HW1 ucos 실행 및 코드 분석

G201871003 박연희

1. 개발환경 세팅 및 컴파일

[Windows 10 세팅]

먼저 Turbo c++ 5.0 기준 Windows 10으로 컴파일 및 실행 모두 불가하였음.

[Windows XP 세팅 및 Turbo 4.5 IDE 사용]

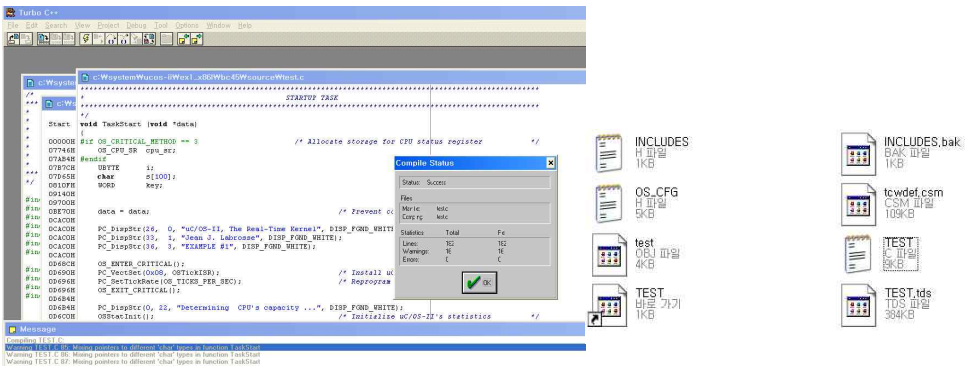
Turbo 4.5 설치, Virtual machine 환경에서 Windows XP 설치하여 빌드 및 실행 진행

- 설치한 컴파일러 기준 source\Include.h에서 사용하는 경로 설정

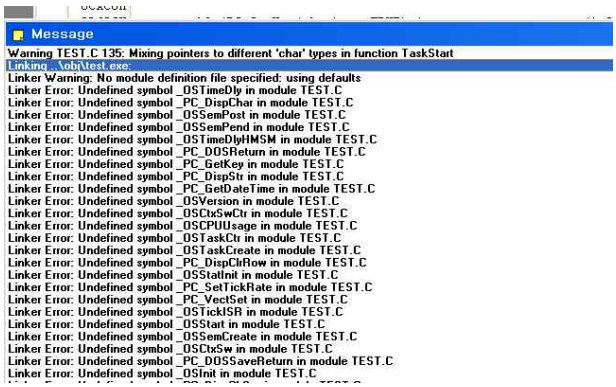
```
#include    <stdio.h>
#include    <string.h>
#include    <ctype.h>
#include    <stdlib.h>
#include    <conio.h>
#include    <dos.h>
#include    <setjmp.h>

#include    "C:\SYSTEM\UCOS-II\ix86l\bc45\os_cpu.h"
#include    "os_cfg.h"
#include    "C:\SYSTEM\blocks\pc\bc45\pc.h"
#include    "C:\SYSTEM\UCOS-II\source\ucos_ii.h"
```

