

為了能夠記錄每個task的優先級，我新增了一個陣列

uint8\_t context\_priority[MAX\_TASKS];

在os.h中，我新增了一個context用來記錄kernel 進入 schedule 前的狀態，以便task結束後能返回kernel。

struct context os\_tasks;

接著我在kernel裡，schedule() 的前一步使用 sys\_switch() 將當前kernel的狀態保存下來

    printf("\n\n\n----- schedule start -----\n\n\n");

    //利用 sys\_switch() 的特性，讓進入 schedule() 前的狀態能保存在 kernel\_context 裡

    sys\_switch(&kernel\_context, &kernel\_context);

    schedule();

製作 task\_exit() 函數，呼叫它就可以利用switch\_to() 回到kernel，並且剛好下一步就會再進入schedule() 中

void task\_exit()

{

    switch\_to(&kernel\_context);

}

另外我做了10個顯示不同圖案的task用來測試，他們會讀入一個字串指標的參數並顯示出來，以下是其中之一

//第1個task

void user\_task1(char\* sentence)

{

    printf("Task 1: Running...|%s\n", sentence);

    uart\_puts("                  |...\n");

    uart\_puts("                     |...\n");

    uart\_puts("                        |...\n");

    task\_delay(6000);

    task\_exit();

}

以下是我改寫的 task\_create() 函數，在新增task時會把要給該task的參數放在它的context中的a0暫存器，並且記錄下它的優先級。

int task\_create(void (\*start\_routin)(char \*param), char \*param, uint8\_t priority)

{

    if (\_top < MAX\_TASKS) {

        //以當前的TOP值來分配給函數指標指向的task一個context

        //並設定好它的stack point 和 raturn address

        ctx\_tasks[\_top].sp = (reg\_t) &task\_stack[\_top][STACK\_SIZE - 1];

        ctx\_tasks[\_top].ra = (reg\_t) start\_routin;

        //將要給該task的參數設在該context的 a0 暫存器，這樣 context\_switch 完就能使用

        ctx\_tasks[\_top].a0 = (reg\_t) param;

        //把該 task 的 priority 存在對應的陣列位置中

        context\_priority[\_top] = priority;

        \_top++;

        return 0;

    } else {

        return -1;

    }

}

另外也做一個用來存放task是否做過的陣列，做過就將值設為1

uint8\_t task\_finish[MAX\_TASKS];

有了以上的材料，下一頁開始實作schedule()

實作schedule()時，我順便做了用來顯示當前task狀態的表

void schedule()

{

    int max = 0;

    int temp = 0;

    int final = 0;

    printf("\n|   task   ");

    for(int i = 0; i< MAX\_TASKS; i++)

    {

        //顯示所有task編號

        printf("| task %d ", i);

    }

    printf("|\n------------------------------------------------------------------------------------------------------");

    printf("\n| priority ");

    //掃描整個 context\_priority[] 陣列

    for(int i = 0; i< MAX\_TASKS; i++)

    {

        //顯示所有task的優先級

        printf("|   %d    ", context\_priority[i]);

        //找到優先度最高的task，並且必須是沒有被做過的

        if( context\_priority[i] >= temp && task\_finish[i] != 1)

        {

            temp = context\_priority[i];

            //將最優先要做的task編號設給 max 變數

            max = i;

        }

    }

    printf("|\n------------------------------------------------------------------------------------------------------");

    printf("\n|  finish  ");

    for(int i = 0; i< MAX\_TASKS; i++)

    {

        //顯示所有task是否被執行過的狀態

        printf("|   %d    ", task\_finish[i]);

    }

    printf("|\n\n");

    //找到優先度最低的task

for(int i = 0; i< MAX\_TASKS; i++)

    {

        if( context\_priority[i] <= temp )

        {

            temp = context\_priority[i];

            //將最低的task編號設給 final 變數

            final = i;

        }

    }

    //如果 context\_priority[final]為0 且 task\_finish[final] 也是0，代表所有任務都已經輪過一遍

    if(context\_priority[final] == 0 && task\_finish[final] == 1)

