리눅스 디바이스 드라이버 오류 테스트 모듈 설계

장승주*, 김동건*, 임채덕**, 마유승**
*동의대학교 컴퓨터공학과, ETRI 임베디드 S/W 연구단
e-mail : sjjang@deu.ac.kr, roknavy447@naver.com, {cdlim, ysma}@etri.re.kr

Design of the Error Test Module for a Linux Device Driver

Dong-Gun Kim*, Seung-Ju Jang*, Chae Duk Lim**, Yu Sung Ma**
*Dept. of Computer Engineering, Dong-Eui University, **ETRI

요 약

임베디드 리눅스 디바이스 드라이버의 개발이 증가하면서 리눅스 디바이스 드라이버에 대한 오류 테스트 기능을 가진 모듈의 필요성이 증가되고 있다. 본 논문은 리눅스 디바이스 드라이버를 위한 오류 테스트 모듈의 기본 개념을 제시하며, 기본 개념을 바탕으로 오류 테스트 모듈을 설계한다. 오류 테스트 모듈의 설계를 위해 리눅스 디바이스 드라이버의 메모리 관련 오류를 분류하고, 오류가 발생할 가능성이 존재하는 부분에 대한 검증을 위한 코드를 추가하여 테스트 모듈을 작성한다. 또한 작성된 오류 테스트 모듈의 실험을 진행하였다. 실험을 통해 리눅스 디바이스 드라이버의 오류 테스트 모듈이 동작을 확인 할 수 있다.

1. 서론

하루가 다르게 새로운 하드웨어가 나오고 있는 요즘, 이런 하드웨어를 리눅스 플랫폼에서 사용하기 위한 디바이스 드라이버 개발이 증가하고 있다.

리눅스를 위한 커널 디바이스 드라이버를 제작하는 제작사나 자신의 필요 때문에 직접 디바이스 드라이버를 만드는 사용자들이 그 예라고 할 수 있다.

이미 나와 있는 리눅스 디바이스 드라이버 뿐만 아니라 앞으로 나올 디바이스 드라이버의 오류를 테스트 할 수 있는 모듈의 필요성 또한 증가되고 있다.

본 논문 내용은 리눅스 디바이스 드라이버의 오류 테스트를 위한 모듈 설계를 목적으로 한다. 리눅스 디바이스 드라이버를 위한 오류 테스트 모듈의 기본 개념을 제시하며, 기본개념을 바탕으로 오류 테스트 모듈을 제작한다. 오류 테스트 모듈의 제작을 위해 리눅스 디바이스 드라이버의 메모리 관련 오류를 찾아 보았으며, 오류가 발생할 가능성이 존재하는 부분에 대한 검증을 위한 추가 코드를 추가하고 테스트 모듈을 작성하였다.

본 논문의 구성은 2 장에서 관련 연구를 살펴 보고, 3 장에서 오류 테스트 모듈의 기본 개념 및 설계를 살펴보고, 4 장 결론 순으로 되어있다.

본 논문은 한국전자통신연구원 2005 년도 연구비 지원에 의하여 연구되었습니다.

2. 관련 연구

2.1 디바이스 드라이버(Device Driver)

장치 제어기 또는 구동 드라이버로서 정의 된다. 하드웨어와 운영체제 응용프로그램의 연결 고리가 되는 프로그램으로 하드웨어 구성 요소가 운영체제하에서 작동하기 위해 필요한 모듈이다.

리눅스 디바이스 드라이버는 모듈 형태로 구성되기 때문에 사용자의 편의에 의해 커널에 적재와 해제가 용이하다.

본 논문에서는 리눅스 디바이스 드라이버를 위한 오류 테스트 모듈을 작성하기 위해 USB storage 디바이스 드라이버를 사용한다.

2.1.1 USB storage 디바이스 드라이버

USB Mass Storage를 위한 전송 프로토콜에는 데이터 전송에 사용되는 transaction format 에 따라 구분할 수 있는데, Control Bulk Interrupt Transfer를 사용하는 CBI Transport와 Bulk Transfer만을 사용하는 Bulk-Only Transport가 있다.

