韓國 컴퓨터情報學會 2006동계학술발표논문집&학회지
第14卷 第2號, 2006. 12.

이식 가능한 RTOS용 Character LCD 디바이스 드라이버

홍 정 환*

Portable RTOS Character LCD Device Driver

Hong Jung Hwan *

요 약

임베디드 소프트웨어(embedded software)에서 디바이스 드라이버(device driver)는 하드웨어와 운영체제 및 응용 프로그램 사이의 연결고리 역할을 하는 핵심 구성 요소로서, 응용 프로그램이 하드웨어에서 제공하는 기능을 사용할 수 있도록 제어 및 상호동작을 위한 일관된 인터페이스를 제공하는 소프트웨어이다. 이러한 디바이스 드라이버는 하드웨어와 소프트웨어의 양쪽 측면에 모두 관련이 있어서 개발이 어렵기 때문에 하드웨어 플랫폼이 바뀔 때마다 새로 작성하는 것은 굉장한 손실이다. 따라서 본 논문에서는 이기종의 하드웨어에서도 쉽게 이식 가능한 디바이스 드라이버 개발방법에 대해 살펴보고 효율적인 디바이스 드라이버 모델 개발방법에 대해 제안한다.

Abstract

A device driver is the core construction providing connectable rules between a hardware and an operating system in Embedded softwares. It provides consistent interface for the control and mutual interactions so that an application program can use hardware's functionalities. However, the device driver is suffered to develop as it is related to both hardware and software and it is also wasted to newly develop whenever hardware platform changed. Therefore, this paper researches the method to provide high portability in heterogenous hardware and finally suggests efficient device driver development.

▸ Keyword : 임베디드 소프트웨어(Embedded software), 디바이스 드라이버(device driver), 인터페이스(interface)

• 제1저자 : 홍정환
* 한양대학교 정보통신학과 석사과정

Ⅰ. 서 론

최근 들어 유비쿼터스 컴퓨팅 사회를 구현하기 위해 임베디드 시스템으로 대표되는 많은 디지털 기기들 간의 컨버전스 혹은 기존의 아날로그 기기와의 컨버전스가 빈번해지면서, 임베디드 소프트웨어의 효율적인 개발 방법, 특히 하드웨어와 맞닿아 있는 저수준(low-level)의 소프트웨어인 디바이스 드라이버를 개발하는 체계적이고 효율적인 방법에 대한 중요성이 증대되고 있다. 임베디드 시스템을 동작시키는 소프트웨어의 핵심이라 할 수 있는 RTOS 역시 많은 관심을 끌고 있다.

임베디드 소프트웨어(embedded software)에서 디바이스 드라이버(device driver)는 하드웨어와 운영체제 및 응용 프로그램 사이의 연결고리 역할을 하는 핵심 구성 요소로서, 응용 프로그램이 하드웨어에서 제공하는 기능을 사용할 수 있도록 제어 및 상호동작을 위한 일관된 인터페이스를 제공하는 소프트웨어이다. 이러한 디바이스 드라이버는 하드웨어와 소프트웨어의 양쪽 측면에 모두 관련이 있어서 개발이 어렵기 때문에 하드웨어 플랫폼이 바뀔 때마다 새로 작성하는 것은 굉장한 손실이다.

PC 안에 존재하는 하드웨어들은 각각의 고유한 값을 갖고 있으며 고유의 동작 방식을 갖게 된다. 만약 USER들이 직접 장비를 제어해야 한다면 각각의 장비에 대한 정보를 갖고 있어야 하며 장비 제어 방식 또한 장비마다 달라져야 한다. 이처럼 동일한 기능의 프로그램을 장비가 달라져서 또다시 만들어야 한다면 상당히 비효율적이게 될 것이다.

따라서 본 논문에서는 기존의 운영체제의 디바이스 드라이버 분석을 통해 이식 가능한 디바이스 드라이버 모델을 추출하고 이를 이용하여 Character LCD 디바이스 드라이버를 구현한다.

