# Exercise #10 REPORT

**Name: 송예린**

**Student ID: 2171023**

1. **Exercise Objective (10%)**

- Task 의 기능을 이해하고 Handler 를 통해 Priority 를 변경할 수 있다.
- Priority 를 이해하고 Priority 를 이용해 Task 를 실행시킬 수 있다.

### 1.1 이론적 배경

Real- time 시스템에서 가장 먼저 고려해야 하는 건 correctness 이고 그 다음이 시간이다. 만약 Correctness 는 문제가 없지만 시간 안에 완성을 하지 않는다면 코드를 수정해야 한다. Real time 에서 가장 중요시되는 요소는 deadline 이다. Aperiodic 은 이벤트를 기준으로 Task 가 발생하는 것이고 Periodic Tasks 는 주기적으로 Task 가 발생함을 의미한다. 등록된 여러 개의 Task 들이 Priority 에 따라서 Scheduling 되는 것이다. Preemptive 는 Task 가 실행 중일 때 다른 Task 가 실행 중인 Task 를 멈추고 Task 를 돌릴 수 있는 것을, Non-Preemptive 는 Task 가 실행 중이라면 다른 Task 는 Task 를 멈추지 못하고 순서를 기다려야 함을 의미한다. 이 실험에서는 Periodic Tasks, Non-Preemptive 를 실습할 예정이다.Task 는 return 되지 않고 task 의 함수를 stack 에 저장한 후 Pointer 를 이용해 함수가 반복된다 필요한 task 의 함수를 적으면 함수의 Pointer 를 넘겨주게 된다. 디버깅 때 확인하기 위해서 Task 의 이름도 지정한다. 스택 사이즈를 설정해주고 Task 에게 넘겨주고 싶은 인자를 적어준다. Priority 를 Task 의 priority 를 정하고 만약 실행시킬 Task 가 1 개라면 handler 에게 NULL 을 전달해준다. 만약 1 개가 아닌 여러 개의 Task 를 실행시켜야 한다면 Handler 에게 Task1Handler 를 전달하고, Handler 를 통해 다른 Task 에서 Task 의 Priority 를 변경할 수 있다 Handler 에게 Task1Handler 를 전달한다는 것은 포인트를 통해서 Task 의 정보를 Handler 에게 전달한다는 것이다.

2. **Implementation (60%)**

Port1 에 빨간색 LED 를 출력하기 위해서 P1 DIR Register pin 0 에 1bit 을 주고 Port2 를 출력모드로 사용하기 위해서 P1 DIR Register pin 0,1,2 에 1bit 을 준다. Task 1,Task2, Task3, Task4 를 만들고 Task 는 값을 반환하지 않고 함수가 끝나도 Task 의 시작위치를 가리키게 된다. Priority 를 Task 1 이 9, Task2 가 8, Task 3 7, Task4 6 을 갖도록 한다. Task1 이 실행되면 P1 OUT Register pin 0 에 1 을 넣어서 빨간색 LED 를 키고 딜레이를 주고 다시 끄고 딜레이를 준다. vTaskPrioritySet()은 Handler 를 통해서 다른 Task 에 접근한 후 Priority 를 변경한다. vTaskPrioritySet()으로 Task1 Priority 를 5 로 낮추면 Priority 8 인 그 다음 Task2 가 실행된다. Task2 에서 LED 를 깜빡이고 Task2 Priority 를 4 로 낮추면 Priority 7 인 Task3 이 실행된다. Task3 에서 LED 를 깜빡이고 Task3 Priority 를 3 으로 낮추면 Priority 가 6 인

Task4 가 실행된다. 실행된 Task4 에 서 LED 를 깜빡인 후 Priority 가 Task 1 이 9, Task2 가 8, Task 3 7, Task4 6 을 갖도록 한다.

실행된 Task 가 가장 낮은 Priority 를 갖게 되므로 그 다음 Task 가 실행되고 마지막 Take 는 다시 첫번째 Task 부터 순서대로 높은 Priority 를 갖게 설정한다. 이렇게 하면 반복문을 사용하지 않고 연속으로 Task 가 실행시킬 수 있다.

**3. Discussion & Conclusion (10%)**

반복문을 사용하지 않고 Task 를 번갈아 가면서 실행시키는 법을 학습하였다. Task 를 등록하고 Priority 를 설정해주면 높은 Priority 의 Task 만 실행되기 때문에 Task 가 끝난 후 Handler 로 다른 Task 의 Priority 를 변경해줘야 한다.

만약 if 문을 삽입하여 조건이 만족할 때만 Priority 가 바뀌도록 한다면 특정 조건에서만 Priority 가 바뀌서 다른 함수가 실행될 것으로 예상된다. 즉 이벤트를 기준으로 Task 가 발생하는 Aperiodic 을 할 수 있는 것이다.