    {

        //將所有優先度不為 0 的 task 的完成狀態都重設為 0

        for(int i = 0; i< MAX\_TASKS; i++)

        {

            if(context\_priority != 0)

            {

                task\_finish[i] = 0 ;

            }

        }

        printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  task reset  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

    }

    //不是的話代表還有task沒做完，於是將下個 task 的 context 指針設為 max 所代表的task 然後switch\_to()

    else

    {

        printf("max = task %d\n\n\n", max);

        struct context \*next = &ctx\_tasks[max];

        task\_finish[max] = 1; //將該task的task\_finish設為1，代表已經做了

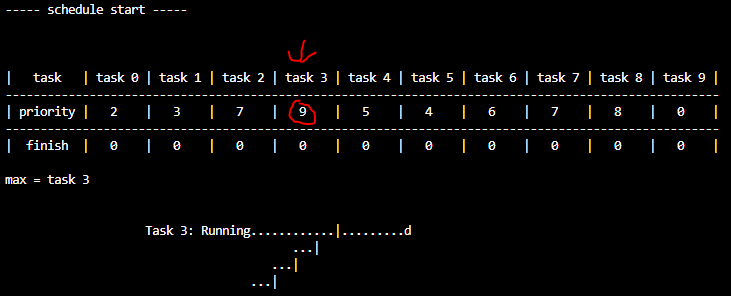
        switch\_to(next);

    }

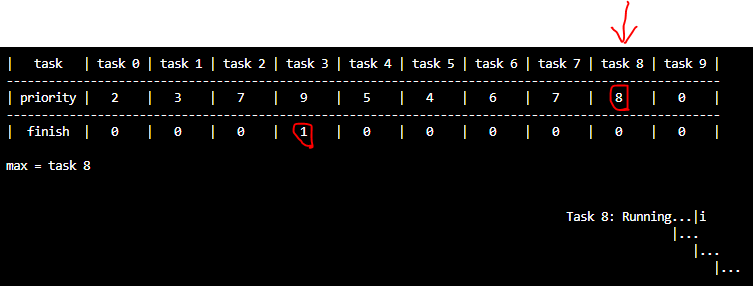
}

編譯執行看看

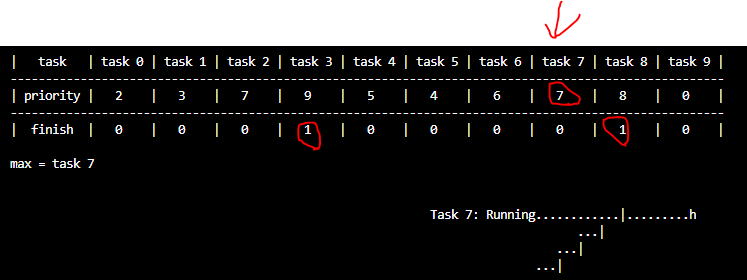
第一個執行的是task3 (此處我以數字大的為優先，因為用0來做為最後一個執行的優先度程式碼會比較簡潔)



