**강의명: 실시간 커널**

**실습 번호: 1**

**실습 제목: Multi tasks(멀티 태스크)**

**학생 이름: 황귀훈**

**학번: 201710885**

**1. Simple multi tasks**

**1.1**

작성한 프로그램은 TaskStart(), TaskTime(), Task1(), ..., Task5() 총 7개의 함수를 작성했다.

먼저 TaskStart() 함수는 다음과 같이 작성했다.

void TaskStart(void \*pdata)

{

OSSoCInit(); // The first task should call this.

OSStatInit();

OSTaskCreate(TaskTime, (void\*)0, &TaskTimeStk[TASK\_STK\_SIZE - 1], 20);

OSTaskCreate(Task1, (void\*)0, &Task1Stk[TASK\_STK\_SIZE - 1], 21);

OSTaskCreate(Task2, (void\*)0, &Task2Stk[TASK\_STK\_SIZE - 1], 22);

OSTaskCreate(Task3, (void\*)0, &Task3Stk[TASK\_STK\_SIZE - 1], 23);

OSTaskCreate(Task4, (void\*)0, &Task4Stk[TASK\_STK\_SIZE - 1], 24);

OSTaskCreate(Task5, (void\*)0, &Task5Stk[TASK\_STK\_SIZE - 1], 25);

UARTputclear();

OSTaskDel(10);

}

프로그램을 이렇게 작성한 이유는 다음과 같다. TaskStart()함수는 제일 먼저 생성되어 수행되는 task로 TaskTime(), Task1(), ..., Task5()를 생성한 후, 함수 UARTputclear()를 불러서 화면을 clear 한 후, 커널 함수 OSTaskDel()을 불러서 종료하는 함수이다. 그렇기 때문에 먼저 OSTaskCreate()함수를 이용하여 6개의 TaskTime(), Task1(), ..., Task5()를 만들었다. 이때 OSTaskCreate()함수는 4개의 인수(task함수, (void \*) 0, task stack의 top주소, task priority (0~61))를 전달받아 task를 생성한다.

그 다음에 화면을 clear 하기 위해 UARTputclear()를 호출했다. 마지막으로 TaskStart()의 priority인 10을 인수로 OSTaskDel(10) 함수를 호출하여 종료했다.

그 다음으로 TaskTime()함수를 작성했는데 다음과 같다.

void TaskTime(void \*pdata)

{

for (;;) {

OSTimeDly(100);

UARTprintf("\nTime=%d: ", OSTimeGet()/100);

}

}

프로그램을 이렇게 작성한 이유는 다음과 같다. TaskTime()함수는 1초(=100 ticks)를 WAIT한 후 화면에 "\nTime=%d: " 형식으로 커널이 수행된 후 경과한 시간(sec)을 출력하는 일을 무한 반복 수행하는 함수이다. 먼저 이 함수는 무한히 수행되기 때문에 for(;;)문안에 프로그램을 작성했다.

이 함수는 1초(=100 ticks)을 기다리기 때문에 OSTimeDly()함수에 인수로 100ticks(=1 sec)을 주어 호출했다. Wait 한 후에는 화면에 "\nTime=%d: " 형식으로 커널이 수행된 후 경과한 시간(sec)을 출력해야 되기 때문에 ‘tm4c1294xl.h’를 참고하여 UARTprintf() 문을 이용해 표준 출력을 수행했다.

경과 시간을 알기 위해 현재 tick을 알게 해주는 시스템 호출 함수 OSTimeGet()을 사용했고 1 sec는 100 tick이기 때문에 100으로 나누어서 초 단위로 출력했다.

그 다음으로 taskN() 함수를 작성했는데 다음과 같다.

void Task1(void \*pdata)

{

for (;;) {

OSTimeDly(100);

UARTprintf("Task1 ");

}

}

void Task2(void \*pdata)

{

for (;;) {

OSTimeDly(200);

UARTprintf("Task2 ");

}

}

void Task3(void \*pdata)

{

for (;;) {

OSTimeDly(300);

UARTprintf("Task3 ");

}

}

void Task4(void \*pdata)

{

for (;;) {

OSTimeDly(400);

UARTprintf("Task4 ");

}

}

void Task5(void \*pdata)

{

for (;;) {

OSTimeDly(500);

UARTprintf("Task5 ");

}

}

taskN()에서 1 <= N <= 5 이고 각 task들은 N초 wait한 후 화면에 "TaskN "을 출력하는 일을 무한 반복 수행한다. 프로그램을 이렇게 작성한 이유는 이 함수는 무한 반복 수행하기 때문에 for(;;)안에 프로그램을 작성했고 N초를 wait해야 되기 때문에 시스템 호출 함수 OSTimeDly()에 인수로 N\*100tick(N sec = N\*100 ticks)을 주어 호출했고 화면에 "TaskN "을 출력하기 위해 ‘tm4c1294xl.h’를 참고하여 UARTprintf() 문을 이용해 표준 출력을 수행했다.