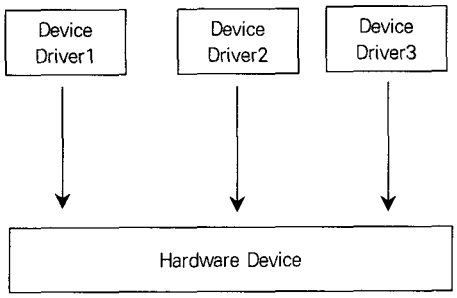

그림 1. 일반적인 디바이스 드라이버 모델
Fig 1.Typical Device Driver Model

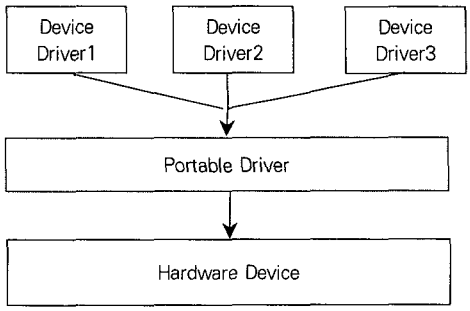

그림 2. 이식 가능한 디바이스 드라이버 모델
Fig 2. Portable Device Driver Model

Ⅱ. 관련 연구

Linux의 운영체제나 사용자 프로그램 개발은 많은 경우 공개 소스 코드 기반으로 이루어지고 있고, 개발자는 유사한 기능의 프로그램 소스 코드로부터 개발을 시작하는경향이 있다. 이와 같이 기존의 소스 코드를 수정하고 이를 기반으로 개발을 수행 하는 개발 방식은 해당 분야 기술의 발전과 개발 기간 단축 등의 긍정적인 영향도 가져올 수 있으나, 때로는 검증되지 않은 소스 코드의 반복적 유포와 재사용으로 인해, 오작동과 성능저하 등의 심각한 부작용을 가져오기도 한다. 디바이스 드라이버는 큰 의미로 운영체제의 일부에 속하기 때문에 소프트웨어의 성능과 신뢰성이 전체 시스템의 품질에 매우 중요하기 때문에, 개발 방식에 변화가 필요하다.

소프트웨어 개발 시간, 노력, 비용을 감소시키고 소프트웨어의 신뢰성을 향상시키면서 기술 지식이 풍부하지 않은 개발자에게 쉬운 개발 시작점을 제공하여 개발의 진입 장벽을 낮출 수 있는 코드 자동 생성기술과 같은 여러 가지 개발방법들이 연구 되고 있다.[1] 새로운 프로젝트를 생성하거나, 새로운 프로그램 모듈 또는, 객체지향 프로그래밍의 경우는 클래스를 추가하고자 할 때, 이름이나 속성 등을 프로젝트 마법사 형식 사용자 인터페이스를 통해 입력 받아서, 그들의 템플릿 소스코드를 생성하는 것이다. 이러한 관련 연구들은 제작하려는 소프트웨어의 복잡한 핵심 동작 논리 부분이 아니지만 공통적으로 삽입되어야 하는 소스코드 부분을 생성하여, 소프트웨어 개발자의 시간과 수고를 덜어줄 수 있는 방법들이다.

