2015104124
전자공학과
진우빈

IoT 소프트웨어 과제13

예제 1)

키패드 모듈에서 받는 입력을 통해 Fnd모듈을 제어하는 실습입니다. 저번 실습에서 xQueue를 이용하여 InterTask Communication을 하는 방법과 달리 이번에는 전역변수와 Semaphore를 이용해 Fnd모듈에 키패드 값을 전달합니다. 먼저 키패드에 4개의 입력을 받아 각각 1, 2, 3, 4의 번호가 FndModule로 전달될 수 있게 합니다. 인터럽트를 발생시켜야 하므로 아두이노의 인터럽트 핀인 2, 3, 21, 20번 핀에 키패드 모듈을 연결하였으며, FndPin과 FndSelectPin에도 각각 6개와 8개의 출력핀을 설정하였습니다.

SemaPhorerHandle_t 로 Sem이라는 세마포어를 선언하였고 이후에 Setup에서 BinarySemaPhore를 생성하게 됩니다. 즉 Mutex를 생성하여 하나의 인터럽트 값만이 세마포어를 통해 제어되게 합니다. KeypadControl함수는 각각 인터럽트에 걸린 번호에 따라서 해당하는 값이 전역변수인 SendValue에 복사되게 합니다. 복사가 되면 Semaphore에 Wait을 걸어 0으로 설정되어있는 Semaphore가 1 증가해서 1이 됩니다. 즉 이후에 Signal(Take)를 사용할 수 있는 환경을 만들어 주는 것입니다. KeypadTask에서는 while문을 통해서 게속 대기하다가 Semphore값이 1이 돼서 xSemaphoreTake의 return값이 1이 되면 SendValue 값을 ShiftInsert 함수의 매개변수로 전달하게 됩니다. ShiftInsert는 FndModule의 0번 즉, 맨 오른쪽에 있는 SelectPin에 전달받은 data값을 넣어주는 함수입니다. 성공적으로 Signal이 되면 Semaphore의 값을 다시 0으로 바꿔주게 됩니다. 이후에 FndDisplay 함수와 FndSelect로 진행되는 FndTask 함수를 통해서 Fnd 번호의 출력이 일어나게 됩니다. FndTask는 KeypadTask와 다르게 병렬적으로 수행되는 태스크로서 Setup에서 별도로 스케쥴링을 통해 두 가지 Task가 순서에 따라 실행될 수 있게 하였습니다.

컴파일 후 실행결과 1, 2, 3, 4번 키패드를 누를 때 마다 해당 번호와 일치하는 값이 Fnd모듈의 오른쪽부터 삽입되는 것을 확인할 수 있었습니다. xQueue 외에도 SemaPhore를 통해서 전역변수 값을 바꿔줌으로써 데이터 전달이 가능한 것을 확인할 수 있는 예제였습니다.

```
#include <FreeRTOS_AVR.h>
#define MS2TICKS(ms) (ms / portTICK_PERIOD_MS)
#define FND_SIZE 6

const int Keypad[4]={2,3,21,20};
const int FndSelectPin[6]={22,23,24,25,26,27};
const int FndPin[8]={30,31,32,33,34,35,36,37};
const int FndFont[10]={0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x60, 0x7D, 0x07, 0x7F, 0x67};

SemaphoreHandle_t Sem;
int SendValue = 0;
int Fnd[FND_SIZE]={0,};

void ShiftInsert(int data) {
  int i;
  for(i=1; i<FND_SIZE; i++) {
    Fnd[FND_SIZE-i]=Fnd[FND_SIZE-i-1];
  }
  Fnd[0]=data;
}
```

```
void setup() {
  int i;
  for(i=0; i<6; i++) {
    pinMode(FndSelectPin[i], OUTPUT);
  }

  for (i=0; i<8; i++) {
    pinMode(FndPin[i], OUTPUT);
  }

  for(i=0; i<4; i++) {
    pinMode(Keypad[i], INPUT);
  }

  attachInterrupt(0, KeyPadControl1, RISING);
  attachInterrupt(1, KeyPadControl2, RISING);
  attachInterrupt(2, KeyPadControl3, RISING);
  attachInterrupt(3, KeyPadControl4, RISING);

  vSemaphoreCreateBinary(Sem);

  xTaskCreate(KeypadTask, NULL, 200, NULL, 2, NULL);
  xTaskCreate(FndTask, NULL, 200, NULL, 1, NULL);
  vTaskStartScheduler();
}
```

<lab3-4_1.c 코드 주요 부분>

과제 1)