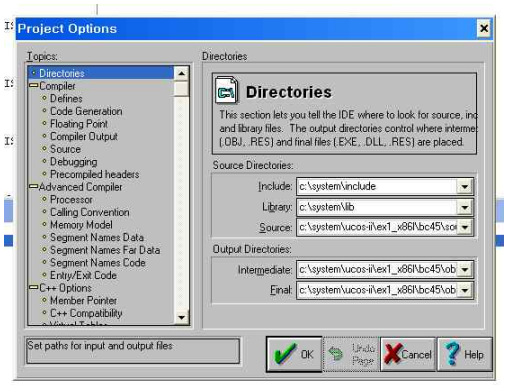
- 소스코드 컴파일 및 오브젝트 파일 생성 확인



- 코드 동작 확인 (DEBUG -> RUN)시 링커 오류 발생



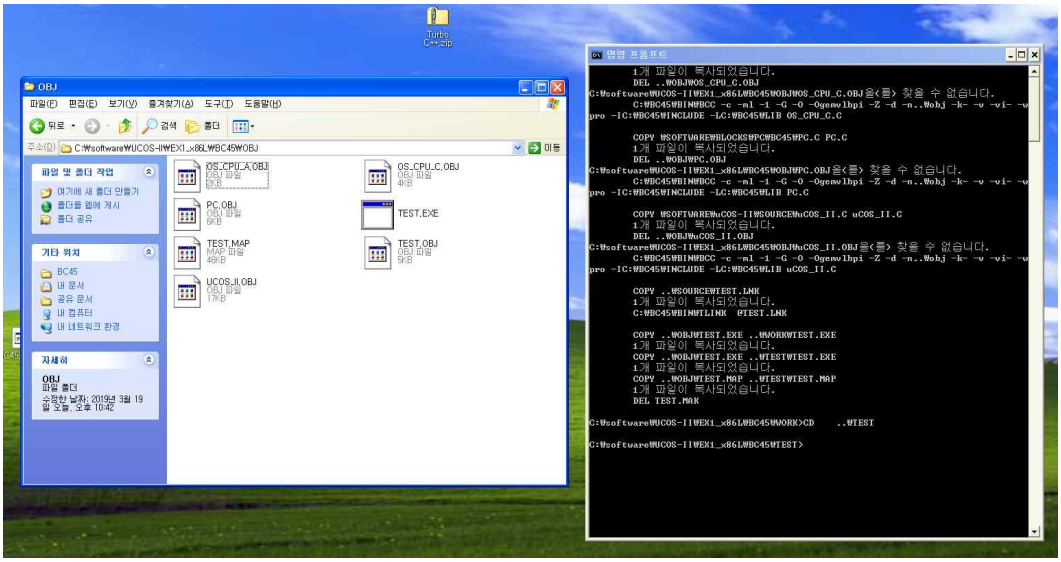
* LIB, include, source file에 대한 환경 설정은 해주었으나 링커 파일 연결이 제대로 안 된 것으로 추정.
Turbo c++에서 링커 파일을 링킹해주는 방법에 대한 숙지 필요.



[BC45 컴파일러 다운로드 후 파일 실행]

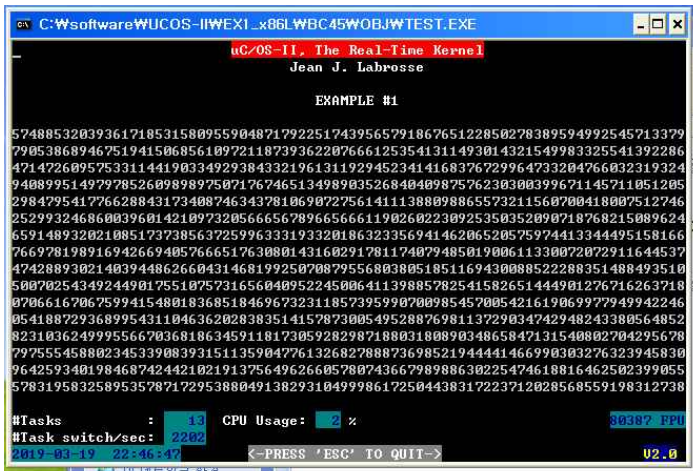
참고자료

1. https://electronics.tistory.com/106 에서 제공하는 BC45 컴파일러 설치.
2. 교수님께서 제공해주신 ucos2 소스코드 실행
3. 빌드 및 실행파일 생성 성공



2. 소스코드 실행 및 소스코드 분석

[TEST1]



*** 기능 요약**

프로그램 실행시 10개의 Task가 생성되는데, 각 Task들은 Random한 위치(x,y)에 자신의 Task Number에 해당하는 숫자를 출력하는 프로그램이다.