앞으로 실시간 시스템이 가전기기의 정보화 등으로 점차 산업용 가정용 기기 전반에 확산 될 것으로 전망되기 때문

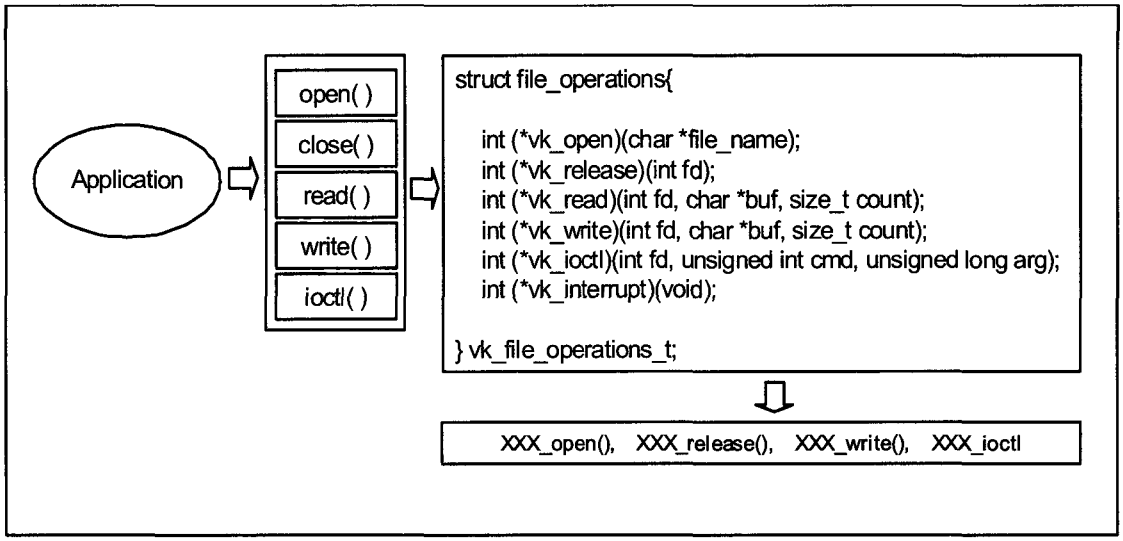

그림 3. 디바이스 드라이버 구조
Fig. 3 Device Driver Structure

에 시스템에 있어서 작업수행의 효율성에 있어서 이러한 디바이스 드라이버에 대한 개발기법, 운영체제를 이용하는 연구가 활발히 연구가 진행되고 있다.

II. 디바이스 드라이버 설계 및 구현

대부분의 디바이스 드라이버는 OS에 의존적이기 때문에, 다른 OS에서 해당 디바이스를 구동시키기 위해서는 디바이스 드라이버를 새로 설계해야 했다. 그러나 하드웨어엔지니어가 설계한 IP를 검증하기 위해 새로운 OS의 디바이스 드라이버 구조를 익히고, 작성하는 것은 쉬운 일이 아니다.

리눅스 기반의 RTOS는 디바이스 드라이버의 재사용성을 위해 리눅스 디바이스 드라이버와 유사한 자료구조와 등록 및 호출 구조를 가지고 있어 쉽게 상호간에 이식될 수 있다.[4][5] 디바이스 드라이버를 구현할 때 필요한 작업을 정리하면, 다음 4가지 정도의 작업이 필요하다.

첫째, 디바이스 드라이버의 고유 이름과 번호를 결정한다.

둘째, 디바이스 드라이버가 제공하는 파일 인터페이스를 위한 함수를 구현한다.

셋째, 구현한 각 함수명을 파일 연산(file_operations) 자료 구조에 등록한다.

넷째, 구현한 디바이스 드라이버를 사용 하고자 하는 응용프로그램에서는 디바이스 드라이버 초기화 함수를 호출해서 디바이스 드라이버를 커널이 관리하는 디바이스 드라이버 테이블에 등록한다.

아래 [그림3]을 보면 응용 프로그램은 커널이 제공하는 open(), close(), read(), write(), ioctl() APIs를 통해서 디바이스 드라이버에 접근할 수 있다. XXX_open()은 디바이스를 오픈 할 때 호출되는 함수로써 디바이스를 초기화 시키는 일을 수행하며, 반대로 XXX_release는 디바이스를 닫을 때 사용한다. XXX_read()는 입력된 데이터를 읽을 때 사용하는 함수이며, XXX_write()는 데이터를 출력할 때 사용하는 함수이다. XXX_ioctl()은 입출력을 제어하는데 사용되는 함수이다.

디바이스 드라이버 설계시 아래 그림과 같이 file_operations 자료 구조를 구성하는 멤버 변수에 대응하는 APIs를 작성해야 한다.

디바이스 드라이버를 사용하기 위해서는 다음 자료구조에 작성한 디바이스 드라이버를 등록시켜 주고 각 멤버 변수에 대응되는 함수를 작성한 후 file_operations 자료구조에 작성한 함수를 등록해야 한다.

디바이스 드라이버가 정상적으로 등록이 되면, 해당 디바이스 드라이버는 이를 관리하는 테이블에 등록되며 고유 번호로써 구분 된다. 이는 리눅스의 주번호, 부번호와 유사한 개념이라 할 수 있다.

register_dev(XXX_dev_number,
XXX_dev_name, &XXX_fops, irq_number)
는 초기화함수에 반드시 필요한 함수이다.

XXX_dev_number는 디바이스 드라이버 개발자가 할당해 주어야 하는 다른 디바이스 드라이버와 구별되는 고유번호이다.