예제 1에서 Fnd모듈을 제어하는 것과 달리 Motor 모듈을 제어하는 실습입니다. 저번 실습인 lab3-3_1과 똑같은 기능을 구현하는 것이지만 이번엔 xQueue가 아니라 Semaphore를 이용하여 구현하는 과제입니다. 모터를 제어하기 위해 아두이노의 9, 10번 핀에 모터모듈의 왼쪽회전과, 오른쪽회전을 할 수 있게 연결하였으며, 아두이노 메가 보드의 0,1,2 인터럽트 번호에 해당하는 2, 3, 21 핀번호에 키패드를 연결하였습니다. 그리고 SemaphoreHandle_t를 통해 Semaphore를 선언하고 Setup에서 BinarySemaphore인 Mutex를 생성하였습니다. 실습처럼 Left, Right, StopKeyControl 함수를 통해 전역변수인 SendValue값을 바꾸고 Semaphore의 값을 1 더해줍니다. MotorTask로 바로 값을 전달할 것이기 때문에 별도의 KeypadTask 함수는 만들지 않았으며 SendValue의 값을 통해서 MotorTask의 stack 변수인 receiveValue에 값을 옮기게 됩니다. 당연히 Semaphore가 1이라서 값을 Take할 수 있을 때만 Sendvalue값이 복사가 되며 복사된 값은 if문을 통해 각각 상황에 맞게 제어됩니다. lab3-3_1의 실습에서 xQueue와 관련된 함수를 Semaphore를 통해 구현한 것입니다. 이후 각 입출력 핀을 Setup에서 설정해주었고 사실 상 하나의 태스크만 수행되는 스케쥴러를 생성하였습니다.

컴파일 후 실행결과 1번을 누르면 왼쪽으로 모터가 돌아가게되고 2번을 누르면 정지, 3번을 누르면 오른쪽으로 모터가 돌아가는 것을 확인할 수 있었습니다.

```
SemaphoreHandle_t Sem;
int SendValue = 0;

void LeftKeyControl() {
  delay(50);
  SendValue = 1;
  xSemaphoreGive(Sem);
}

void StopKeyControl() {
  delay(50);
  SendValue = 2;
  xSemaphoreGive(Sem);
}

void RightKeyControl() {
  delay(50);
  SendValue = 3;
  xSemaphoreGive(Sem);
}

void MotorTask(void* arg) {
  int receiveValue = 0;
  while(1) {
    if(xSemaphoreTake(Sem, portMAX_DELAY)) {
      receiveValue = SendValue;
      if(receiveValue == 1) {
        digitalWrite(MT_P, LOW);
        digitalWrite(MT_N, HIGH);
      }

      //Stop
      else if(receiveValue == 2) {
        digitalWrite(MT_P, LOW);
        digitalWrite(MT_N, LOW);
      }
```

<ass3-4_1.c 코드 주요 부분>

과제 2)

과제 1과 똑같은 기능을 실행하지만 Producer, Consumer Problem을 생각하여 실제 Semaphore가 동작함으로서 Critical Section에서의 변수를 안전하게 제어할 수 있는지 확인할 수 있는 실습 이였습니다. KeypadTask로 인터럽트에서 전달받은 값은 Semaphore를 통해 전달되며 KeypadTask는 Producer 함수로 Bounded Buffer에 값을 입력합니다. 입력 후 MotorTask에 사용 가능하다는 Signal을 보냅니다. Signla이 보내지면 MotorTask는 Consumer 함수로 Bounded Buffer에 있는 값을 쓰고 나서 쓰인 값으로 Motor를 제어합니다 Motor가 제어되면서 MotorTask는 KeypadTask에게 이제 임계 영역에 접근이 가능하다는 Signal을 보냅니다. 즉, Producer와 Consumer는 동시에 동작할 수 없지만 서로 신호를 주고 받음으로써 임계영역에 안전하게 접근하고 해당 변수를 제어할 수 있는 것입니다. 먼저 BoundedBuffer라는 Struct를 만들었습니다. 이 구조체는 Circular Queue처럼 동작하고 크기가 5로 정해져있지만 CircularQueue에서 변수가 생성되고 사용됨으로서 메모리가 절약됩니다. in은 head, out은 tail입니다. 그리고 Mutex로 생성되는 BinarySemaphore 외에 추가로 BoundedBuffer만큼의 크기를 가지고 있는 Counting 가능한 Semaphore를 Empty, Full의 이름으로 각각 만들어 주었습니다. Empty는 처음에 가득 차 있는 상태이고 Full은 비어진 상태입니다. 이유는 Empty는 버퍼에서 비어있는 공간의 개수를 의미하고 Full은 버퍼에서 공간을 차지하는 개수를 의미하기 때문입니다.

Producer인 KeypadTask의 알고리즘을 보면 Semaphore가 동작할 때 데이터를 생성하기 전에 비어있는 공간이 있는지 확인합니다. 가득 찬 상태가 아니고 Mutex가 0보다 크면 Empty Semaphore의 value값이 -1이 되므로 block이 되고, 있다면 임계구역 내부로 진입하여 데이터를 생성합니다. Buffer의 in으로 data값이 복사가 되어 생성이 완료되면 Full세마포의 value값을 1 증가시킵니다. 따라서 Full Semaphore가 이후에 Take를 할 수 있게 되는 것입니다. 이 때 Debounce 문제를 해결하기 위해 Sendvalue값과 keypad 값을 전역변수로 별도로 선언하여 keypad != Sendvalue의 조건을 if문에 추가함으로써 같은 값이 buffer에 담기지 않도록 하였습니다. 반대도 똑같습니다. 데이터를 쓰기 전에 Full Semaphore를 통해 공간을 확인하고 data를 사용할 수 있다면 임계구역 내부로 진입하여 데이터를 사용하게 되는 것입니다. 이후 Empty에 Signal을 주어 KeypadTask가 이후 임계 영역에 접근할 수 있게 합니다.

