	2	1
5	Preemption	task(1)(0)	task(2)(1)		
7	Completion	task(2)(1)	task(1)(0)	2	2
10	Preemption	task(1)(0)	task(2)(2)		
12	Completion	task(2)(2)	task(1)(0)	2	2
14	Completion	task(1)(0)	task(63)	14	6
15	Preemption	task(63)	task(2)(3)		
17	Completion	task(2)(3)	task(1)(1)	2	2
20	Preemption	task(1)(1)	task(2)(4)		
22	Completion	task(2)(4)	task(1)(1)	2	2
25	Preemption	task(1)(1)	task(2)(5)		
27	Completion	task(2)(5)	task(1)(1)	2	2
29	Completion	task(1)(1)	task(63)	14	6
30	Preemption	task(63)	task(2)(6)		
32	Completion	task(2)(6)	task(1)(2)	2	2
35	Preemption	task(1)(2)	task(2)(7)		

Fig 6. Result of Task set 2

Tick	Event	CurrentTask ID	NextTask ID	ResponseTime	# of ContextWswitch
1	Preemption	task(2)(0)	task(1)(0)		
2	Completion	task(1)(0)	task(2)(0)	1	2
4	Preemption	task(2)(0)	task(1)(1)		
5	Completion	task(1)(1)	task(2)(0)	1	2
6	Completion	task(2)(0)	task(2)(1)	6	5
7	Preemption	task(2)(1)	task(1)(2)		
8	Completion	task(1)(2)	task(2)(1)	1	2
10	Preemption	task(2)(1)	task(1)(3)		
11	Completion	task(1)(3)	task(2)(1)	1	2
12	Completion	task(2)(1)	task(2)(2)	6	6
13	Preemption	task(2)(2)	task(1)(4)		
14	Completion	task(1)(4)	task(2)(2)	1	2
16	Preemption	task(2)(2)	task(1)(5)		
17	Completion	task(1)(5)	task(2)(2)	1	2
18	Completion	task(2)(2)	task(2)(3)	6	6
19	Preemption	task(2)(3)	task(1)(6)		
20	Completion	task(1)(6)	task(2)(3)	1	2
22	Preemption	task(2)(3)	task(1)(7)		
23	Completion	task(1)(7)	task(2)(3)	1	2
24	Completion	task(2)(3)	task(2)(4)	6	6
25	Preemption	task(2)(4)	task(1)(8)		
26	Completion	task(1)(8)	task(2)(4)	1	2
28	Preemption	task(2)(4)	task(1)(9)		
29	Completion	task(1)(9)	task(2)(4)	1	2
30	Completion	task(2)(4)	task(2)(5)	6	6

Fig 7. Result of Task set 3

Tick	Event	CurrentTask ID	NextTask ID	ResponseTime	# of ContextWswitch
1	Preemption	task(1)(0)	task(3)(0)		
2	Completion	task(3)(0)	task(1)(0)	1	2
5	Completion	task(1)(0)	task(2)(0)	5	3
6	Preemption	task(2)(0)	task(3)(1)		
7	Completion	task(3)(1)	task(1)(1)	1	2
11	Completion	task(1)(1)	task(3)(2)	5	2
12	MissDeadline	task(2)(0)	-----		

Fig 8. Result of Task set 4

2. Report that describes your implementation (please attach the screenshot of the code and **MARK** the modified part).

```

636 //TCB建立一些指標參數
637 INT32U execution_time;
638 INT32U job_ID;
639 INT32U execution_en;
640 INT32U task_start;
641 INT32U task_stop;
642 INT32U sortID;
643 } OS_TCB;
644

