[PART I] Task Control Block Linked List

1. The screenshot results.

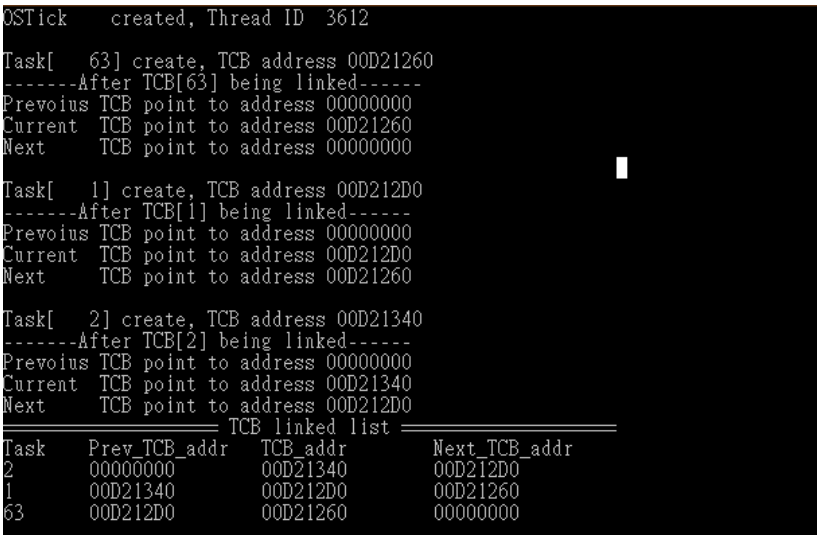


Fig 1. Result of Task Control Block Linked List

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

```
2387    //print出taks目前、上個、下個的TASK狀態
2388    printf("\nTask[\t%d] create, TCB address %08X", prio, OSTCBPrioTbl[prio]);
2389    printf("\n-------After TCB[%d] being linked------",ptcb->OSTCBPrio);
2390    printf("\nPrevoius TCB point to address %08X", ptcb->OSTCBPrev);
2391    printf("\nCurrent  TCB point to address %08X", OSTCBPrioTbl[prio]);
2392    printf("\nNext     TCB point to address %08X\n",ptcb->OSTCBNext);
```

Fig 2. Task Control Block Linked List-code1

```
969     //製作 TCB linked list 表
970     printf("============== TCB linked list ==============\n");
971     printf("Task\tPrev_TCB_addr\tTCB_addr\tNext_TCB_addr\n");
972     while (tcb_ > 0)
973     {
974        printf("%d\t%08X\t%08X\t%08X\n", tcb_->OSTCBPrio, tcb_->OSTCBPrev, tcb_, tcb_->OSTCBNext);
975        tcb_ = tcb_->OSTCBNext;
976     }
977     printf("\nTick\tEvent\t    CurremtTask ID\tNextTask ID\tResponseTime\t # of ContextWsitch\n");
978     //製作 TCB linked list 表
```

Fig 3. Task Control Block Linked List- code2

我在OS_TCBInit找到他每個建立task的排成，並且把這些結果顯示出來，如圖2.所示，最後在道OSStart顯示，建立完之後的TCB linked list，透過OS_TCB傳址將記憶體的資料顯示在視窗上，如圖3.所示。

[PART II] RM Scheduler Implementation.

Fig 4. The schedule of each task.