Setup에서 xSemaphoreCreateCounting 함수로 Empty, Full semaphore를 생성하였으며 Mutex는 Binary로 생성하였습니다. 이후 스케쥴러를 통해 Task의 우선순위를 정할 때 KeypadTask인 Producer가 먼저 값을 넣어야 MotorTask인 Consumer가 값을 사용할 수 있으므로 Producer가 1, Consumer가 2의 순위로 스케쥴링되게 하였습니다.

컴파일 후 실행결과 1번을 누르면 왼쪽으로 모터가 돌아가게 되고 2번을 누르면 정지, 3번을 누르면 오른쪽으로 모터가 돌아가는 것을 확인할 수 있었습니다. Producer, Consumer Isuue를 통해 임계영역에 동시에 접속하는 상황을 방지하는 것을 확인할 수 있는 실습 이였습니다.

```
const int MT_P = 10;
const int MT_N = 9;
const int LeftKey = 2;
const int StopKey = 3;
const int RightKey = 21;

// 크기 5의 CirCular Queue Structure를 만들어 줌
struct BoundedBuffer {
  int arr[N];
  int in = N - 1;
  int out = N - 1;
};

// Binary SemaPhore인 Mutex와 Empty, Full SemaPhore 정의
SemaphoreHandle_t Mutex;
SemaphoreHandle_t Empty;
SemaphoreHandle_t Full;
int SendValue = 0, keypad = 0;

// Buff라는 이름의 Circular Queue 정의
BoundedBuffer Buff;
```

<Bounded Buffer를 Circular Queue 자료구조를 이용해 구현>

```
void KeypadTask(void* arg) {
  while(1) {
    // 인터럽트를 통해 Mutex가 1이 되면 0으로 값을 내리면서 Take, 그리고 Empty의 Semaphore 또한 값을 내리면서 Take
    // 또한 같은 값을 받은 Debouncing을 방지하기 위해 keypad와 SendValue 값을 따로 선언, 두 값이 다를 시 이후 함수 작동
    if(xSemaphoreTake(Mutex, portMAX_DELAY) && keypad != SendValue && xSemaphoreTake(Empty, portMAX_DELAY)) {
      keypad = SendValue;
      Buff.arr[Buff.in] = keypad;
      Buff.in = (Buff.in + 1) % N; // Circular Queue에 값 대입해서 저장
      xSemaphoreGive(Mutex);
      xSemaphoreGive(Full); // Consumer인 MotorTask가 접근할 수 있게 Mutex 와 Full의 Semaphore를 1 더해준다.
    }
  }
}

void MotorTask(void* arg) {
  int data;
  while(1) {

    // Producer를 통해 Full Semaphore에서 Take가 가능하면 Take하면서 Mutex의 Semaphore 또한 0으로 바뀌면서 Take
    if(xSemaphoreTake(Mutex, portMAX_DELAY) && xSemaphoreTake(Full, portMAX_DELAY)) {
      data = Buff.arr[Buff.out];
      Buff.out = (Buff.out + 1) % N; // Circular queue의 out으로 부터 값을 불러온다.

      //Left
      if(data == 1) {
        digitalWrite(MT_P, LOW);
        digitalWrite(MT_N, HIGH);
      }

      //Stop
      else if(data == 2) {
        digitalWrite(MT_P, LOW);
        digitalWrite(MT_N, LOW);
      }

      //right
      else if(data == 3) {
        digitalWrite(MT_P, HIGH);
        digitalWrite(MT_N, LOW);
      }

      xSemaphoreGive(Mutex);
      xSemaphoreGive(Empty); // Producer인 KeypadTask가 접근할 수 있게 Mutex 와 Empty의 Semaphore를 1 더해준다.
    }
 }
}
```

< Producer, Consumer인 Task들의 동작원리>

```
void setup() {
  pinMode(MT_P, OUTPUT);
  pinMode(MT_N, OUTPUT);
  pinMode(LeftKey, INPUT);
  pinMode(StopKey, INPUT);
  pinMode(RightKey, INPUT);

  attachInterrupt(0, LeftKeyControl, RISING);
  attachInterrupt(1, StopKeyControl, RISING);
  attachInterrupt(2, RightKeyControl, RISING);
  vSemaphoreCreateBinary(Mutex);
  Empty = xSemaphoreCreateCounting(5, 5);
  Full = xSemaphoreCreateCounting(5, 0);


  xTaskCreate(KeypadTask, NULL, 200, NULL, 1, NULL);
  xTaskCreate(MotorTask, NULL, 200, NULL, 2, NULL);
  vTaskStartScheduler();
}
```

<스케쥴링>

실행 결과는 동영상으로 Lab&Assignment_Record 폴더에 첨부되어 있습니다.