**1.2**

**테이블이(가) 표시된 사진

자동 생성된 설명**

Time=1에서 Time=19까지 프로그램 수행 화면 결과이다.

**1.3**

Time=1에서 Time=19까지 프로그램 수행이 본 실습이 의도한 대로 바르게 수행되었는 지 확인하는 방법은 프로그램이 수행됨에 따라 출력되는 “Time= %d “과 그때 출력되는 TaskN (1<=N<=5)들을 비교해 봄으로써 의도한 대로 바르게 수행되었는 지 확인할 수 있다. TaskTime()에 의해 “Time= “은 매초 출력되고 각각의 TaskN (1<=N<=5)들은 N초만큼 Wait 한 뒤에 "TaskN "을 출력한다. 그렇기 때문에 “Time= %d: “과 Time의 값(흐른 시간=%d)의 약수 번에 해당하는 taskN이 한 줄에 출력되어야 한다. 또한 “Time= “을 출력하는 TaskTime()과 "TaskN "을 출력하는 각각의 taskN()의 priority를 비교해 보면 TaskTime() > Task1() > Task2() > Task3() > Task4() > Task5() (20>21>22>23>24>25, 숫자가 낮을수록 priority가 높다.)이기 때문에 이 순서대로 출력 되어야한다. **1.2**의 프로그램 수행 결과를 보면 본 실습이 의도한 대로 바르게 수행되었음을 확인할 수 있다. (ex: Time=12일 때 Time=12: Task1 Task2 Task3 Task4 순서대로 출력, 1 2 3 4는 12의 약수이고 priority가 높은 task 부터 수행됨)

**1.4**

taskN의 priority를 비교하면 Task1 > Task2 > Task3 > Task4 > Task5 이기 때문에 priority가 높은 task부터 수행되어 time=10일 때 Task1 Task2 Task5 순서로 출력된다.(**1.3**에 따라 1 2 5는 10의 약수 이고 priority가 높은 Task 순으로 출력되어 올바르게 수행 되었음을 확인할 수 있다.)

**1.5**

Time=60: Task1 Task2 Task3 Task4 Task5

Time이 60일 때 처음으로 TaskN이 모두 수행된다.

**1.6**

**텍스트, 화이트보드이(가) 표시된 사진

자동 생성된 설명**

Time=1에서 Time=4까지 TaskN(여기서 1<=N<=5)들의 수행을 슬라이드 1-25와 유사한 그림으로 표현하였다. 그림에서 가로 축 단위는 시간(초)으로, 세로 축에는 각 TaskN(총 5 개)을, TaskN이 RUN한 시간은 검정색 막대기로 표현했다.

**1.7**

이번 실습에서 작성한 프로그램은 시간에 따라 정확히 수행되는 실시간 프로그램이었는데 task를 만드는데 priority를 다르게 주어 수행 순서를 제어할 수 있었고 OSTimeDly() 함수를 이용해 시간을 제어하는 것을 쉽게 할 수 있었다. Task를 만들거나 시간을 제어할 때 사용되는 함수들을 보면 실시간 커널에서 제공하는 서비스(커널 함수)를 사용하여 짧은 코드로 짤 수 있었다. 이번 경험을 통해 실시간 커널을 사용하면 주기적으로 수행하는 실시간 멀티 태스킹 프로그래밍이 간단 해진다는 것을 알게 되었다.

**2. Multi tasks의 priority**

**2.1**

수정한 Task의 priority는 다음과 같다.

Priority priority

Task1 21 -> 25

Task2 22 -> 24

Task3 23 -> 23

Task4 24 -> 22

Task5 25 -> 21

위와 같이 priority를 수정한 이유는 실습1의 번호2번의 Time=1에서 Time=19까지 프로그램 수행결과를 보면 출력되는 TaskN이 Task5 Task4 Task3 Task2 Task1순으로 출력 되고 있음을 알 수 있다.