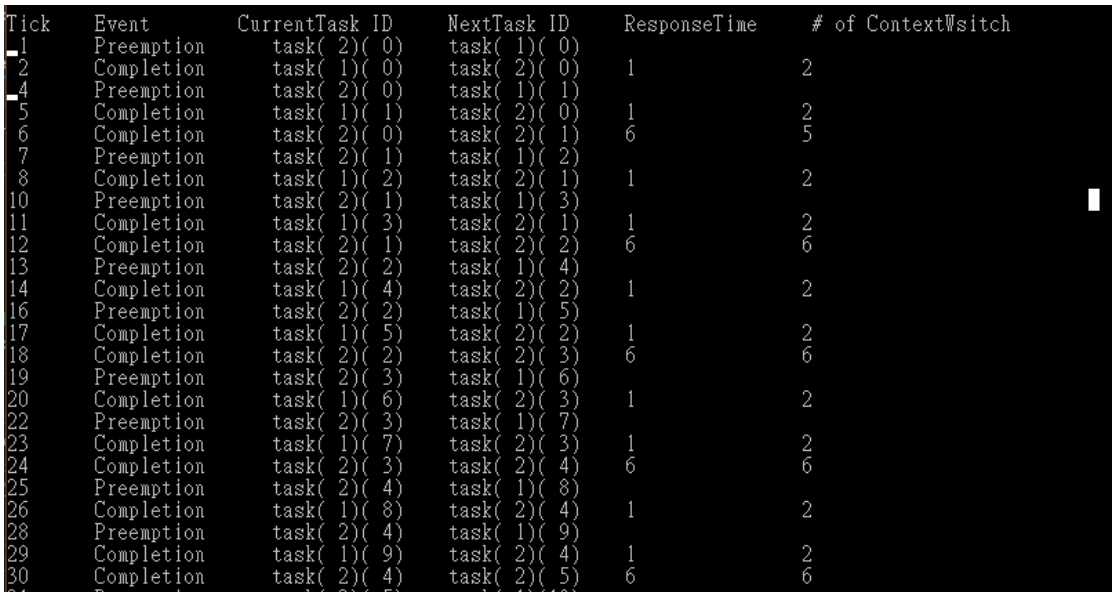
1. **Screenshot results (with the given format) of four task sets. (Time ticks 0-30 or miss deadline).**

Tick	Event	CurrentTask ID	NextTask ID	ResponseTime	# of ContextWsitch
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

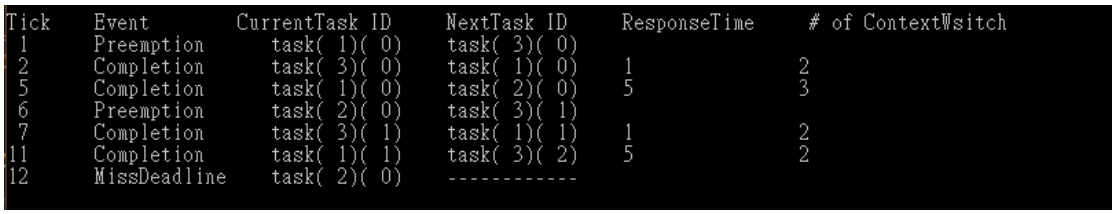
Tick	Event	CurrentTask ID	NextTask ID	ResponseTime	# of ContextWsitch
2	Completion	task(2)(0)	task(1)(0)	2	1
5	Preemption	task(1)(0)	task(2)(1)		
7	Completion	task(2)(1)	task(1)(0)	2	2
10	Preemption	task(1)(0)	task(2)(2)		
12	Completion	task(2)(2)	task(1)(0)	2	2
14	Completion	task(1)(0)	task(63)	14	6
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 ContextWsitch
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 ContextWsitch
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).**

