MCU를 위한 콘솔 장치 구동기 구현

김규형*, 이형봉*
*강릉원주대학교 컴퓨터공학과
e-mail:kping2@cs.gwnu.ac.kr, hblee@gwnu.ac.kr

Implementation of a Console Device Driver for MCUs

Kyu-Hyung Kim*, Hyung-Bong Lee*
*Dept of Computer Science & Engineering, Gangneung-Wonju National University

요 약

보통 MCU를 기반으로 하는 임베디드 소프트웨어는 펌웨어 형태로 구현되어 콘솔이 없고, UART는 데이터 송신용으로 사용된다. 그러나 MCU 소프트웨어에 콘솔 드라이버를 구현하면 유닉스·리눅스와 같은 대화적 표준 입·출력 개발환경과 타스크 개념을 실현할 수 있다. 이 논문에서는 운영체제 교과에서 익힌 내용을 바탕으로 Atmega2560 MCU에 인터럽트 기반 콘솔 장치 구동기를 구현하고 실험한다.

1. 서론

보통 MCU를 위한 내장 소프트웨어는 그림 1과 같이 무한 루프 형태의 펌웨어로 구성되어 타이머나 외부 인터럽트 등 이벤트에 따라 해당 작업을 처리한다.

```
while(1) {
      if      (event_flag1) do_event1();
      else if (event_flag2) do_event2();
      :
}
```

(그림 1) 전형적인 MCU 내장 소프트웨어 형태

그러나 그림 2와 같이 유닉스나 리눅스와 같이 시리얼 통신으로 연결된 단말기를 통하여 명령어 형태로 실시간 지시가 가능하기 위해서는 콘솔 장치 구동기가 필요하다.

```
char      buf[MAX_BUF];
while(1) {
      if (scanf("%s", buf) <= 0)
          break;
      if      (!strcmp(buf, "job1")) do_job1();
      else if (!strcmp(buf, "job2")) do_job2();
      :
}
```

(그림 2) 콘솔형 MCU 내장 소프트웨어 형태의 예

이 논문에서는 그림 3의 GEMS-CRC 모트와 AVRStudio 4 개발도구 환경에서 Atmega2560 MCU[1]의 콘솔 장치 구동기를 구현하고 실험한다.

(그림 3) GEMS-CRC 모트와 AVRStudio 4

2. UART 콘솔 장치 구동기

콘솔은 기본적으로 키보드로부터 문자를 읽어들이고 문자를 출력하여 화면에 나타나게 하는 도구로서 MCU의 UART0 장치를 이용하여 구현한다.

■ 콘솔 장치 입·출력 버퍼

UART0의 입력과 출력 버퍼로 그림 4와 같이 환형큐 두 개를 두어 사용한다.

```
char  qi[QI_SIZE], fi, ri, qo[QO_SIZE], fo, ro;
int   qi_insert(char c), qo_insert(char c); //full if 0
int   qi_delete(), qo_delete(); //empty if 0
```

(그림 4) 콘솔용 UART0를 위한 환형큐

■ 콘솔 입력 장치 구동기

Scanf(), fgets() 등과 같은 표준 입력 라이브러를 위한 입력 장치 구동기와 RX 인터럽트 핸들러는 각각 그림 5와 같고, 개략적인 동작 방식은 아래와 같다.

- 표준 입력 라이브러리들은 입력 구동기로부터 문자 단위로 하나씩 읽어간다.
- 키보드로부터의 입력은 RX 인터럽트에 의해 인식되고 입력된 문자는 입력 버퍼에 저장된다. 다만, 버퍼가 꽉 차 있다면 버린다.
- 표준 라이브러리는 구동기로부터 문자 단위로 하나씩 읽어가되, 버퍼가 비어있으면 인터럽트에 의해 데이터가 버퍼에 삽입될 때까지 바쁘게 대기한다.
- 입력 구동기와 RX 인터럽트 핸들러 사이의 입력 버퍼에 대한 경쟁상황은 인터럽트 마스킹으로 해결한다(그림 5의 cli() 및 sei()).
- ISR에서 입력받은 문자는 즉시 반송하여 키보드 입력에 대한 에코 효과를 얻도록 한다. 문자를 반송할 때는 출력 장치가 유휴상태(idle)라면 즉시 장치에 출하면서 장치의

사용중(busy) 표시를 설정하고, 이전 데이터가 아직 출력중(busy)이라면 출력 버퍼에 삽입해 준다.

```
#include <stdio.h>
#include <avr/ interrupt.h>
#include "queue.h “

int c_read(FILE *fp)
{
   char  c;

   do {
      cli();
      c = qi_delete();
      sei();
   } while (c == 0);

   if (c == ETX) return(-1);
   else          return(c);
}
```

