제25회 한국정보처리학회 춘계학술발표대회 논문집 제13권 제1호 (2006. 5)

리눅스 디바이스 드라이버에서 freed memory 기능 검증 모듈 설계

장승주*, 임채덕**, 마유승**
*동의대학교 컴퓨터공학과, 한국전자통신연구원 임베디드 SW 연구단
e-mail : sjjang@deu.ac.kr

A Design of the module freed memory verification at device driver of linux

Seung-Ju Jang*, Rim Chae-Duk**, Ma Yu-Sung**
*Dept. of Computer Engineering, Dong-eui University, ETRI**

요 약

임베디드 리눅스 디바이스 드라이버의 개발이 증가하면서 이에 대한 오류 테스트 기능을 가진 모듈의 필요성이 증가되고 있다. 본 논문은 리눅스 디바이스 드라이버를 위한 freed 메모리 오류 테스트 모듈의 기본 개념을 제시하며, 기본 개념을 바탕으로 오류 테스트 모듈을 설계한다. freed 메모리 오류 테스트 모듈 설계를 위해 리눅스 USB 디바이스 드라이버에 설계하고, 오류가 발생할 가능성이 존재하는 부분에 대한 검증을 위한 코드를 추가하여 테스트 모듈을 작성한다. 오류 테스트 모듈 설계를 위해서 usb storage 디바이스 드라이버를 대상으로 하였다. 또한 작성된 오류 테스트 모듈의 실험을 진행하였다. 실험을 통해 리눅스 디바이스 드라이버의 오류 테스트 모듈의 동작을 확인할 수 있다.

1. 서론

임베디드 시스템을 임베디드 운영체제의 사용은 급증하고 있는 추세이다. 그러나 임베디드 운영체제의 동작을 정확히 검증하고 시스템 안정화를 도모할 수 있는 방안에 대한 연구는 시스템의 신뢰성을 높이는데 필수적인 요소이다. 따라서 본 연구에서는 이러한 임베디드 시스템의 안정화를 위하여 임베디드 운영체제의 디바이스 드라이버 모듈의 검증과 요구 사항 분석, 관련 기법 연구를 통해서 개발 환경에서 필요로 하는 검증 및 테스트 기능을 개발한다. 이러한 개발을 통해서 보다 안정된 임베디드 운영체제 환경의 제공으로 신뢰성 높은 시스템을 개발하고자 한다.

임베디드 리눅스 가용성 및 성능 테스트 기술은 차세대 임베디드 시스템 관련 핵심 기술로써 중요한 의미를 가진다. 모든 컴퓨터 관련 기기들은 불안정한 특성을 가지고 있다. 이런 불안정한 특성을 가진 시스템을 안정화시키기 위해서 지금까지 많은 비용을 들여서 연구 개발을 진행해 왔다. 앞으로 컴퓨터의 기능이 임베디드 시스템으로 이전되는 추세를 가지고 있다. 임베디드 시스템의 동작을 안정화시키기 위한 핵심기술로써 중요성을 가진다. 따라서 임베디드 시스템을 제조하는 업체들은 커널 안정화에 많은 관심을 가지고 있다.

임베디드 리눅스 디바이스 드라이버의 개발이 증가하면서 이에 대한 오류 테스트 기능을 가진 모듈의 필요성이 증가되고 있다. 본 논문은 리눅스 디바이스 드라이버를 위한 freed 메모리 오류 테스트 모듈의 기본 개념을 제시하며, 기본 개념을 바탕으로 오류 테스트 모듈을 설계한다. freed 메모리 오류 테스트 모듈 설계를 위해 리눅스 USB 디바이스 드라이버에 설계하고, 오류가 발생할 가능성이 존재하는 부분에 대한 검증을 위한 코드를 추가하여 테스트 모듈을 작성한다. 오류 테스트 모듈 설계를 위해서 usb storage 디바이스 드라이버를 대상으로 하였다. 또한 작성된 오류 테스트 모듈의 실험을 진행하였다. 실험을 통해 리눅스 디바이스 드라이버의 오류 테스트 모듈의 동작을 확인할 수 있다.

2. Freed Memory 오류의 정의

Freed Memory 오류란 임의의 프로세스에게 할당된 메모리가 해제된 이후에 메모리 영역을 사용하기 위해 접근할 경우에 발생하는 에러를 말한다. 리눅스

커널이나 사용자 프로그램에서 이러한 오류가 발생하게 되면 예상치 못한 결과를 초래할 수 있다. [그림 1]은 freed memory 가 발생할 수 있는 경우의 프로그램 예제를 보여준다.