*** 프로그램 설명**

```
void main (void)
{
    PC_DispClrScr(DISP_FGND_WHITE + DISP_BGND_BLACK);      /* Clear the screen                            */
    OSInit();                                              /* Initialize uC/OS-II                         */
    PC_DOSSaveReturn();                                    /* Save environment to return to DOS           */
    PC_VectSet(uCOS, OSCtxSw);                             /* Install uC/OS-II's context switch vector    */
    RandomSem    = OSSemCreate(1);                         /* Random number semaphore                     */
    TaskFPUFlag = _8087;
    OSTaskCreate(TaskStart, (void *)0, &TaskStartStk[TASK_STK_SIZE - 1], 0);
    OSStart();                                             /* Start multitasking                          */
}
```

먼저 Main 함수에서는 uC/os-II 초기화 및 Semaphore 생성 후 OSTaskCreate 함수를 통해 실제 Task를 생성할 TaskStart를 생성한다.

TaskStart 내부에서는 OSTaskCreate 함수를 통해 N_Tasks (=10) 개의 Task를 생성하게 되고, 각각의 Task는 Task Number에 해당하는 우선순위 및 숫자를 부여 받는다.
cf. 숫자 부여는 ASCII 표현시 숫자 0 에 해당하는 '0'에 Task 숫자를 더하는 방식을 이용.

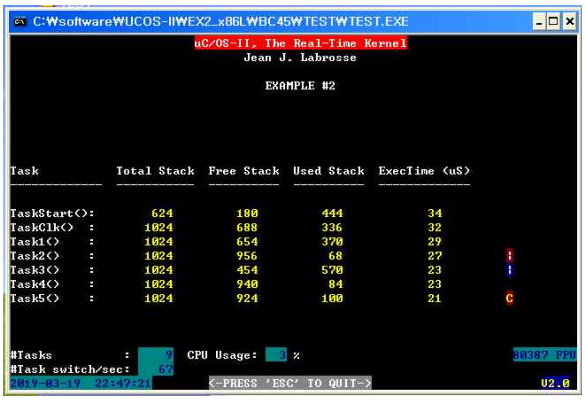
```
for (i = 0; i < N_TASKS; i++) {                        /* Create N_TASKS identical tasks          */
    TaskData[i] = '0' + i;                             /* Each task will display its own letter   */
    OSTaskCreate(Task, (void *)&TaskData[i], &TaskStk[i][TASK_STK_SIZE - 1], i + 1);
}
```

Task 내부에서는 먼저 각각 Task의 위치를 생성할 때 OSSempend 함수를 통해 Task를 wait 상태로 만들어 숫자를 표출할 위치를 얻어오게 하고, 위치를 얻은 후에는 OSSempost를 통해 Task를 릴리즈 시켜 받아온 랜덤한 값을 PC_DispChar 함수를 통해 화면에 표출할 수 있게 한다.

```
void Task (void *data)
{
    UBYTE x;
    UBYTE y;
    UBYTE err;


    for (;;) {
        OSSemPend(RandomSem, 0, &err);          /* Acquire semaphore to perform random numbers        */
        x = random(80);                         /* Find X position where task number will appear      */
        y = random(16);                         /* Find Y position where task number will appear      */
        OSSemPost(RandomSem);                   /* Release semaphore                                  */
                                                /* Display the task number on the screen              */
        PC_DispChar(x, y + 5, *(char *)data, DISP_FGND_LIGHT_GRAY);
        OSTimeDly(1);                           /* Delay 1 clock tick                                 */
    }
}
```

[TEST2]



*** 기능 요약**

서로 다른 기능을 가지는 6개의 Task를 생성하여 각각의 기능이 동작하는 동안 할당된 스택의 사이즈가 어떻게 변화하는지 표출하는 프로그램.

각각의 기능 명세는 아래와 같음

1) TaskStart : Task1~5 및 TaskClk를 생성하는 Task

2) Taskclk : OS_TICKS_PER_SEC(200 Tick) 마다 PC_GetDateTime()함수를 통해 현재 시간을 얻어 표출하는 task

```
for (;;) {
    PC_GetDateTime(s);
    PC_DispStr(0, 24, s, DISP_FGND_BLUE + DISP_BGND_CYAN);
    OSTimeDly(OS_TICKS_PER_SEC);
}
```