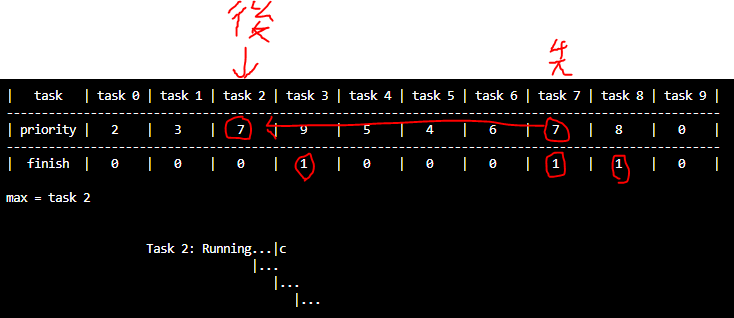
第二個執行的是task8，同時可以看到task3的finish被標為1了

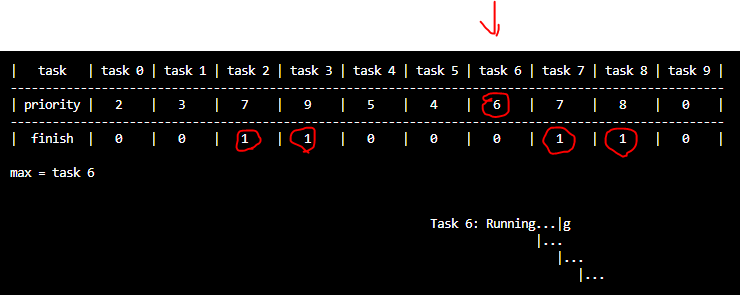


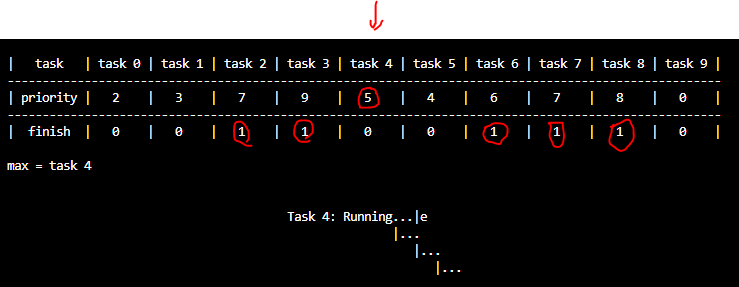
以下依序

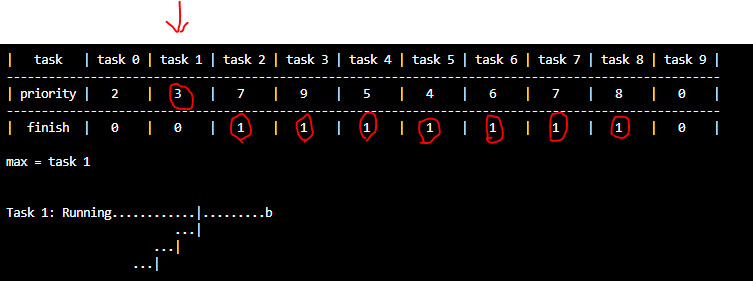


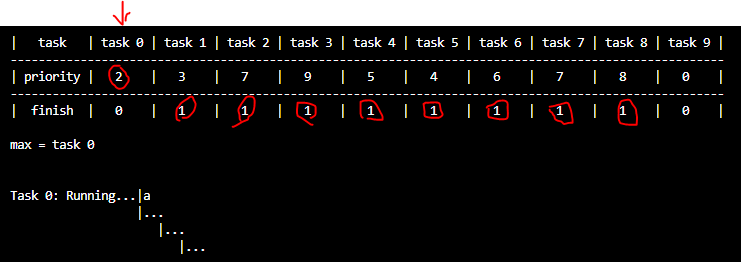
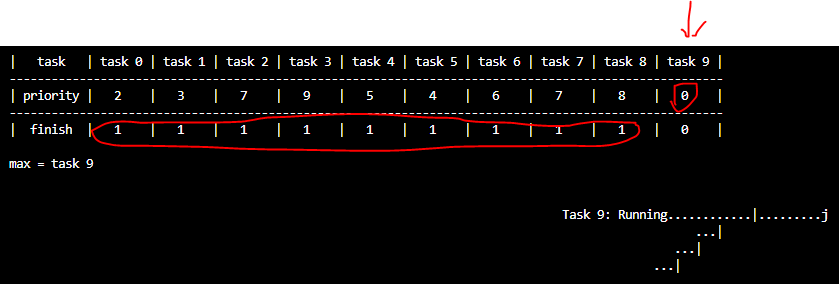
如果兩個task同級，因為用掃描搜尋陣列的特性，在陣列後方的會先做，也就是task7 -> task2