(ex: Time=10: Task5 Task2 Task1, Time=12: Task4 Task3 Task2 Task1) Task의 수행 순서는 priority가 높은(숫자가 낮은) 순서이기 때문에 TaskN의 priority를 오름차순에서 내림차순으로 수정했다.

**2.2**

**테이블이(가) 표시된 사진

자동 생성된 설명**

Time=1에서 Time=19까지 프로그램 수행 화면 결과이다.

**2.3**

**텍스트, 화이트보드이(가) 표시된 사진

자동 생성된 설명**

Time=1에서 Time=4까지 TaskN(여기서 1<=N<=5)들의 수행을 슬라이드 1-25와 유사한 그림으로 표현하였다. 그림에서 가로 축 단위는 시간(초)으로, 세로 축에는 각 TaskN(총 5 개)을, TaskN이 RUN한 시간은 검정색 막대기로 표현했다.

**2.4**

TaskN은 기본적으로 N초 동안 WAIT 상태가 되었다가 READY상태가 된다. state가 READY가 된 Task들이 존재하고 있다가 priority가 높은 순서대로 실행하게 된다. 실습1의 2번에서 수정한 것은 priority밖에 없기 때문에 Task1~Task5의 실행 순서만 변경되었을 뿐 각 Task의 실시간 성과 주기는 변화되지 않는다. 실제로 **1.6**의 그림과 **2.3**의 그림을 비교해보면 priority를 변화하기 전과 후 모두 1초 안에 5개의 Task가 모두 run 하고 있고 매 Time 마다 실행되는 Task의 변화(주기의 변화)는 없다.

**3. Multi tasks 응용**

**3.1**

번호 3번 Multi tasks 응용은 먼저 TaskStart(), Task0, ..., Task9() 11개의 Task를 생성하기 때문에 다음과 같이 11개의 task함수를 정의했다.

void TaskStart(void \*pdata);

void Task0(void \*pdata);

void Task1(void \*pdata);

void Task2(void \*pdata);

void Task3(void \*pdata);

void Task4(void \*pdata);

void Task5(void \*pdata);

void Task6(void \*pdata);

void Task7(void \*pdata);

void Task8(void \*pdata);

void Task9(void \*pdata);

그리고 나서 TaskStart() 함수는 다음과 같이 작성했다.

void TaskStart(void \*pdata)

{

unsigned char i;

OSSoCInit(); // The first task should call this.

OSStatInit();

////////////////////////////////////////////////////////////////////////// 직접 작성한 부분

OSTaskCreate(Task0,(void\*)0,&TaskStk[0][TASK\_STK\_SIZE-1],20);

OSTaskCreate(Task1,(void\*)0,&TaskStk[1][TASK\_STK\_SIZE-1],21);

OSTaskCreate(Task2,(void\*)0,&TaskStk[2][TASK\_STK\_SIZE-1],22);

OSTaskCreate(Task3,(void\*)0,&TaskStk[3][TASK\_STK\_SIZE-1],23);

OSTaskCreate(Task4,(void\*)0,&TaskStk[4][TASK\_STK\_SIZE-1],24);

OSTaskCreate(Task5,(void\*)0,&TaskStk[5][TASK\_STK\_SIZE-1],25);

OSTaskCreate(Task6,(void\*)0,&TaskStk[6][TASK\_STK\_SIZE-1],26);

OSTaskCreate(Task7,(void\*)0,&TaskStk[7][TASK\_STK\_SIZE-1],27);

OSTaskCreate(Task8,(void\*)0,&TaskStk[8][TASK\_STK\_SIZE-1],28);

OSTaskCreate(Task9,(void\*)0,&TaskStk[9][TASK\_STK\_SIZE-1],29);

///////////////////////////////////////////////////////////////////////////

refresh:

UARTputclear();

UARTprintf("Real-time kernel: random tasks");

UARTputgoto(0, 20);

UARTprintf("Type any char to refresh the screen.");

for (;;) {

if (UARTgetc\_nb() != -1)

goto refresh;

UARTputgoto(0, 1);

UARTprintf("Ticks=%d, Tasks=%d, CPU=%d%% ", OSTimeGet(), OSTaskCtr,

OSCPUUsage);

OSTimeDly(100);

