**임베디드 소프트웨어 Lab7 실습보고서**

2011104043 장원엽

1. 수업 내용

OS에서는 여러 가지 task들을 동시에 수행할 수 있다. 이를 멀티테스킹이라고 한다. 그런데 원활한 멀티테스킹을 하기 위해서는 OS가 제공해야 할 몇 가지 작업이 있다. 그 중 한가지가 동기화 문제이다. 만약 동기화 문제가 해결되지 않는다면 여러 가지 task들이 동시에 컴퓨터 자원에 접근하면서 프로그램에 큰 오류를 만들어낼 수 있다. 동기화 문제를 해결하기 위한 방법으로는 세마포어와 뮤텍스를 포함하여 이외에도 여러 가지 방법이 있다.

세마포어란 임계영역을 보호하기 위하여 키를 가지고 있는 프로세스만이 임계영역에 접근할 수 있도록 제어하기 위한 도구이다. uC/OS-II에서는 OSSemCreate 함수로 세마포어를 만들고 OSSemPend 함수로 임계영역에 접근하며 OSSemPost 함수로 임계영역에서의 작업이 끝났음을 알린다. 하나의 프로세스에서의 작업이 끝나 OSSemPost 함수가 호출되면 다른 프로세스에서 OSSemPend 함수로 세마포어를 잡아 임계영역에 접근할 수 있는 권한을 얻는 원리이다.

세마포어와 비슷하게 뮤텍스라는 것이 있다. 뮤텍스의 사용 방법은 세마포어의 사용 방법과 매우 비슷하다. 단지 함수 이름의 Sem이라는 부분이 Mutex로 바뀌고 함수의 인자 형식이 약간 바뀌는 수준이다. 뮤텍스는 OSMutexCreate 함수로 생성한다. 이후 임계영역에 접근하기 위하여 OSMutexPend 함수로 뮤텍스를 가져온다. 뮤텍스는 하나의 프로세스가 뮤텍스를 가져와서 임계영역에서 작업을 진행하는 동안에 다른 프로세스들이 임계영역에 접근할 수 없도록 한다. 임계영역에서의 작업이 끝났다면 OSMutexPost 함수를 호출하여 다른 프로세스들에게 임계영역에서의 작업이 끝났음을 알려준다. 이후 다른 프로세스들은 마찬가지로 OSMutexPend 함수를 이용하여 임계영역에의 접근을 시도하는 방식이다.

1. 실습

아래 코드는 실습 내용 Lab. 6)의 FND task를 분초 시계로 변경하고, LED task에서 1초마다 수행시키도록 semaphore를 사용하여 동기화(보다 정확한 시계)의 각 작업의 정의 부분만을 복사하여 붙여 넣은 것이다.

void LedTask (void \*data)

{

data = data;

Sem = OSSemCreate(0); // 이 부분에서 세마포어를 만들어 낸다.

DDRA = 0xff;

while (1) {

PORTA = 0xaa;

OSTimeDlyHMSM(0, 0, 0, 500);

PORTA = 0x55;

OSTimeDlyHMSM(0, 0, 0, 500);

OSSemPost(Sem); // 작업이 끝나고 세마포어를 넘겨준다.

}

}

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;

INT8U err;

data = data;

DDRC = 0xff;

DDRG = 0x0f;

PORTC = FND\_DATA[0];

PORTG = 0x01;

while(1) {

OSSemPend(Sem, 0, &err); // 세마포어를 기다린다. Led작업에서 OSSemPost(Sem) 함수가 호출되기 전까지는 아래 코드가 수행될 수 없다.

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];

PORTG = 0x04;

\_delay\_ms(3);

PORTC = FND\_DATA[num1];

PORTG = 0x02;

\_delay\_ms(2);

PORTC = FND\_DATA[num0];

PORTG = 0x01;

\_delay\_ms(3);

}

}

아래 코드는 실습 응용 부분의 각 task들의 정의 부분 소스 코드이다.

void LedTask (void \*data)

{

data = data;

Sem = OSSemCreate(1); // 세마포어를 생성한다.

DDRA = 0xff;

while (1) {

PORTA = 0xaa;

OSTimeDlyHMSM(0, 0, 0, 500);

PORTA = 0x55;

OSTimeDlyHMSM(0, 0, 0, 500);

OSSemPost(Sem); // LED작업을 끝내고 세마포어를 넘겨준다.

}

}

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 // \_

};

INT8U err;

data = data;

DDRC = 0xff;

DDRG = 0x0f;

PORTC = FND\_DATA[0];

PORTG = 0x01;

INT8U err2;

while(1) {

OSSemPend(Sem, 0, &err); // 세마포어를 받아와야 아래 부분의 소스가 실행될 수 있다.

OSMutexPend(mutex, 0, &err2); // 뮤텍스를 받아와야 마찬가지로 아래 부분의 소스가 실행될 수 있다.

num++;

num3 = (num / 1000) % 10;

num2 = (num / 100) % 10;

num1 = (num / 10) % 10;

num0 = num % 10;

OSMutexPost(mutex); // 작업을 마치고 뮤텍스를 돌려준다.

}

}

void FndDisplayTask (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 // \_

};

INT8U err;

data = data;

DDRC = 0xff;

DDRG = 0x0f;

PORTC = FND\_DATA[0];

PORTG = 0x01;

while(1) {

OSMutexPend(mutex, 0, &err); // 뮤텍스를 받아와야 아래부분 코드가 수행된다.

PORTC = FND\_DATA[num3];

PORTG = 0x08;

\_delay\_ms(2);

PORTC = FND\_DATA[num2];

PORTG = 0x04;

\_delay\_ms(3);

PORTC = FND\_DATA[num1];

PORTG = 0x02;

\_delay\_ms(2);

PORTC = FND\_DATA[num0];

PORTG = 0x01;

OSTimeDlyHMSM(0, 0, 0, 10);

OSMutexPost(mutex); // 작업 수행을 마치고 뮤텍스를 돌려준다.

}

}