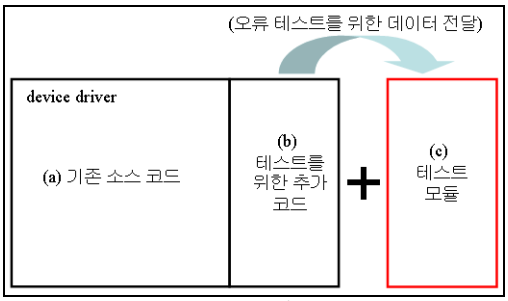
3. 오류 테스트 모듈 설계

3.1 오류 테스트 모듈 기본 개념

리눅스 디바이스 드라이버 오류 테스트 모듈을 기본 개념에서 두 가지 요소를 가진다. 이것은 검증을 위한 추가코드와 테스트 모듈이다.

검증을 위한 추가 코드는 리눅스 디바이스 드라이버 오류 테스트를 위해 기존 소스 코드(device driver)에 추가 코드를 삽입한다. 추가 코드의 기능은 디바이스 드라이버 오류 테스트를 위한 오류 데이터를 테스트 모듈로 전달하게 된다.

테스트 모듈은 검증을 위한 추가 코드에서 전달 받은 오류 데이터를 사용하여 디바이스 드라이버 오류 테스트를 수행한다.



[그림 1] 오류 테스트 모듈

[그림 1]은 리눅스 디바이스 드라이버 오류 테스트 모듈 기본 개념을 그림으로 표현한 것이다. [그림 1]에서 기존 소스 코드(device driver)가 (b)테스트을 위한 추가 코드와 결합하여 (c)테스트 모듈로 오류 테스트를 위한 데이터 전달 상태를 보여 주고 있다.

3.2 기존 USB storage 관련 파일

리눅스 디바이스 드라이버 오류 테스트 모듈의 기본 개념을 바탕으로 오류 테스트 모듈 제작하고자 한다. 제작에 앞서 기존 USB storage 관련 파일들이 어디에 위치하고 어떠한 파일들이 있는지 알아 볼 필요가 있다.

현재 리눅스 Kernel-2.6.x 버전의 USB storage 관련 디렉토리는 kernel-2.6.x/drivers/usb/storage 에 위치한다. 그리고, USB storage 관련 파일들은 scsiglue.c, protocol.c, usb.c, transport.c, initializers.c 들이 존재한다.

3.3 USB storage 메모리 관련 부분

리눅스 디바이스 드라이버에서 메모리 할당 시 오류가 발생할 수 있는 경우를 찾아 보았다. 첫 번째 GFP_KERNEL, GFP_AOTMIC 을 인자로 사용한 메모리 관련 함수, 두 번째 “Out of memory” 관련 부분, 세 번째 메모리 관련 함수에서 –E[type]을 리턴하는 부분을 찾을 수 있다.

3.3.1 GFP_KERNEL, GFP_ATOMIC 을 인자로 사용한 메모리 관련 함수

메모리 관련 함수에서 인자로 GFP_KERNEL 또는 GFP_ATOMIC 옵션을 사용할 경우 deadlock 발생을 가능성을 가진다.

```
us->cr = usb_buffer_alloc(us->pusb_dev, sizeof(*us->cr),
                          GFP_KERNEL, &us->cr_dma);
```

[그림 2] GFP_KERNEL 을 인자로 사용한 함수

[그림 2]는 GFP_KERNEL 을 인자로 사용한 메모리 관련 함수이다. [그림 2]의 usb_buffer_alloc 함수는 usb.c 파일의 associate_dev 함수 내의 디바이스 관련 DMA-mapped 버퍼 할당 기능을 수행하는 함수이다. [그림 2]에서 usb_buffer_alloc 함수는 3 번째 인자로 GFP_KERNEL 옵션을 사용한다.