예를 들어 5 분이 지나면 화면이 검정색으로 변하는 구현을 Priority 를 이용한다면 화면이 검정색이 되는 함수의 Priority 를 낮게 두다가 interrupt 가 없이 5 분이 지나면 Priority 가 높아져서 화면이 꺼지는 것이다. 화면이 검정색으로 변하는 함수는 interrupt 가 발생(기기를 건드림)하면 다시 Priority 가 낮아진다.

다른 예시로 엘리베이터 버튼을 각 층에서 동시에 누르면 가까운 층부터 멈추는데 이 또한 Priority 로 생각이 가능할 것 같다. interrupt 가 발생하면 엘리베이터와 가까운 층부터 순서대로 Priority 를 높게 설정한다.(현재 층이 가장 높은 Priority 를 가짐) 그 후 현재층의 priority 를 1, 안 눌린 층들의 Priority 를 0 으로 설정한 후 register flag 에서 수행된 핀번호를 0 으로 만들고 Priority 대로 층을 이동한다. 층을 이동 후 register flag 가 모두 0 이 아니라면 도착층의 priority 를 0 으로 낮춘다. 움직이는 방향도 고려해야 하는데, 맨 처음 버튼이 눌린 층을 기준으로, 도착할 층에서 현재층을 뺀 값이 양수면(if) (현재층+1) 층이 현재층 다음으로 가장 높은 Priority 를 갖고 1 층이 가장 낮은 Priority 를 갖는다 반대로 (도착층-현재층)이 음수면 (현재층-1)층이 현재층 다음으로 가장 높은 Priority 를 갖고 꼭대기층이 가장 낮은 Priority 를 갖는다. 즉 엘리베이터는 버튼을 누르지 않으면 Priority 가 높은 그 자리에 멈춰 있다. 예시를 즉흥적으로 생각해서 모순이 많을 것으로 예상되고 구현도 어려울 것이라고 생각된다. 엘리베이터가 각 층을 방문할 때마다 버튼이 눌린 것을 확인하지 않고 미리 Priority 를 정하는 이유는 이동속도를 높이기 위해서이다. 이동할 때마다 체크하는 것 보다 미리 정하는 것이 더 빠르다고 여길 것이다.

**4. Reference(s)**

## 5. Code (20%)

**FILE NAME (ex: main.c)**(수정 후 최종코드)

```
/* Driver configuration */
#include <ti/devices/msp432p4xx/inc/msp.h>
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>
#include <ti/drivers/Board.h>

/* RTOS header files */
#include <FreeRTOS.h>
#include <task.h>
#include <semphr.h>

TaskHandle_t Task1Handle,Task2Handle,Task3Handle,Task4Handle;

/* function prototypes */
void Task1( void *pvParameters );
void Task2( void *pvParameters );
void Task3( void *pvParameters );
void Task4( void *pvParameters );



void main(void)
{
    /* Configuring S1 & S2 buttons / Red LED & RGB LED in mainboard */
    P1->DIR = BIT0;
    P1->DIR &= ~(BIT1|BIT4);
    P1->DIR|=1<<0;
    P2->DIR |= 1<<2 | 1<< 1 | 1<<0;
    P1->REN |= (BIT1|BIT4);
    P1->OUT |= (BIT1|BIT4);
    P2->OUT = 0x00;

    xTaskCreate( Task1,                    /* The function that implements the task. */
                 "Task 1",                 /* The text name assigned to the task - for debug only as it is
not used by the kernel. */
                 configMINIMAL_STACK_SIZE,/* The size of the stack to allocate to the task. */
                 ( void * ) 0,             /* The parameter passed to the task - just to check the
functionality. */
                 9,                       /* The Priority assigned to the task. */
                 & Task1Handle );                  /* The task handle is not required, so NULL is
passed. */
    xTaskCreate( Task2,                    /* The function that implements the task. */
```

```
                "Task 2",               /* The text name assigned to the task - for debug only as it is
not used by the kernel. */
                configMINIMAL_STACK_SIZE,/* The size of the stack to allocate to the task. */
                ( void * ) 0,           /* The parameter passed to the task - just to check the
functionality. */
                8,                      /* The Priority assigned to the task. */
                & Task2Handle );                /* The task handle is not required, so NULL is
passed. */
    xTaskCreate( Task3,                 /* The function that implements the task. */
                "Task 3",               /* The text name assigned to the task - for debug only as it is
not used by the kernel. */
                configMINIMAL_STACK_SIZE,/* The size of the stack to allocate to the task. */
                ( void * ) 0,           /* The parameter passed to the task - just to check the
functionality. */
                7,                      /* The Priority assigned to the task. */
                & Task3Handle );                /* The task handle is not required, so NULL is
passed. */
    xTaskCreate( Task4,                 /* The function that implements the task. */
                "Task 4",               /* The text name assigned to the task - for debug only as it is
not used by the kernel. */
                configMINIMAL_STACK_SIZE,/* The size of the stack to allocate to the task. */
                ( void * ) 0,           /* The parameter passed to the task - just to check the
functionality. */
                6,                      /* The Priority assigned to the task. */
                & Task4Handle );                /* The task handle is not required, so NULL is
passed. */

    /* Start the tasks and timer running. */
    vTaskStartScheduler();

    while (1)
    {
    }

}
void Task1( void *pvParameters )
{
    int i;
    while (1)
    {
        P1->OUT = BIT1|BIT4 | BIT0;
        for (i=0;i<100000;i++);
        P1->OUT = BIT1|BIT4;
        for (i=0;i<100000;i++);
        vTaskPrioritySet(Task1Handle,4);
    }
}
```

```
    void Task2( void *pvParameters )
                {
                    int i;
                    while (1)
                    {
                       P2->OUT = BIT0;
                       for (i=0;i<100000;i++);
                       P2->OUT = 0x00;
                       for (i=0;i<100000;i++);

                       vTaskPrioritySet(Task2Handle,3);
                    }
                }
 void Task3( void *pvParameters )
        {
            int i;
            while (1)
            {
               P2->OUT = BIT1;
               for (i=0;i<100000;i++);
               P2->OUT =0x00;
               for (i=0;i<100000;i++);
               vTaskPrioritySet(Task3Handle,2);
            }
        }
void Task4( void *pvParameters )
            {
                int i;
                while (1)
                {
                   P2->OUT = BIT2;
                   for (i=0;i<100000;i++);
                   P2->OUT = 0x00;
                   for (i=0;i<100000;i++);
                   vTaskPrioritySet(Task1Handle,9);
                   vTaskPrioritySet(Task2Handle,8);
                   vTaskPrioritySet(Task3Handle,7);
                   vTaskPrioritySet(Task4Handle,6);
                }
            }
```