XXX_dev_name도 역시 다른 디바이스 드라이버와 구별되는 고유 이름으로 할당해 주어야 한다. XXX_fops는 작성한 함수를 등록한 디바이스 드라이버 등록 구조체의 주소이다. irq_number는 해당디바이스의 인터럽트 번호이다. 주의할 점은 인터럽트를 사용하지 않는다면, NO_INT로 인자를 넘겨주어야 한다.)

vu_unregister_dev(XXX_dev_number, XXX_dev_name)는 디바이스 드라이버를 해제하는 함수로써 디바이스 드라이버 관리 테이블에서 해당 드라이버를 해제시켜 준다.[9][10]

```
typedef struct file_operations{

int (*vk_open)(char *file_name);
int (*vk_release)(int fd);
int (*vk_read)(int fd, char *buf, size_t
count);
int (*vk_write)(int fd, char *buf,
 size_t count);
int (*vk_ioctl)(int fd, unsigned int

cmd, unsigned long arg);
int (*vk_interrupt)(void);
} vk_file_operations_t;
```

〈디바이스 드라이버 등록 자료구조〉

```
typedef struct file_operations{

int (*vk_open)(char *file_name);
int (*vk_release)(int fd);
int (*vk_read)(int fd, char *buf, size_t
count);
int (*vk_write)(int fd, char *buf,
 size_t count);
int (*vk_ioctl)(int fd, unsigned int
cmd, unsigned long arg);
int (*vk_interrupt)(void);
} vk_file_operations_t;
```

〈디바이스 드라이버 등록 자료구조〉

```
struct file_operations XXX_fops =
{

/*      open            = */
      XXX_open,
/*      release         = */
      XXX_release,
/*      read            = */
      XXX_read,
/*      write           = */
      XXX_write,
/*      ioctl           = */
      XXX_ioctl,
/*      interrupt       = */
      XXX_interrupt }
```

〈디바이스 드라이버 등록 예제〉

```
 int XXX_driver_init(void)
     {
       int result;
      result=vu_register_dev(XXX_dev
_number, XXX_dev_name, &XXX_fops,
irq_number);
          if (result <0) return result;
          return 0;
     }
     void XXX_driver_cleanup(void)
     {
     vu_unregister_dev(XXX_dev_number,
XXX_dev_name);
     }
```

(디바이스 드라이버 초기화 및 해제 함수 예)

III. LCD 디바이스 드라이버 구현

본 논문에서는 문자 LCD가 실제 어떻게 하드웨어적으로 인터페이스 되어 있는지에 따라 그 내용이 달라지는 하드웨어 의존적인 부분은 배제하고 이 부분만을 수정하여 쉽게 이 기종으로 이식 가능한 필수 함수를 구현한다.

보드의 LCD모듈에 대한 번지, 장치번호 및 장치명, LCD모듈의 Instruction address 및 Data address 물리 번지는 각 하드웨어에 맞게 설정해 주어야 한다.

LCD에 문자를 출력하기위한 과정을 살펴보면 다음 〔그림4〕와 같다.

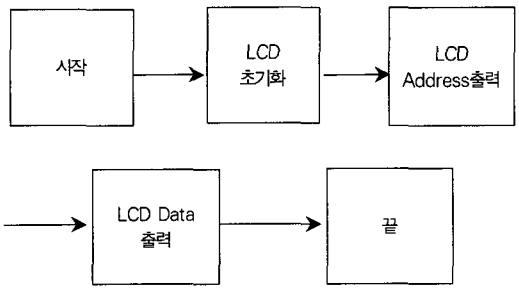

그림 4. LCD 출력 과정
Fig 4. LCD output process

LCD모드(cursor, bits, dots등)를 설정하고 DD RAM Address를 참조하여 출력하고자 하는 문자의 위치에 해당하는 번지 값을 LCD IR Write 번지에 출력하고 출력하고자 하는 문자에 해당하는 ASCII code를 LCD DR Write 번지에 출력하게 된다.