3.3.2 “Out of memory” 검사 부분

“Out of memory” 검사 부분은 시스템의 용량 한계로 인하여 오류가 발생할 수 있는 가능성을 가진다.

```
us = (struct us_data *)kmalloc(sizeof(*us), GFP_KERNEL);
if (!us) {
   printk(KERN_WARNING USB_STORAGE “Out of
          memory\n”);
   return –ENOMEM;
}
```

[그림 3] “Out of memory” 검사 부분

[그림 3]은 us_data 구조체 할당 시 “Out of memory” 검사 부분을 나타내고 있다. [그림 3]에서는 us_data 구조체 할당 시 시스템의 용량 한계로 인하여 오류가 발생 할 수 있는 부분이다.

3.3.3 메모리 관련 함수 내에서 –E[type]을 리턴하는 부분

메모리 관련 함수 내에서 –E[type]을 리턴하는 부분을 찾을 수 있다.

```
switch (result) {
      case -ETIMEDOUT:
         US_DEBUGP("-- timeout or NAK\n");
         return USB_STOR_XFER_ERROR;
 }
```

[그림 4] –E[type]을 리턴하는 부분

[그림 4]는 transport.c 파일의 interpret_urb_result 함수 내에서 –E[type]을 리턴하는 부분을 나타내고 있다.

3.4 USB storage 검증을 위한 코드 추가

리눅스 디바이스 드라이버에서 메모리 할당 시 오류가 발생 할 수 있는 경우를 바탕으로 검증을 위한 추가 코드를 작성한다.

3.4.1 GFP_KERNEL, GFP_ATOMIC 을 인자로 사용한 메모리 관련 함수

```
us->cr = usb_buffer_alloc(us->pusb_dev, sizeof(*us->cr),
                          GFP_KERNEL, &us->cr_dma);
/* TEST_DRIVER ***************/
struct usb_ctrlrequest *back_us_cr;
back_us_cr = us->cr; us->cr = NULL;
test_ctrlrequest_alloc(us);
us->cr = back_us_cr;
/*****************************/
if (!us->cr) {
US_DEBUGP("usb_ctrlrequest allocation failed\n");
        return -ENOMEM;
}
```

[그림 5] GFP_KERNEL 옵션 추가 코드

[그림 5]는 GFP_KERNEL 옵션을 사용한 메모리 관련 함수에 대한 검증을 위한 추가 코드를 보여준다. [그림 5]에서 사용한 검증을 위한 추가 코드 작성법은 첫 번째 us->cr 데이터를 back_us_cr 로 백업, 두 번째 us->cr 데이터에 오류 데이터(NULL)을 할당, 세 번째 테스트 함수인 test_ctrlrequest_alloc 의 인자로 전달, 네 번째 테스트 모듈에서 us->cr 값을 사용하여 오류 테스트 수행, 다섯 번째 us->cr 데이터를 정상 데이터 back_us_cr 로 복구 한다.

3.4.2 "Out of memory" 검사 부분

```
us = (struct us_data *) kmalloc(sizeof(*us),
                                GFP_KERNEL);
/* TEST_DRIVER *************/
struct us_data *back_us;
back_us = us; us = NULL;
test_out_of_memory(us);
us = back_us;
/*****************************/
if (!us) {
   printk(KERN_WARNING USB_STORAGE "Out of
          memory\n");
   return -ENOMEM;
}
```

[그림 6] "Out of memory" 추가 코드

[그림 6] 에서는 "Out of memory"에 대한 검증을 위한 추가 코드를 보여준다. [그림 6] 에서 사용한 검증을 위한 추가 코드 작성법은 첫 번째 us 데이터를 back_us 로 백업, 두 번째 us 데이터에 오류 데이터(NULL)을 할당, 세 번째 테스트 함수인 test_out_of_memory 의 인자로 전달, 네 번째 테스트 모듈에서 us 값을 사용하여 오류 테스트 수행, 다섯 번째 us 데이터를 정상 데이터 back_us 로 복구 한다.