3) Task1 : 100ms 마다 실행되며, 모든 Task(7)의 실행시간을 측정하여 각각의 Task에 대한 Stack 사용량(Total stack, Free Stack, Used stack, exec time)을 표출해주는 task

```
for (;;) {
    for (i = 0; i < 7; i++) {
        PC_ElapsedStart();
        err  = OSTaskStkChk(TASK_START_PRIO+i, &data);
        time = PC_ElapsedStop();
        if (err == OS_NO_ERR) {
            sprintf(s, "%4ld        %4ld        %4ld        %6d",
                    data.OSFree + data.OSUsed,
                    data.OSFree,
                    data.OSUsed,
                    time);
            PC_DispStr(19, 12+i, s, DISP_FGND_YELLOW);
        }
    }
    OSTimeDlyHMSM(0, 0, 0, 100);                    /* Delay for 100 mS                    */
}
```

4) Task2 : 화면에 10 tick 마다 '|', '/', '-', '\\' 문자를 순차적으로 표출하는 task

```
for (;;) {
    PC_DispChar(70, 15, '|',  DISP_FGND_WHITE + DISP_BGND_RED);
    OSTimeDly(10);
    PC_DispChar(70, 15, '/',  DISP_FGND_WHITE + DISP_BGND_RED);
    OSTimeDly(10);
    PC_DispChar(70, 15, '-',  DISP_FGND_WHITE + DISP_BGND_RED);
    OSTimeDly(10);
    PC_DispChar(70, 15, '\\', DISP_FGND_WHITE + DISP_BGND_RED);
    OSTimeDly(10);
}
```

5) Task3 : Task2와 같이 20 tick 마다 '|', '\\', '-', '/' 특정 문자를 순차적으로 표출하나, task2와는 다르게 불필요한 500byte 만큼의 공간을 추가로 할당 받아 사용한다.

```
for (i = 0; i < 499; i++) {        /* Use up the stack with 'junk'
    dummy[i] = '?';
}
for (;;) {
    PC_DispChar(70, 16, '|',  DISP_FGND_WHITE + DISP_BGND_BLUE);
    OSTimeDly(20);
    PC_DispChar(70, 16, '\\', DISP_FGND_WHITE + DISP_BGND_BLUE);
    OSTimeDly(20);
    PC_DispChar(70, 16, '-',  DISP_FGND_WHITE + DISP_BGND_BLUE);
    OSTimeDly(20);
    PC_DispChar(70, 16, '/',  DISP_FGND_WHITE + DISP_BGND_BLUE);
    OSTimeDly(20);
}
```

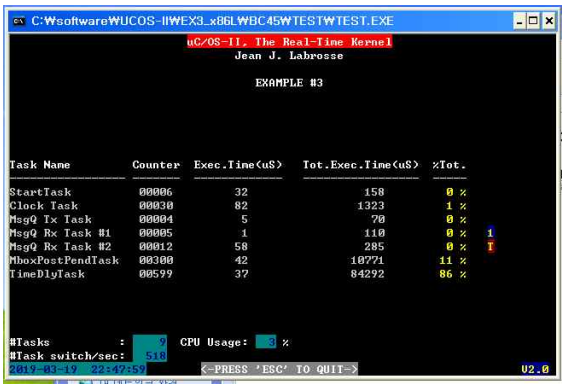
6) Task4 : Task4와 task5는 mailbox 통신을 하며, Task4에서는 A부터 Z까지 하나의 문자열이 담긴 메세지를 Task5에 보내며 Task5가 정상적으로 수신되었는지 확인할 때 까지(ack 수신할때까지) 대기한다.

```
txmsg = 'A';
for (;;) {
    while (txmsg <= 'Z') {
        OSMboxPost(TxMbox, (void *)&txmsg);  /* Send message to Task #5                  */
        OSMboxPend(AckMbox, 0, &err);        /* Wait for acknowledgement from Task #5   */
        txmsg++;                             /* Next message to send                    */
    }
    txmsg = 'A';                             /* Start new series of messages            */
}
```