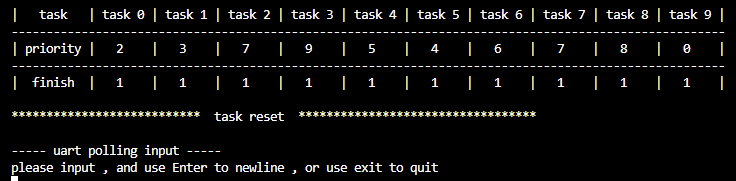


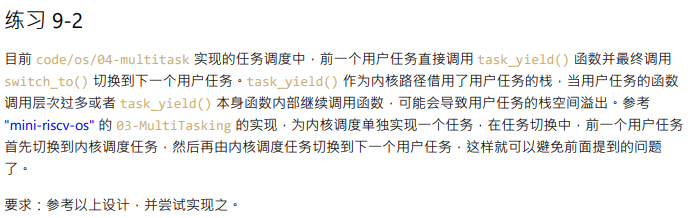




全部任務完成後便重設所有finish狀態。為了測試，這個schedule() 還沒有使用無限循環，所以會離開並進到kernel裡的下一個程式。





用9-1的作業來延伸，這次選擇用task0來當作內核調度任務

//將所有task初始化

void schedule\_init()

{

    //優先把schedule() 設為task0，每個task離開後都會先進 schedule()

    ctx\_tasks[\_top].sp = (reg\_t) &task\_stack[\_top][STACK\_SIZE - 1];

    ctx\_tasks[\_top].ra = (reg\_t) schedule;

    \_top++;

    task\_create(user\_task1, "a", 2);

    task\_create(user\_task2, "b", 3);

    task\_create(user\_task3, "c", 7);

    task\_create(user\_task4, "d", 9);

    task\_create(user\_task5, "e", 5);

    task\_create(user\_task6, "f", 4);

    task\_create(user\_task7, "g", 6);

    task\_create(user\_task8, "h", 7);

    task\_create(user\_task9, "i", 8);

}

Task\_exit() 用switch\_to跳回schedule()

void task\_exit()

{

    switch\_to( &ctx\_tasks[0]);

}

下一頁是 schedule()

void schedule()

{

    int max = 0;

    int temp = 0;

    printf("\n|   task   ");

    for(int i = 0; i< MAX\_TASKS; i++)

    {

        //顯示所有task編號

        printf("| task %d ", i);

    }

    printf("|\n------------------------------------------------------------------------------------------------------");

    printf("\n| priority ");

    //掃描整個 context\_priority[] 陣列

    for(int i = 0; i< MAX\_TASKS; i++)

    {

        //顯示所有task的優先級

        printf("|   %d    ", context\_priority[i]);

        //找到優先度最高的task，並且必須是沒有被做過的

        if( context\_priority[i] >= temp && task\_finish[i] != 1)

        {

            //將最優先要做的task編號設給 max 變數

            temp = context\_priority[i];

            max = i;

        }

    }

    printf("|\n------------------------------------------------------------------------------------------------------");

    printf("\n|  finish  ");

    for(int i = 0; i< MAX\_TASKS; i++)

    {

        //顯示所有task是否被執行過的狀態

        printf("|   %d    ", task\_finish[i]);

    }

    printf("|\n\n");

    //如果 context\_priority[max]為0 且 task\_finish[max] 也是0，代表所有任務都已經輪過一遍

    if(context\_priority[max] == 0 && task\_finish[max] == 0)

    {

        for(int i = 0; i< MAX\_TASKS; i++)

        {

            //將所有優先度不為 0 的 task 的完成狀態都重設為 0

            if(context\_priority != 0)

            {

                task\_finish[i] = 0 ;

            }

        }

        printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  task reset  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

    }

    //不是的話代表還有task沒做完，於是將下個 task 的 context 指針設為 max 所代表的task 然後switch\_to()

    else

    {

        printf("max = task %d\n\n\n", max);

        struct context \*next = &ctx\_tasks[max];

        //將該task的task\_finish設為1，代表已經做了

        task\_finish[max] = 1;

        switch\_to(next);

    }

}

可以看到，這次的schedule() 更優雅，不需要檢查所有的任務是否做完，最後的任務一定是task0，這時候就能輕易判斷並重設所有狀態。而且這時的context，ra暫存器內的地址就是schedule()本身，所以重設完離開schedule() 後又自動跳回schedule() 開始下一個輪迴，因此不需要在schedule() 內使用無限循環。

下面的圖可以看到重設完後又自動進入新的輪迴