}

}

프로그램을 이렇게 작성한 이유는 다음과 같다. TaskStart()함수는 제일 먼저 생성되어 수행되는 task로 Task0(), ..., Task9()를 생성한 후 화면 상단에 아래 수행 화면과 같이 현재 시간(tick), 총 thread 수, CPU Usage %를 출력하고 화면 하단에 아래 수행 화면과 같이 "Type any char to refresh the screen"을 출력하는 일을 100 tick 주기로 무한 반복 수행하는 함수이다. 따라서 먼저 OSTaskCreate()함수를 이용하여 11개의TaskTime(),Task0(),...,Task9()를 만들었다. 이때 OSTaskCreate()함수는 4개의 인수(task함수, (void \*) 0, task stack의 top주소, task priority (0~61))를 전달받아 task를 생성한다. 화면 상단에 아래 수행 화면과 같이 현재 시간(tick), 총 thread 수, CPU Usage %를 출력하고 화면 하단에 아래 수행 화면과 같이 "Type any char to refresh the screen"을 출력하는 일을 하기 위해서 커널 함수인 UARTputgoto(), UARTprintf(), OSTimeGet()과 커널 변수인 OSTaskCtr, OSCPUUsage를 사용했다. 또한 TaskStart()는 키보드에서 어떤 입력이 들어오면 화면을 다시 refresh하는 역할을 하기 때문에 함수 UARTgetc\_nb()을 사용했다.

그 다음으로 화면 상단에 출력된 시간(tick), 총 thread 수, CPU Usage %와 화면 하단에 출력된 "Type any char to refresh the screen"사이에 TaskN()(여기서 0<=N<=9)함수가 화면(여기서 0<=x=49, 3<=y<=18) 내의 random 위치 (x, y)에 random color(여기서 30<=color<=37)로 문자 N을 5 tick 주기로 출력하는 일을 무한 반복하여 수행하도록 다음과 같이 함수 TaskN()을 작성했다.

void Task0(void \*pdata)

{

for(;;){

OSTimeDly(5);

int x,y;

unsigned short color;

x = OS\_GetRandom(50);

y = OS\_GetRandom(16)+3;

color = OS\_GetRandom(8)+30;

UARTputgoto(x,y);

UARTputcolor(color);

UARTputc('0');

}

}

위의 함수는 TaskN() (여기서 0<=N<=9)중 첫 번째로 Task0()함수이다. 나머지 Task1~Task9 함수들과의 차이는 마지막에 출력하는 문자 N 뿐이므로 Task0() 함수로만 설명하겠다.

Task0()함수를 위와 같이 작성한 이유는 우선 이 함수가 5 tick을 주기로 무한 반복 수행 하기 때문에 for(;;)문 안에 프로그램을 작성하였고 OSTimeDly()함수에 인수로 5 tick을 주어 호출 했다.

주어진 범위 내의 랜덤한 위치에 랜덤한 색으로 문자 N을 출력해야 되기 때문에 ‘tm4c1294xl.h’과 ‘utility.h’을 참고하여 0보다 크거나 같고 n보다 작은 랜덤 번호를 리턴하는 OS\_GetRandom(n)함수에 인수를 적당히 조절하여 호출했고 원하는 범위내의 랜덤 위치를 각각 x, y에 저장했다. 또한 원하는 범위내의 랜덤 color 값 역시 OS\_GetRandom(n) 함수를 이용하여 color 변수에 저장했다. 커서를 화면의 위치 (x, y)로 보내기 위해 UARTputgoto(x,y)를 사용했고, 출력할 문자의 색을 변경하기 위해 UARTputcolor(color)함수를 사용했다. 이때 인수로 들어가는 값의 color 매크로에 따라 색이 결정된다. 랜덤 위치에 랜덤 한 색으로 문자 N (여기서 N은 TaskN에서 N, Task 번호)을 출력하기 위해 문자 c를 출력하는 UARTputc('0')함수를 이용해 N을 출력했다. (Task6은 문자N으로 6을 출력한다.)

+ OS\_GetRandom(n) 함수는 0 <= y <= n-1 사이의 랜덤한 값 y를 리턴 하기 때문에 3<=y<=18 의 범위를 만족 시키려면 y = OS\_GetRandom(16)+3 => 0+3 <= y <= 15+3 다음을 y에 저장하면 된다.

**3.2**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

프로그램 수행 결과 화면으로 실습 설명에 있는 프로그램의 화면과 일치한다.

끝.