멀티미디어 시스템 Lab6 실습보고서

2011104043 장원엽

1. 수업 내용

오늘은 멀티테스킹에 대하여 배웠다. 멀티테스킹이란 이름 그대로 여러 개의 테스크들을 수행하는 것이다. 멀티테스킹을 위해서는 스케쥴링과 컨텍스트 스위칭 등의 기능이 필요하며 동기화를 위한 세마포어와 뮤텍스 문제도 고려해야 한다.

uC/OS-II에서는 최대 64개의 작업을 만들 수 있다. 테스크 ID는 해당 테스크의 우선순위를 나타내며 uC/OS-II는 고정 우선순위를 사용하고 있다. 테스크 0번이 가장 높은 우선순위를 가지는 작업이다. 가장 낮은 우선순위의 작업은 idle task로, OS에서 예약되어있는 작업이다. 또한 idle task보다 한 단계 높은 우선순위를 가지는 작업은 통계 작업으로 예약되어있다.

uC/OS-II의 스케쥴링 알고리즘은 우선순위에 따라 작업을 스케쥴링하며, 그 속도는 O(1)로, 효과적으로 자료구조를 사용하였음을 알 수 있다.

uC/OS-II는 다른 OS들과 마찬가지로 크게 Ready, Running, Waiting의 세가지 상태로 나뉠 수 있다.

1. 실습

다음은 멀티테스킹을 이용하여 FND와 LED를 동시에 제어하기 위한 소스코드이다.

#include "includes.h"

#define F\_CPU 16000000UL // CPU frequency = 16 Mhz

#include <avr/io.h>

#include <util/delay.h>

#define TASK\_STK\_SIZE OS\_TASK\_DEF\_STK\_SIZE

#define N\_TASKS 2

OS\_STK TaskStk[N\_TASKS][TASK\_STK\_SIZE]; // 스텍 2개를 기본 스텍 사이즈로 만든다.

void LedTask(void \*data); // 각각의 작업들을 정의할 함수를 선언한다.

void FndTask(void \*data);

int main (void)

{

OSInit();

OS\_ENTER\_CRITICAL();

TCCR0=0x07;

TIMSK=\_BV(TOIE0);

TCNT0=256-(CPU\_CLOCK\_HZ/OS\_TICKS\_PER\_SEC/ 1024);

OS\_EXIT\_CRITICAL();

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

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

// 항상 실행되어야 하는 FND의 우선순위를 0번으로 주고, 시간 간격이 있는 LED의 우선순위를 1로 줌으로서 원활하게 실행될 수 있도록 한다.

OSStart();

return 0;

}

void LedTask (void \*data)

{

data = data;

DDRA = 0xff;

while (1) {

PORTA = 0xaa;

OSTimeDlyHMSM(0, 0, 0, 500); // What happens if \_delay\_ms(500) ?

//\_delay\_ms(500);

// LED 제어 코드에서 OSTimeDlyHMSM를 쓴 이유는, LED가 500ms를 기다리는 동안 FND 작업을 수행하기 위함이다. 만약 \_delay\_ms를 사용했다면 LED작업 내부에서 500ms만큼 기다리기 때문에 FND 작업이 수행될 수 없다.

PORTA = 0x55;

OSTimeDlyHMSM(0, 0, 0, 500);;

//\_delay\_ms(500);

}

}

void FndTask (void \*data)

{

unsigned char FND\_DATA[ ]= {

0x3f, // 0

0x06, // 1

0x5b, // 2

0x4f, // 3

0x66, // 4

0x6d, // 5

0x7d, // 6

0x27, // 7

0x7f, // 8

0x6f, // 9

0x77, // A

0x7c, // B

0x39, // C

0x5e, // D

0x79, // E

0x71, // F

0x80, // .

0x40, // -

0x08 // \_

};

unsigned int num=0, num0, num1, num2, num3;

data = data;

DDRC = 0xff;

DDRG = 0x0f;

while(1) {

num++;

num3 = (num / 1000) % 10;

num2 = (num / 100) % 10;

num1 = (num / 10) % 10;

num0 = num % 10;

PORTC = FND\_DATA[num3];

PORTG = 0x08;

//OSTimeDlyHMSM(0, 0, 0, 2);

\_delay\_ms(2);

// 여기서는 OSTimeDlyHMSM을 사용하지 않고 \_delay\_ms를 사용했는데, 그 이유는 OS자체의 클럭의 간격이 2ms보다 크기 때문에 OS를 통해 2ms만큼의 시간 제어는 불가능하기 때문이다. 이떄문에 기기적으로 시간을 제어하기 위하여 \_delay\_ms를 사용한다. 만약 OSTimeDlyHMSM를 쓴다면 딜레이가 걸리지 않게 되어 LED코드가 수행되지 않을 것이다.

PORTC = FND\_DATA[num2] | FND\_DATA[16];

PORTG = 0x04;

//OSTimeDlyHMSM(0, 0, 0, 3);

\_delay\_ms(3);

PORTC = FND\_DATA[num1];

PORTG = 0x02;

//OSTimeDlyHMSM(0, 0, 0, 2);

\_delay\_ms(2);

PORTC = FND\_DATA[num0];

PORTG = 0x01;

//OSTimeDlyHMSM(0, 0, 0, 3);

\_delay\_ms(3);

}

}

1. 실습 응용

실습 응용 2. 실습의 내용을 OS를 이용하지 않고 while문 안에 적절히 배치하는 것이다. 이것은 LED 소스코드에서 500ms를 기다리는 코드가 존재하는데, 기다리는 대신에 500ms동안 FND를 제어하는 데에 사용하면 2. 실습의 동작과 같은 코드를 작성할 수 있다. 아래는 while문 내용이다.

while (1) {

PORTA = 0xaa;

//\_delay\_ms(500);

// 500ms를 기다리는 대신에, 500ms동안 FND를 제어한다.

for(int i=0; i<50; i++){

num++;

num3 = (num / 1000) % 10;

num2 = (num / 100) % 10;

num1 = (num / 10) % 10;

num0 = num % 10;

PORTC = FND\_DATA[num3];

PORTG = 0x08;

\_delay\_ms(2);

PORTC = FND\_DATA[num2] | FND\_DATA[16];

PORTG = 0x04;

\_delay\_ms(3);

PORTC = FND\_DATA[num1];

PORTG = 0x02;

\_delay\_ms(2);

PORTC = FND\_DATA[num0];

PORTG = 0x01;

\_delay\_ms(3);

}

PORTA = 0x55;

//\_delay\_ms(500);

for(int i=0; i<50; i++){

num++;

num3 = (num / 1000) % 10;

num2 = (num / 100) % 10;

num1 = (num / 10) % 10;

num0 = num % 10;

PORTC = FND\_DATA[num3];

PORTG = 0x08;

\_delay\_ms(2);

PORTC = FND\_DATA[num2] | FND\_DATA[16];

PORTG = 0x04;

\_delay\_ms(3);

PORTC = FND\_DATA[num1];

PORTG = 0x02;

\_delay\_ms(2);

PORTC = FND\_DATA[num0];

PORTG = 0x01;

\_delay\_ms(3);

}

}