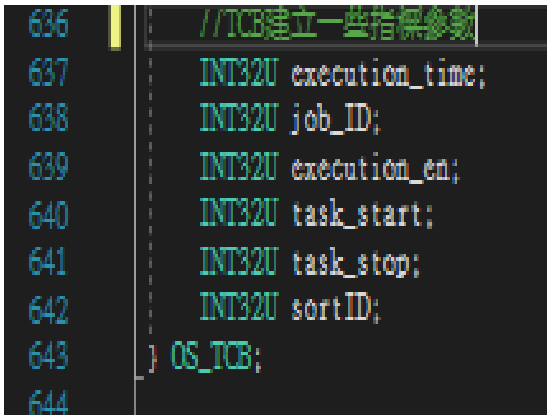


Fig 9. TCB 建立指標參數

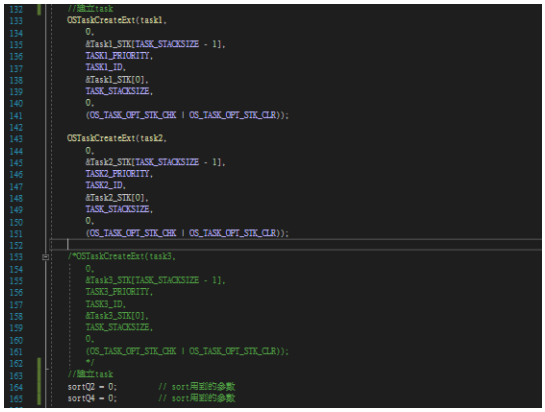


Fig 10. task 建立與 sort 參數

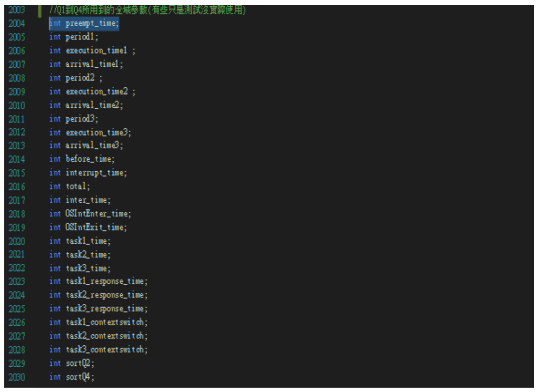


Fig 11. 全域變數宣告

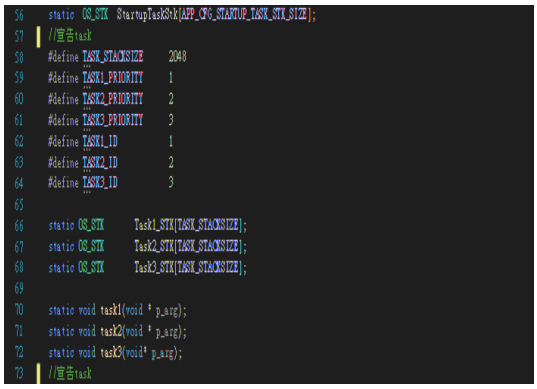


Fig 12. 宣告 task

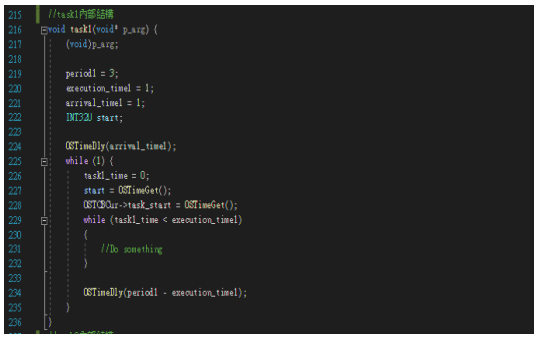


Fig 13. Task1 內部結構

Fig 14. Task2 內部結構

Fig 15. task3 內部結構

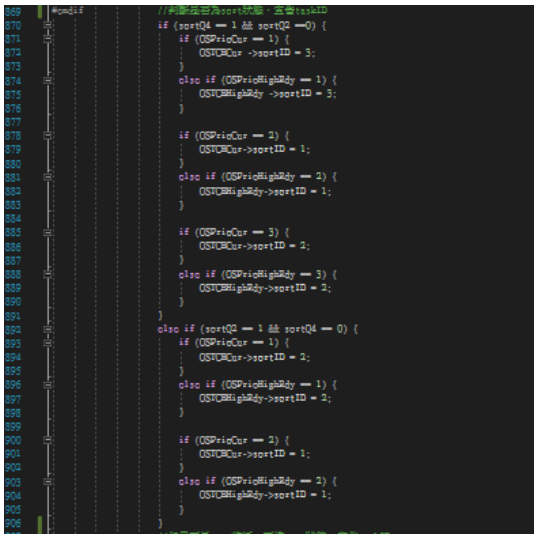