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include "queue.h “
ISR(USART0_RX_vect)
{
  char  c = UDR0;
  if (c != ETX) {
     if (c==‘₩r’) c = ‘₩n’;
     key_echo(c);
  }
  qi_insert(c);
}

void  key_echo(char c)
{
  extern int  uart_busy;
  if (c == ‘₩n’)
      key_echo(‘₩r’);

  if (!uart_busy) {
      UDR0 = c;
      uart_busy = 1;
  }
   else qo_insert(c);
}
```

(그림 5) UART0 입력 장치 구동기

■ 콘솔 출력 장치 구동기

Printf(), fputs() 등과 같은 표준 출력라이브러리를 위한 출력 장치 구동기와 송신 완료를 표시하는 TX 인터럽트 핸들러는 그림 6과 같고, 개략적인 동작 방식은 아래와 같다.

- 표준 출력 라이브러리들은 출력 구동기에게 문자 단위로 하나씩 출력해간다.

- 출력 장치 구동기는 TX 인터럽트 핸들러와의 출력 버퍼에 대한 경쟁상황을 해결하기 위해 인터럽트를 마스크한 상태에서 동작하며, 장치의 현재 상태가 유휴 상태이면 즉시 장치에 출력하고 상태를 사용중으로 설정한다. 만약 사용중 상태라면 출력 버퍼에 삽입하되, 만약 버퍼가 꽉찬 상태라면 바쁜 대기 형식으로 버퍼 삽입이 성공할 때까지 대기와 삽입 시도를 반복한다. 대기 중에는 TX 인터럽트 핸들러가 버퍼에 접근하여 문자를 꺼내갈 수 있도록 인터

```
#include <stdio.h>
#include <avr/io.h>
#include <avr.interrupt.h>
#include <util/delay.h>
#include "queue.h “
int  uart_busy = 0;
int  c_write(char c, FILE *fp)
{
   if (c == ‘₩n’)
      c_write(‘₩r’, fp);
   cli();
   if (!uart_busy) {
      UDR0 = c;
      uart_busy = 1;
   }
   else {
     while(qo_insert(c) == 0){
        sei();
       _delay_us(100);
        cli();
     }
   }
   sei();
   return(1);
 }
```

```
#include <avr/io.h>
#include <avr.interrupt.h>
#include "queue.h"

ISR(USART0_TX_vect)
{
   extern int uart_busy;
   char  c;

   c = qo_delete();
   if (c == 0)
         uart_busy = 0;
   else
         UDR0 = c;
}
```

(그림 6) UART0 출력 장치 구동기

럽트 마스킹을 해제한다.

- 출력 버퍼에 대기 중인 문자들은 하나의 문자가 출력 완료될 때마다 TX 인터럽트 핸들러에 의해서 차례로 출력된다.

3. UART 콘솔 장치 구동기와 표준 입출력 라이브러리의 연계

AVR Libc(AVR C Library)는 그림 7의 과정으로 표준 입·출력 라이브러리와 UART 장치 구동기를 연계하도록 한다.

```
#include <stdio.h>
int  c_read(FILE *stream), c_write(char c, FILE *fp);
FILE Mystdin  = FDEV_SETUP_STREAM(NULL,c_read,
                                  _FDEV_SETUP_READ);
FILE Mystdout = FDEV_SETUP_STREAM(c_write, NULL,
                                  _FDEV_SETUP_WRITE);
void  c_init()
{
    stdin = &Mystdin; stdout = &Mystdout;
}
```

(그림 7) 콘솔 장치 구동기와 표준 입·출력 라이브러리의 연계

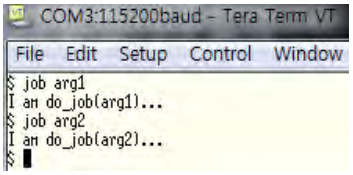
4. 실험 및 결론

MCU 주 루틴을 그림 8과 같이 작성하고 콘솔 단말기로 TeraTerm을 연결하여, 그림 9와 같이 명령어 형태의 대화적 콘솔 입·출력을 이룰 수 있다. 운영체제에서 익힌 인터럽트 기반 입·출력을 구현한 점에 큰 보람을 느낀다.

```
#include <stdio.h>
#include <string.h>
void do_job(void *ap)
{
    printf("I am do_job(%s)...₩n", (char *)ap);
}
int main()
{
    char  buf[80], *cp, *ap;
    while(1) {
        printf("$ ");
        if (!fgets(buf, sizeof(buf), stdin)) break;
        if (!(cp = strtok(buf, "₩n₩r ₩t"))) continue;
        ap = strtok(NULL, "₩n₩r ₩t");
        if (!strcmp(cp, "job")) do_job(ap);
        else  fprintf(stdout, "Unknown command...₩n");

    } return(0);
}
```

(그림 8) Atmega2560 주 루틴



(그림 9) 실험결과

참고문헌

[1] Microchip, "Atmega2560 Datasheet", http://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf

[2] Microchip, "AVR Libc User Manual", https://ti.tuwien.ac.at/ecs/teaching/courses/mclu/manuals/atmega1280/avr-libc-manual