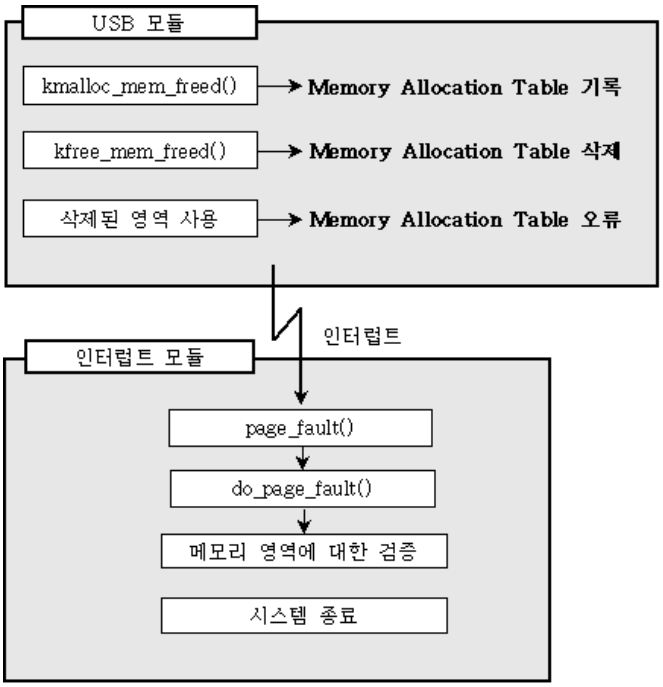
```
01: Function(arguments) {
02:         ……
03:         mem = kmalloc(Allocation_Page_Size,Allocation_option);
04:         ……
05:         kfree(mem);
06:         ……
07:         // any process access about " mem" ;  // Error 발생
08:         ……
09: }
```

(그림 1) Memory Freed 오류가 발생하는 프로그램 예제

(그림 1)은 Freed Memory 오류가 발생할 수 있는 경우에 대한 프로그램 코드이다. Line 03 은 kmalloc()을 사용하여 커널 메모리 영역을 mem 이라는 변수에 할당한다. Line 05 은 mem 변수에 할당한 커널 메모리 영역을 해제한다. Line 07 는 Line 05 에서 이미 할당을 해제한 커널 메모리 영역을 사용하게 된다. 이미 해제된 메모리 영역을 사용하게 되면 이 메모리 영역에 대해서 오류가 발생한다.

3. freed memory 오류 검증 모듈 동작 구조

본 논문에서 제안하는 freed memory 오류 검증 모듈에 대한 동작 구조는 다음 (그림 2)와 같다.



(그림 2) freed 메모리 오류 검증 모듈 동작 구조

USB 모듈의 kmalloc_mem_freed() 함수에서 memory allocation table 에 메모리 관련 정보를 기록한다. Kfree_mem_freed() 함수에서 memory allocation table 에 메모리 관련 정보를 삭제한다. USB 모듈에서 삭제된 메모리 공간에 대한 접근을 시도할 경우에 메모리 오류 관련 인터럽트가 발생하게 된다. 여기까지가 USB 모듈에서 발생하는 동작 과정을 나타낸다. 메모리 관련 인터럽트가 발생되면 인터럽트 모듈의 page_fault()함수를 수행하게 된다. Page_fault() 함수에서 do_page_fault() 함수를 수행하게 된다. do_page_fault() 함수 내에서 메모리 해제된 영역에 대한 검증을 수행한다. 검증 수행을 마치게 되면 시스템은 kill 하게 된다.

4. linux Kernel Memory Freed 테스트 모듈 구현

4.1 Memory Allocation Table의 구현

Freed Memory Error 방지를 위해서 다음 (표 1)과 같이 Memory Allocation Table을 구축한다

(표 1) Memory Allocation Table

메모리 free/allocate	메모리 할당 주소	PID (Process ID)	Last
0	00FFA1FD	38	0
1	00FFA1A3	40	0
. . .	. . .	. . .	. . .
1	00FE7432	432	1

(표 1)은 커널 메모리 영역의 할당 여부를 기록하는 Memory Allocation Table 이다. Memory Allocation Table 은 "메모리 free/allocate, 메모리 할당 주소, PID" 필드로 구성된다.

1) 메모리 free/allocate 필드는 메모리 영역의 할당 여부를 나타낸다. 메모리 영역이 정상적으로 할당된 경우 메모리 free/allocate 필드를 '1' 로 설정하고, 정상적으로 해제된 경우 메모리 free/allocate 필드를 '0' 으로 설정함으로써 메모리 영역의 할당 여부를 알 수 있도록 구현한다.
2) 메모리 할당 주소 필드는 메모리가 할당된 영역의 주소를 기록한다.
3) PID 필드는 메모리 할당 및 해제를 필요로 하는 프로세스의 ID 를 기록한다.
4) Last 필드는 마지막으로 free 된 메모리를 찾기 위해서 설정한 필드로써 마지막에 free 된 메모리의 Last 필드를 '1' 로 설정함으로써 마지막으로 free 된 메모리 주소를 찾을 수 있게 한다.