```

Fig 9. TCB 建立指標參數

```

132 //建立task
133 OSTaskCreateExt(task1,
134 0,
135 #task1_STK(TASK_STKSIZE - 1),
136 TASK1_PRIORITY,
137 TASK1_ID,
138 #task1_STK(0),
139 TASK_STKSIZE,
140 0,
141 (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR));
142
143 OSTaskCreateExt(task2,
144 0,
145 #task2_STK(TASK_STKSIZE - 1),
146 TASK2_PRIORITY,
147 TASK2_ID,
148 #task2_STK(0),
149 TASK_STKSIZE,
150 0,
151 (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR));
152
153 OSTaskCreateExt(task3,
154 0,
155 #task3_STK(TASK_STKSIZE - 1),
156 TASK3_PRIORITY,
157 TASK3_ID,
158 #task3_STK(0),
159 TASK_STKSIZE,
160 0,
161 (OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR));
162
163 //宣告task
164 sortID = 0; // sort用到的參數
165 sortID = 0; // sort用到的參數
166

```

Fig 10. task 建立與 sort 參數

```

2004 //在task中使用的宏(有些宏在測試或運行時使用)
2005 int period;
2006 int execution_time;
2007 int arrival_time;
2008 int period2;
2009 int execution_time2;
2010 int arrival_time2;
2011 int period3;
2012 int execution_time3;
2013 int arrival_time3;
2014 int before_time;
2015 int interrupt_time;
2016 int total;
2017 int inter_time;
2018 int OSTimeEnter_time;
2019 int OSTimeExit_time;
2020 int task1_time;
2021 int task2_time;
2022 int task3_time;
2023 int task1_response_time;
2024 int task2_response_time;
2025 int task3_response_time;
2026 int task1_context_switch;
2027 int task2_context_switch;
2028 int task3_context_switch;
2029 int sortID;
2030

```

Fig 11. 全域變數宣告

```

56 static OS_STK StartupTaskStk[APP_OS_STARTUP_TASK_STK_SIZE];
57
58 //宣告task
59 #define TASK_STKSIZE 2048
60 #define TASK1_PRIORITY 1
61 #define TASK2_PRIORITY 2
62 #define TASK3_PRIORITY 3
63 #define TASK1_ID 1
64 #define TASK2_ID 2
65 #define TASK3_ID 3
66
67 static OS_STK Task1Stk[TASK_STKSIZE];
68 static OS_STK Task2Stk[TASK_STKSIZE];
69 static OS_STK Task3Stk[TASK_STKSIZE];
70
71 static void task1(void *p_arg);
72 static void task2(void *p_arg);
73 static void task3(void *p_arg);
74
75 //宣告task
76

```

Fig 12. 宣告 task

```

215 //task1內部結構
216 void task1(void *p_arg) {
217 (void)p_arg;
218
219 period = 3;
220 execution_time = 1;
221 arrival_time = 1;
222 INT32U start;
223
224 OSTimeDly(arrival_time);
225 while (1) {
226 task1_time = 0;
227 start = OSTimeGet();
228 OSTimeCur->task_start = OSTimeGet();
229 while (task1_time < execution_time)
230 {
231 //Do something
232 }
233 OSTimeDly(period - execution_time);
234 }
235 }
236

```

Fig 13. Task1 內部結構

```

237 //task2內部結構
238 void task2(void *p_arg) {
239 (void)p_arg;
240 period2 = 4;
241 execution_time2 = 4;
242 arrival_time2 = 0;
243 INT32U start2;
244 int dc = 0;
245
246 OSTimeDly(arrival_time2);
247 while (1) {
248 start2 = OSTimeGet();
249 OSTimeCur->task_start = OSTimeGet();
250 while (task2_time < execution_time2)
251 {
252 task2_time = 0;
253 if (period2 - (task1_response_time * execution_time1 + execution_time2) == 0) {
254 OS_Sched();
255 }
256 else {
257 OSTimeDly(period2 - (task1_response_time * execution_time1 + execution_time2));
258 }
259 }
260 }
261 }
262

```

Fig 14. Task2 內部結構

```

263 //task3內部結構
264 void task3(void *p_arg) {
265 (void)p_arg;
266 period3 = 10;
267 execution_time3 = 2;
268 arrival_time3 = 2;
269 INT32U start3;
270
271 while (1) {
272 task3_time = 0;
273 start3 = OSTimeGet();
274 OSTimeCur->task_start = OSTimeGet();
275 while (task3_time < execution_time3)
276 {
277 //Do something
278 }
279 OSTimeDly(period3 - (task1_response_time * execution_time1 + execution_time2 + task3_response_time * execution_time3));
280 }
281 }
282