본 논문에서는 현재 LCD 디스플레이 모듈로 HD44780을 LCD컨트롤로 사용하는 모듈이 가장 많이 사용 되고 있으며 모듈별로 조금씩 다르지만 거의 비슷하기 때문에 이 모듈을 기반으로 디바이스 드라이버를 개발한다.〔2〕〔3〕〔6〕

이 드라이버에서 제공하는 기능을 살펴보면 80문자까지 표시 가능한 LCD모듈제어, ASCII문자, 문자열 표시, 5×7도트 매트릭스 기호를 8개까지 정의, 바그래프 표시 등이 있다.

문자 LCD모듈은 LCD와 구동회로로 구성된다. 문자 표시장치는 16~40 개의 문자로 구성된 라인을 포함하며, 보통 1~4개의 라인을 가진다. 각문자 블록은 5×8의 도트 매트릭스로 구성되고, ASCII문자와 제한된 종류의 기호를 표시할 수 있다.

문자모듈은 전화기, 에어컨 음향기기, 의료기기, 보안시스템 등 많은 임베디드 시스템에 사용되고 있다.

RTOS를 사용하는 멀티태스킹 환경에서는 태스크가 여러 개 동작한다. 즉, 한 태스크가 동작하고 있을 때 우선순위가 더 높은 태스크가 현재 실행중인 태스크를 선점해서 실행 될 수 있다. 이 때 기존의 태스크가 LCD같은 공유자원에 접근해서 사용 중이고 새로 실행된 태스크가 같은 공유자원을 사용하게 되면 LCD에 깨진 글자가 보이는 문제가 발생 할 수 있다. 멀티태스킹 환경에서는 이러한 공유자원 문제를 고려해야한다. 이러한 문제를 해결할 수 있는 가장 바람직한 방법은 공유자원 충돌을 해결할 수 있는 함수를 만들어서 태스크들이 사용토록 하는 것이다. Character LCD드라이버는 멀티 태스킹 환경에서 각 태스크가 사용할 수 있는 함수들을 제공하는데 이 함수들을 통해서 자원충돌 문제를 야기하지 않게 접근해야 한다. 즉, 이 함수들은 LCD나 시리얼 포트 같은 하드웨어 공유자원뿐 아니라 메모리상에 존재하면서 공유되는 변수, 행렬, 구조체 등도 안전하게 사용하게 해줘야한다.〔7〕〔8〕〔11〕

〔그림5〕에서 태스크 레벨에서 사용할 수 있는 함수와 하웨어 인터페이스 부분을 구분한 것을 볼 수 있다.

하드웨어 의존적인 부분은 Character LCD 드라이버가 하드웨어적으로 어떻게 인터페이스 되어 있는지에 따라 그 내용이 달라지는 부분이다. 디바이스 드라이버에서 사용되는 헤더파일을 선언해주고, 장치번호 및 장치명에 대한 정의, 응용프로그램에서 iocctl을 사용하여 명령어를 입력할 수 있도록 _IOW 매크로 정의, 디바이스 드라이버와 사용자 응용프로그램사이에 데이터를 전달할 때 그 매개체 역할의 구조체 변수 strcommand_variable 을 정의, 명령어 전달, lcd 초기화, display 컨트롤 명령어 set 정의, 커서shift 등의 함수 선언 등 디바이스 개발 시 꼭 필요한 필수 함수 부분만 분리하여 간단하게 이식하는 방법으로 개발자들은 쉽게 LCD 디바이스 드라이버를 구현할 수 있다.

디바이스 드라이버가 커널에 올라가는 순간 함수가 실행되고, 장치명과 메이저 번호를 가지고 드라이버를 커널에 등록하고 ioremap 과정을 통해서 io의 물리 주소를 새로운 커널 변수에 맵핑시키게 되는 것이다. 응용프로그램에서 오픈함수를 통해 불러들이면 매크로를 통해 드라이버가 사용중인 커널에 알려주고 초기화 함수로 LCD를 초기화해준다. 입력받은 문자들을 커널 메모리에 적재하기위해서 커널함수를 사용하고 문자를 문자열에서 하나씩 읽어 LCD에 부려주는 것이다. 디바이스 드라이버가 커널에 삽입될 때, 즉 insmod할 때는 커널에 등록하고 I/O공간을 커널로부터 할당받아야 한다.