위의 4 가지 필드를 가진 Memory Allocation Table 을 구현함으로써 접근하려는 메모리 영역에 대해서 해당 프로세스가 사용가능 한지를 확인할 수 있다.

주) 이 table 은 커널 소스의 /root/linux-2.6.9/drivers/usb/storage 에 mem_freed.h 에 작성되어 있다.

4.2 kmalloc_mem_freed(), kfree_mem_freed()의 구현

Memory Allocation Table에 메모리 영역 할당 여부를 기록하기 위해 kmalloc(), kfree()를 부분을 대신할 함수를 구현한다. 다음 (그림 2)는 kmalloc_mem_freed()를, (그림 3)은 kfree_mem_freed()를 구현한 것이다.

kmalloc_mem_freed()함수는 kmalloc()를 대신한 함수이고 kfree_mem_freed()는 kfree()함수를 대신한 함수이다.

```
int kmalloc_mem_freed(size_t size, unsigned int flag)
{
        struct us_data *us;
        us = (struct us_data *) kmalloc(size, flag);
        struct mat *kmat;
        kmat = pmem_alloc_table;

        kmat->bit = 1;
        kmat->addr = (int)us;
        kmat->pid = (pid_t)current->tgid;
        kmat->last = 0;
        kmat++;

        return ((int)us);
}
void kfree_mem_freed(const void *addr)
{
        int i;
        struct mat *kmat = pmem_alloc_table;
        for(i=0;i<TABLE_SIZE;i++){
                if((int)kmat++->addr == (int)addr){
                        kmat--;
                        break;
                }
        }
        kmat->bit = 0;
        for(i=0;i<TABLE_SIZE;i++){
                if((int)kmat++->addr != (int)addr){
                        kmat->last = 0;
                }
        }
        kmat->last = 1;
        kfree(addr);
}
```

(그림 3) kmem_freed()와 kfree_mem_freed()의 구현

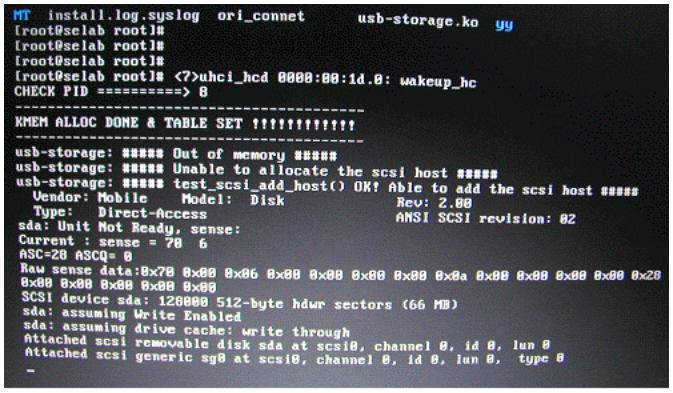
(그림 2)는 kmalloc_mem_freed(), kfree_mem_freed()를 이용하여 kmalloc(), kfree()를 실행한다. 기존의 kmalloc(), kfree() 처리 보다 추가된 점은 [표 1]에서 구현한 Memory Allocation Table 에 메모리 영역 할당 정보를 추가하는 부분이다.

1) kmalloc_mem_freed() : 기존의 kmalloc() 함수를 이용하여 커널 메모리 영역 할당이 성공하면 Memory Allocation Table 에 할당 정보를 기록하고 테이블을 초기화 한다. 메모리 할당이 실패할 경우는 메모리 할당이 되지않기 때문에 메모리 할당 테이블에 기록하지 않는다.

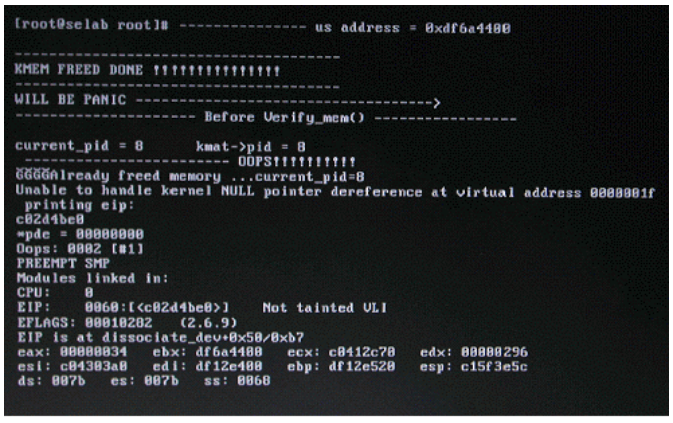
2) kfree_mem_freed() : 기존의 kfree()를 이용하여 커널 메모리 영역을 해제하고, Memory Allocation Table의 할당 정보를 삭제한다

5. 실험

본 논문은 리눅스 USB 디바이스 드라이버에서 freed memory 기능을 실제 구현하여 검증하는 과정을 실험하였다. 앞에서 제시한 freed memory 기능을 USB 디바이스 드라이버에 구현하여 실험하였다. 실험후의 결과 화면은 다음 (그림 4), (그림 5)와 같다.