```

Fig 15. task3 內部結構

```

869 //判斷是否sort成功，宣告taskID
870 if (sortQ4 == 1 && sortQ2 == 0) {
871     if (OSPrCur == 1) {
872         OSTCHighdy->sortID = 3;
873     }
874     else if (OSPrHighdy == 1) {
875         OSTCHighdy->sortID = 3;
876     }
877 }
878 if (OSPrCur == 2) {
879     OSTCHighdy->sortID = 1;
880 }
881 else if (OSPrHighdy == 2) {
882     OSTCHighdy->sortID = 1;
883 }
884 if (OSPrCur == 3) {
885     OSTCHighdy->sortID = 2;
886 }
887 else if (OSPrHighdy == 3) {
888     OSTCHighdy->sortID = 2;
889 }
890 else if (sortQ2 == 1 && sortQ4 == 0) {
891     if (OSPrCur == 1) {
892         OSTCHighdy->sortID = 2;
893     }
894     else if (OSPrHighdy == 1) {
895         OSTCHighdy->sortID = 2;
896     }
897 }
898 if (OSPrCur == 2) {
899     OSTCHighdy->sortID = 1;
900 }
901 else if (OSPrHighdy == 2) {
902     OSTCHighdy->sortID = 1;
903 }
904 }
905 }
906 }

```

Fig 16. sort 判斷-1

```

//如果不是sort的話，不論sort狀態，宣告taskID
else if (sortQ2 == 0 && sortQ4 == 0)
{
    if (OSPrCur == 1) {
        OSTCHighdy->sortID = 1;
    }
    else if (OSPrHighdy == 1) {
        OSTCHighdy->sortID = 1;
    }
}
if (OSPrCur == 2) {
    OSTCHighdy->sortID = 2;
}
else if (OSPrHighdy == 2) {
    OSTCHighdy->sortID = 2;
}
}
if (OSPrCur == 3) {
    OSTCHighdy->sortID = 3;
}
else if (OSPrHighdy == 3) {
    OSTCHighdy->sortID = 3;
}
}
//.....

```

Fig 17. sort 判斷-2

```

773 //判斷task目前是否會在進程tick做完，如果會做完，會直接做完到timedly()，不會經過interrupt的sw
774 if (task2_time == execution_time2) {
775     OS_TRACE_ISR_EXIT();
776     OS_EXIT_CRITICAL();
777 }
778 //判斷task目前是否會在進程tick做完，如果不會做完，會進入死循環
779 else if (OSTimeGet() == 12 && task1_time < execution_time3) {
780     printf("task1 is dead time task 2) (0) t-----\n", OSTimeGet());
781     while (1) {
782     }
783 }
784 }

```

Fig 17. 判斷是否無法預期做完與強制做完繞過中斷判斷

```

808 if (OSTCHighdy == OSTCHighdy) {
809     if (OSTask_Priority > 0) {
810         OSTCHighdy->OSTaskCtr++;
811     }
812     #endif
813     OSTaskCtr++;
814     #endif
815     //判斷OSTask_Priority 是否為 OSPrCur，如果是，則是優先級提升。
816     task2_response_time = task1_response_time + execution_time1 + execution_time2;
817     printf("task1 completion task 2) (0) t-----\n", OSTimeGet(), OSPrCur, OSTCHighdy->job_ID,
818         OSTCHighdy, OSTCHighdy->job_ID + 1, task2_response_time, task2_contextswitch);
819     OSTCHighdy->job_ID += 1;
820 }
821 task1_response_time = 0;
822 task2_contextswitch = 1;
823 //判斷OSTask_Priority 是否為 OSPrCur，如果是，則是優先級提升。
824 }
825 }