7) Task5 : Task4와 task5는 mailbox 통신을 하며, Task5에서는 task4에서 문자열이 올 때 까지 대기하며, 해당 문자열을 표출하고 1초 대기 후 task4에게 ack 메세지를 전송한다.

```
for (;;) {
    rxmsg = (char *)OSMboxPend(TxMbox, 0, &err);                /* Wait for message from Task #4 */
    PC_DispChar(70, 18, *rxmsg, DISP_FGND_YELLOW + DISP_BGND_RED);
    OSTimeDlyHMSM(0, 0, 1, 0);                                  /* Wait 1 second                 */
    OSMboxPost(AckMbox, (void *)1);                             /* Acknowledge reception of msg  */
}
```

[TEST3]



*** 기능 요약**

서로 다른 기능을 가지는 6개의 Task를 생성하여 각각의 기능에 따른 실행 시간이 어떻게 차이나는지 표출하는 프로그램.

각각의 기능 명세는 아래와 같으며, 각 Task의 우선순위는 아래의 순서로 부여.

1) TaskStart : Task1~5 및 TaskClk를 생성하는 Task

2) Taskclk : OS_TICKS_PER_SEC(200 Tick) 마다 PC_GetDateTime()함수를 통해 현재 시간을 얻어 표출하는 task

3) MsgQ Tx Task : 메시지 큐에 메시지 생성. 생성할 메시지는 1, 2, 3 (ASCII 문자)로 각 1, 1.5초, 2.5초 마다 생성됨.

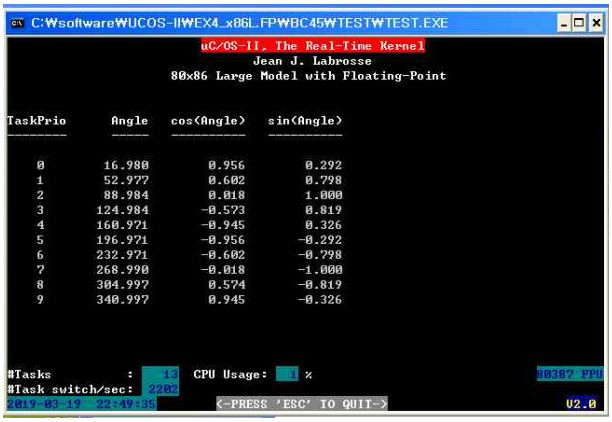
4) MsgQ Rx Task #1 : 메시지 큐에 메시지가 들어있는지 확인 후, 메시지 발생시 이를 수신하여 화면에 출력. 이 Task의 메시지 대기 시간은 무한대임.

5) MsgQ Rx Task #2 : 메시지 큐에 메시지 들어있는지 화인 후 메시지 발생시 이를 수신하며, 대기 시간은 250ms로 할당. 250ms내에 할당 받지 못한 경우는 Timeout으로 화면에 'T' 표출.

6) MboxPostPendTask : 10ms마다 메일 박스로 메시지를 송신하고 바로 수신.

7) TimeDlyTask : 1초 딜레이하는 task. task 실행시 바로 waiting 상태로 들어가나 타 Task가 먼저 실행되고 Task 큐가 비워지기 전까지 이 Task는 실행되지 못하므로 이로 인한 지연시간이 굉장히 크게 됨.

[TEST4]



*** 기능 요약**

10개의 task가 생성되어 task별 할당된 우선순위(0~9)가 고유 숫자가 되어, 이 값을 10배로 한 것을 각도로 하여 sin값과 cos값을 구해 출력하는 프로그램. 각도는 매 반복마다 0.01씩 증가하는 형태.

```
ypos  = *(INT8U *)data + 8;
angle = (FP32)(*(INT8U *)data * 36.0);
for (;;) {
    radians = 2.0 * 3.141592 * angle / 360.0;
    x       = cos(radians);
    y       = sin(radians);
    sprintf(s, "    %d      %8.3f     %8.3f     %8.3f", *(INT8U *)data, angle, x, y);
    PC_DispStr(0, ypos, s, DISP_FGND_LIGHT_GRAY);
    if (angle >= 360.0) {
        angle /= 360.0;
    } else {
        angle +=   0.01;
    }
    OSTimeDly(1);
}
```