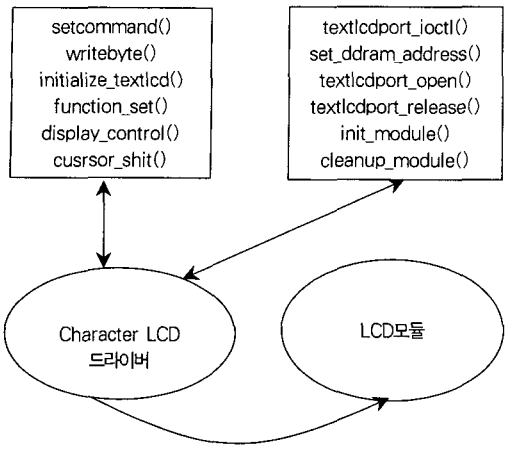

그림 5. 코드 분리
Fig 5. Code separation

반대로, 디바이스 드라이버가 커널로부터 제거될 때는 등록된 디바이스드라이버를 삭제하고 할당된 메모리 역역을 해제해 줘야한다.

IV. 결론 및 향후 과제

Character LCD 디바이스 드라이버에서는 setcommand()와 writebyte()를 사용하여 LCD의 write 타이밍을 맞추고 LCD모듈의 제어 명령에 따라 각각의 함수를 작성하였다. LCD 모듈을 초기화하여 곧바로 다른 함수에서 사용할 수 있게 만들고 텍스트를 받아서 LCD의 첫 번째 또는 두 번째 줄에 쓰도록 했다. 응용프로그램과 디바이스드라이버의 통신을 쉽게 하기 위해 strcommand_variable 구조체를 만들어 사용하고 이 구조체에 LCD모듈의 제어를 위한 변수들과 텍스트를 적을 수 있는 버퍼가 있어서 이곳에 사용할 명령코드나 텍스트를 넣어 사용자 응용프로그램이 디바이스 드라이버에게 전달하면 디바이스 드라이버에서도 위의 구조체를 활용하여 코드나 텍스트를 쉽게 분리 가능하다. 모든 테스크들은 Character LCD드라이버에서 제공하는 함수를 호출하므로 각 태스크가 원하는 위치에 문자를 표시하게 된다.

본 논문에서 구현한 디바이스 드라이버 모델을 기반으로 임베디드 시스템에서 시스템 규모가 커짐에 따라 요구되는 멀티태스킹 기능 특히, 네트워크나 멀티미디어에서의 네트워킹, GUI, 오디오 비디오 등이 요구되는 실시간 요소에 대한 향후 RTOS용 소프트웨어 개발의 가장 큰 장벽이었던 디바이스 드라이버의 개발이 더욱 쉬워질 것으로 기대한다. 또한 응용개발의 유연성을 제공하기위한 램디스크를 이용한 파일시스템, GPIO(general purpose I/O)를 통한 디바이스 드라이버를 제작을 통해 시스템의 안정성을 확보하는 등의 연구가 필요할 것으로 보인다.

참고문헌

[1] ETRI, "디바이스 드라이버 개발도구 동향", 2006
[2] 송태훈, "Intel PXA255와 임베디드 리눅스 응용", 홍릉과학출판사 2004
[3] 우상철, "임베디드 리눅스 시스템 설계", Ohm사, 2004
[4] Corbet, Jonathan, "리눅스 디바이스 드라이버", 2005
[5] 김지민, "VPOS", 한양대 실시간 시스템 연구실, 2005
[6] 김용민, "(PXA255 기반의) 임베디드 리눅스 프로그래밍", 2005
[7] Jean J. Labrosse, "MicroC/OS-II 실시간 커널", 에이콘출판, 2005
[8] 임주환, "임베디드 시스템 프로그래밍" 사이텍미디어, 2004
[9] 유영창, "IT EXPERT 리눅스 디바이스 드라이버", 한빛미디어, 2004
[10] 알렉산드로 로비니, "리눅스 디바이스 드라이버(개정3판)", 한빛미디어, 2005
[11] 홍원기, "고급 임베디드 시스템" 인터비젼, 2006

저 자 소 개

홍 정 환

2004년 2월 : 경희대학교 전파통신공학 학사
2005년 ~ 현재 : 한양대학교 정보통신학 석사과정
관심분야 :
실시간 시스템, 정보보안