**FILE NAME (ex: main2.c)(채점받은 코드)**

```
    int i;
    while (1)
    {
       P1->OUT = BIT1|BIT4 | BIT0;
```

```
        for (i=0;i<100000;i++);
        P1->OUT  = BIT1|BIT4;
        for (i=0;i<100000;i++);
        vTaskPrioritySet(Task2Handle,9);
        vTaskPrioritySet(Task3Handle,8);
        vTaskPrioritySet(Task4Handle,7);
        vTaskPrioritySet(Task1Handle,6);
    }
}
    void Task2( void *pvParameters )
                {
                    int i;
                    while (1)
                    {
                       P2->OUT  =  BIT0;
                       for (i=0;i<100000;i++);
                       P2->OUT  = 0x00;
                       for (i=0;i<100000;i++);
                       vTaskPrioritySet(Task3Handle,9);
                       vTaskPrioritySet(Task4Handle,8);
                       vTaskPrioritySet(Task1Handle,7);
                       vTaskPrioritySet(Task2Handle,6);
                    }
                }
 void Task3( void *pvParameters )
        {
            int i;
            while (1)
            {
               P2->OUT  = BIT1;
               for (i=0;i<100000;i++);
               P2->OUT  =0x00;
               for (i=0;i<100000;i++);
               vTaskPrioritySet(Task4Handle,9);
               vTaskPrioritySet(Task1Handle,8);
               vTaskPrioritySet(Task2Handle,7);
               vTaskPrioritySet(Task3Handle,6);
            }
        }
void Task4( void *pvParameters )
            {
                int i;
                while (1)
                {
                   P2->OUT  =  BIT2;
                   for (i=0;i<100000;i++);
                   P2->OUT  = 0x00;
```

```
            for (i=0;i<100000;i++);
            vTaskPrioritySet(Task1Handle,9);
            vTaskPrioritySet(Task2Handle,8);
            vTaskPrioritySet(Task3Handle,7);
            vTaskPrioritySet(Task4Handle,6);
        }
    }
```

실습이 끝난 후 수정한 부분은 빨간색으로 표시했습니다.

Code 1 은 최종 코드이고 Code2 는 채점 받은 코드입니다.

도착할 층 – 가장 높은 Priority

움직이는 중에 버튼을 누른다면 연산을 끝낼 때 엘리베이터 위치 기준으로 Priority 를 정한다

Priority 의 순서를 정하고 Priority 가 8 인 층을 체크 후 방향 계산하고 다시 Priority 를 변경해준다.

꼭대기층이나 맨 아래 층을 찍으면 엘리베이터의 방향이 바뀌고 priority 도 방향 기준으로 다시 설정된다.

S1,S2 Button 을 사용하려면 GPIO Port 1 의 pin 1, pin4 를 입력모드로 바꿔줘야 하므로 P1 DIR Register pin 1,4 만 0bit 을 준다. 이때 P1 의 출력모드가 풀렸을 수도 있으므로 다시 P1 DIR pin 0 에 1bit 을 다시 넣어준다. P1 REN Register pin 1,pin4 에 1bit 을 줘서 pull up/down 을 활성화한 후 P1 OUT Register pin 1,pin4 에 1bit 을 줘서 pull up 을 선택해준다.