(그림 4) USB 메모리를 추가한 경우의 실험 결과



(그림 5) USB 메모리를 삭제했을 때의 실험 결과

(그림 4), (그림 5)는 USB 메모리를 삽입, 삭제했을 때의 실험 결과이다. 이 실험은 본 논문에서 제시한 freed memory 검증 기능을 USB 디바이스 드라이버에 구현한 후에 실험을 한 결과 화면이다. 이 실험을

통해서 본 논문에서 제시하는 freed 메모리 기능 검증 모듈이 정상적으로 동작됨을 확인할 수 있었다.

6. 결론

본 논문은 리눅스 디바이스 드라이버에서 freed memory 가 발생할 경우에 이것을 검증할 수 있는 기능을 설계한다. 본 논문은 리눅스 디바이스 드라이버를 위한 freed 메모리 오류 테스트 모듈의 기본 개념을 제시하며, 기본 개념을 바탕으로 오류 테스트 모듈을 설계한다. freed 메모리 오류 테스트 모듈 설계를 위해 리눅스 USB 디바이스 드라이버에 설계하고, 오류가 발생할 가능성이 존재하는 부분에 대한 검증을 위한 코드를 추가하여 테스트 모듈을 작성한다. 이 기능은 디바이스 드라이버에서 이미 free 된 메모리를 접근하는 경우에 치명적인 결과를 초래할 수 있다. 이러한 결과가 발생할 수 있는 부분을 미리 찾아 냄으로써 보다 안정된 디바이스 드라이버가 될 수 있도록 보장할 수 있다. 본 논문은 리눅스 USB 디바이스 드라이버에서 freed memory 기능을 실제 구현하여 검증하는 과정을 실험하였다. 실험 결과 정상적인 동작이 됨을 확인할 수 있었다.

참고문헌

[1] Programming Guide for Linux USB Device Drivers, http://www.lrr.in.tum.de/Par/arch-/usb/usbdoc/

[2] Michael Beck,Mirko Dziadzka, Ulrich Kunitz and Harald Bohme, Linux Kernel Internals, Addison - Wesley , 1997.

[3] A.Rubini&j.Corbet, Linux Device Dribver(2nd), O'Relly, 2001

[4] Katayama, T.; Saisho, K.; Fukuda, A., " Prototype of the device driver generation system for UNIX-like operating systems", Proceedings. International Symposium on 1-2 Nov 2000.

[5] Katayama, T.; Saisho, K.; Fukuda, Proposal of a support system for device driver generation, A.,Software Engineering Conference, 1999. (APSEC '99) Proceedings. Sixth Asia Pacific,7-10 Dec. 1999

[6] Albinet, A.; Arlat, J.; Fabre, J.-C., Characterization of the impact of faulty drivers on the robustness of the Linux kernel, Dependable Systems and Networks, 2004 International Conference on28 June-1 July 2004.

[7] A. Rubini, J. Corbet, Linux Device Driver, 3rd Edition, O'Reilly, 2004.

[8] The Linux Kernel, David A Rusling, http://linuxkernel.net/kernel/tlk/origtext/tlk-0.8-3.pdf

[9] 유영창, " IT EXPERT 리눅스 디바이스 드라이버", 한빛미디어, 2004

[10] Peter Jay Salzman, Ori Pomerantz, Linux Kernel Module Programming Guide, http://www.faqs.org/docs/kernel/

[11] Kernel development, http://lwn.net/Articles/22699/

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

[13] Write a Linux Hardware Device Driver, http://www.networkcomputing.com/unixworld-/tutorial/010/010.txt.html

[14] Linux Loadable Kernel Module HOWTO, http://www.ibiblio.org/pub/Linux/docs/HO-WTO/other-formats/html_single/Module-HOWTO.html

[15] Ashfaq A. Khan, Delmar Thomson Learning, " Practical Linux Programming: Device Drivers, Embedded systems, and the Internet", 2002

[16] Porting device drivers to the 2.6 kernel, http://lwn.net/Articles/driver-porting/