3.3.3 메모리 관련 함수 내에서 –E[type]을 리턴하는 부분

```
case -ETIMEDOUT:
/* TEST DRIVER ********************/
test_result_of_URB_transfer(result);
/**********************************/
US_DEBUGP("-- timeout or NAK\n");
```

[그림 7] –E(type) 리턴 부분에서의 추가 코드

[그림 7] 에서는 –E[type]을 리턴하는 부분에서 검증을 위한 추가 코드를 보여준다. [그림 7]에서 사용한 검증을 위한 추가 코드 작성법은 첫 번째 Error Type(result)을 테스트 함수의 인자로 할당, 두 번째 오류 테스트 모듈에서 해당 에러 타입에 대한 오류 테스트 수행을 진행한다.

3.5 USB storage 오류 테스트 모듈

오류 테스트 모듈에서는 검증을 위한 추가 코드에서 인자로 넘어온 데이터를 사용하여 오류 테스트를 수행한다.

3.5.1 GFP_KERNEL, GFP_ATOMIC 을 인자로 사용한 메모리 관련 함수

```
void test_ctrlrequest_alloc(struct us_data *us){
   if (!us->cr) {
      US_DEBUGP("##### usb_ctrlrequest allocation
                    failed  #####\n");
   } else {
      US_DEBUGP("#####      test_ctrlrequest_alloc()
OK!
            usb_ctrlrequest      allocation      success
#####\n");
   }
}
```

[그림 8] GFP_KERNEL 옵션 오류 테스트 모듈

[그림 8]은 GFP_KERNEL 옵션에 대한 오류 테스트 모듈을 보여준다. [그림 8]에서 오류 테스트 모듈은 검증을 위한 추가 코드에서 test_ctrlrequest_alloc 함수가 전달한 struct us_data *us 를 전달 받아 usb ctrlrequest 할당의 성공 여부를 검사하여 결과를 메시지로 출력한다.

3.5.2 "Out of memory" 검사 부분

```
void test_out_of_memory(struct us_data *us){
   if (!us) {
      printk(KERN_WARNING USB_STORAGE "#####
             Out of memory #####\n");
   } else {
```

```
        printk(KERN_WARNING USB_STORAGE "#####
                test_out_of_memory() OK! enough memory
                #####\n");
    }
}
```

[그림 9] "Out of memory" 검사 부분 오류 테스트 모듈

[그림 9] 는 "Out of memory" 검사 부분에 대한 오류 테스트 모듈을 보여준다. [그림 9]에서 오류 테스트 모듈은 검증을 위한 추가 코드에서 test_out_of_memory 함수가 전달한 struct us_data *us 를 전달 받아 "Out of memory" 부분을 검사하여 메시지로 출력한다.

3.5.3 메모리 관련 함수 내에서 –E[type]을 리턴하는 부분

```
void test_result_of_URB_transfer(int result) {
  switch (result) {
    case -EPIPE:
      US_DEBUGP("##### stall on control pipe #####\n");
          break;   }
}
```

[그림 10] –E(type) 리턴 오류 테스트 모듈

[그림 10] –E[type] 리턴에 대한 오류 테스트 모듈을 보여준다. [그림 10]에서 오류 테스트 모듈은 검증을 위한 추가 코드에서 test_result_of_URB_transfer 함수가 전달한 Error Type 을 전달 받아 검사하여 해당 Error Tyep 에 대한 메시지를 출력한다.

3.6 컴파일

3.6.1 컴파일 전 작업

3.6.1.1 헤더 파일 추가

오류 테스트 모듈 test_usb_mod.c 파일에서 구현한 테스트 함수의 사용을 위해서 test_usb_mod.h 헤더 파일을 transport.c 와 usb.c 파일에 헤더 파일로 추가한다.

3.6.1.2 Makefile 수정

USB storage 에서 오류 테스트 모듈을 추가하기 위해서 Makefile 을 수정하였다.

```
usb-storage-objs :=    scsiglue.o protocol.o transport.o usb.o \
                       initializers.o test_usb_mod.o $(usb-storage-obj-y)
```

[그림 11] USB storage Makefile 수정

[그림 11]은 USB storage 의 Makefile 수정 모습을 보여준다. Makefile 에서 오류 테스트 모듈인 test_usb_mod.o 를 추가해 준다.