Fig 16. sort 判斷-1

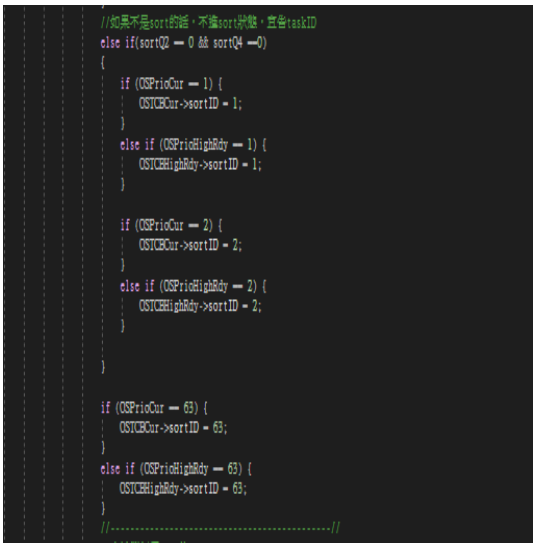


Fig 17. sort 判斷-2

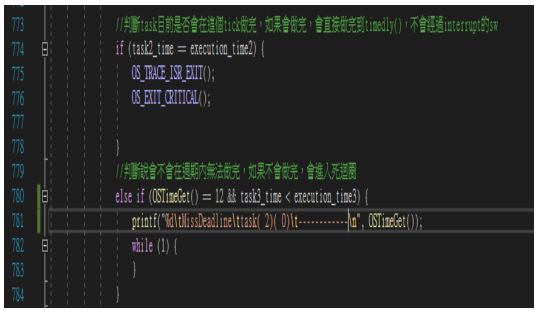


Fig 17. 判斷是否無法預期做完與強制做完繞過中斷判斷

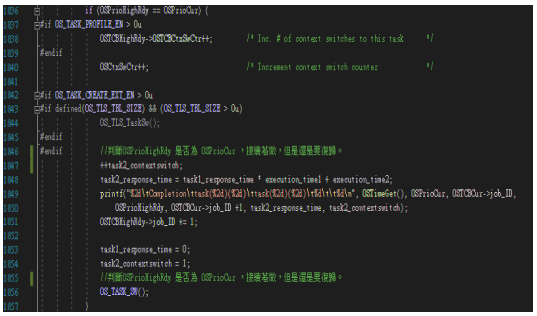


Fig 18. 判斷目前跟下一個 task 一樣

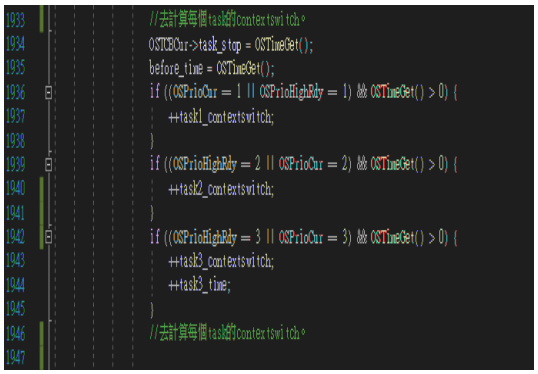


Fig 19. 計算 context switch

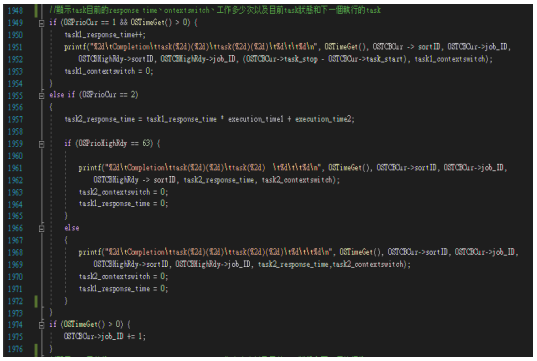


Fig 20. 顯示 task 相關數據

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。