```

Fig 18. 判斷目前跟下一個 task 一樣

```

1933 //計算每個task的contextswitch
1934 OSTCHighdy->task_stop = OSTimeGet();
1935 before_time = OSTimeGet();
1936 if (OSPrCur == 1 || OSPrHighdy == 1 && OSTimeGet() > 0) {
1937     ++task1_contextswitch;
1938 }
1939 if ((OSPrHighdy == 2 || OSPrCur == 2) && OSTimeGet() > 0) {
1940     ++task2_contextswitch;
1941 }
1942 if ((OSPrHighdy == 3 || OSPrCur == 3) && OSTimeGet() > 0) {
1943     ++task3_contextswitch;
1944     ++task3_time;
1945 }
1946 //計算每個task的contextswitch
1947 }

```

Fig 19. 計算 context switch

```

1948 //顯示task的response time contextswitch 工作多少以及目前task的狀態和下一個執行的task
1949 if (OSTCHighdy == 1 && OSTimeGet() > 0) {
1950     task1_response_time++;
1951     printf("task1 completion task 2) (0) t-----\n", OSTimeGet(), OSPrCur->sortID, OSTCHighdy->job_ID,
1952         OSTCHighdy->sortID, OSTCHighdy->job_ID, (OSTCHighdy->task_stop - OSTCHighdy->task_start), task1_contextswitch);
1953     task1_contextswitch = 0;
1954 }
1955 else if (OSTCHighdy == 2) {
1956     task2_response_time = task1_response_time + execution_time1 + execution_time2;
1957 }
1958 if (OSTCHighdy == 3) {
1959     printf("task1 completion task 2) (0) t-----\n", OSTimeGet(), OSPrCur->sortID, OSTCHighdy->job_ID,
1960         OSTCHighdy->sortID, task2_response_time, task2_contextswitch);
1961     task2_contextswitch = 0;
1962     task2_response_time = 0;
1963 }
1964 else {
1965     printf("task1 completion task 2) (0) t-----\n", OSTimeGet(), OSPrCur->sortID, OSTCHighdy->job_ID,
1966         OSTCHighdy->sortID, task2_response_time, task2_contextswitch);
1967     task2_contextswitch = 0;
1968     task2_response_time = 0;
1969 }
1970 if (OSTimeGet() > 0) {
1971     OSTCHighdy->job_ID += 1;
1972 }

```

Fig 20. 顯示 task 相關數據

一開始我先宣告我所要的變數以及 task 的建立，如圖 9.到圖 11.所示，並依據題目的要求去建立幾個 task，以及內部參數的調整，包含週期、執行時間、延遲多久開始。因為透過 ready table 會知道，哪些 task 是已 high ready，但礙於優先權比其他 task 低，而且 response time 時間還是算在裡面所以必須把之前在他 high ready 卻沒執行的時間必須考慮進去，這樣才會是那個 task 的 time delay 時間，如果最高優先權就不會有這樣的問題，還有在當 task 執行完成時，且他的

period 減去所經過的時間等於 0 時，timedly 是不會進到 OS_Sched 去做切換，所以要直接讓他到 OS_Sched 才可順利做下去，如圖 13.到圖 15.。因為考慮到 task 可能優先順序不是照 123 這樣排，如圖 16.與圖 17.所示，去判斷說是否需要 sort 的部分。當前的 task 與下個進來的 task 是相同時，在原本裡面是沒有判斷式，我多設了一個判斷是否有一致，否則他會以為是相同的 task 還沒做完，而一直繼續，直到較優的優先權進來為止，如圖 18.所示。每一個 tick 都會進去檢查說目前是否有較高優先權的 task 進來，且每次進來的延遲到下個 tick 時間，我再每次檢查都會給目前 task 的完成進度，假設進來檢查已表示做完但是下個較高優先權的 task 進來，會強制讓此 task 做完，再到 OSTimeDly 切到較高優先權的 task 再做，避免已經做完的 task 還被中斷，以及當有 task 再最長周期到還無法做完時，會跳出 MissDeadline，如圖 17.所示。計算此 task 完成所需的 context switch、response time，最後再顯示目前 task 狀態，包含工作次數、response time、context switch、event、current task、next task。

備註:給的程式包含全部，但我預設是用 task set1，其他都先註解掉了。