3.6.2 컴파일

```
CC [M]  drivers/usb/storage/scsiglue.o
CC [M]  drivers/usb/storage/protocol.o
CC [M]  drivers/usb/storage/transport.o
CC [M]  drivers/usb/storage/usb.o
CC [M]  drivers/usb/storage/initializers.o
CC [M]  drivers/usb/storage/test_usb_mod.o
CC [M]  drivers/usb/storage/debug.o
CC [M]  drivers/usb/storage/shuttle_usbat.o
CC [M]  drivers/usb/storage/sddr09.o
CC [M]  drivers/usb/storage/sddr55.o
CC [M]  drivers/usb/storage/freecom.o
CC [M]  drivers/usb/storage/dpcm.o
CC [M]  drivers/usb/storage/isd200.o
CC [M]  drivers/usb/storage/datafab.o
```

[그림 12] USB storage 오류 테스트 모듈 컴파일

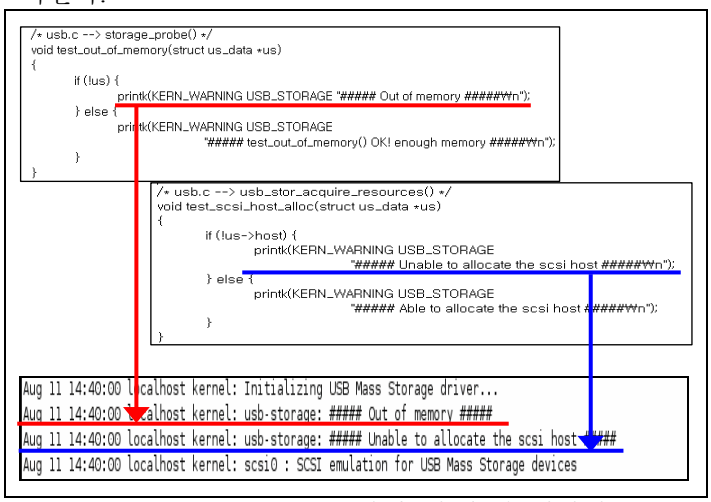
[그림 12]는 USB storage 오류 테스트 모듈이 컴파일 되는 모습을 보여준다.

3.7 실험

USB Mass storage 디바이스 드라이버에서 오류 테스트 모듈을 삽입하여 컴파일 하였다. USB storage 디바이스 드라이버 오류 테스트 모듈이 정상적으로 기능을 수행하는지 검사하기 위하여 테스트 하였다.

3.7.1 USB storage 삽입 실험

USB storage 를 삽입 시 test_usb_mod.c 모듈은 오류 테스트의 결과를 로그 메시지(/var/log/messages)로 출력한다.



[그림 13] USB storage 삽입 시 출력 메시지

[그림 13]은 USB storage 삽입 시 /var/log/messages 로그 파일에 출력되는 오류 테스트 모듈의 로그 메시지를 보여주고 있다.

4. 결론

본 논문은 리눅스 디바이스 드라이버를 위한 오류 테스트 모듈을 설계하였다. 그리고, 실험을 통해서 오류 테스트 모듈의 기능이 정상적으로 동작하는 것을 확인하였다.

본 논문에서 제안하는 리눅스 디바이스 드라이버를 위한 오류 테스트 모듈을 활용하여 디바이스 드라이버에 대한 오류 테스트가 가능하다.

참고문헌

[1] 리눅스 커널 디바이스 드라이버 만들기, http://www.kesl.org/psd/data/linuxdevicedriver.doc

[2] How to NOT write kernel driver, http://people.redhat.com/arjanv/

[3] A. Rubini, J. Corbet, Linux Device Driver, 2nd Edition, O'Reilly, 2001.

[4] 커널 디바이스 드라이버 모듈, http://www.falinux.com/win/study/06/devicedriver03.html