리눅스 디바이스 드라이버 내의 메모리 오류 테스트 모듈 설계

장 승 주†

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 적용하고, 오류가 발생할 가능성이 존재하는 부분에 대한 검증 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.

키워드 : 리눅스 운영체제, 디바이스 드라이버, 메모리 관리, 메모리 관리 모듈 설계

Design of the Memory Error Test Module at a Device Driver of the Linux

Seung Ju Jang†

ABSTRACT

The necessity of error test module is increasing as development of embedded Linux device driver. This paper proposes the basic concept of freed memory error test module in the Linux device driver and designs error test module. The USB device driver is designed for freed memory error test module. I insert the test code to verify the USB device driver. I test the suggested error test module for the USB storage device driver. I experiment error test in this module.

Key Words : Linux O.S, Device driver, Memory management, Design memory management

1. 서 론

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

임베디드 리눅스 가용성 및 성능 테스트 기술은 차세대 임베디드 시스템 관련 핵심 기술로서 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. 앞으로의 컴퓨터 기능은 임베디드 시스템으로 이전되는 추세이다. 임베디드 시스템의 동작을 안정화시키기 위한 핵심기술은 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 적용하고, 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.

※ 이 논문은 2006학년도 동의대학교 교내연구비에 의하여 연구되었음. 2006AA161
†정 회 원 : 동의대학교 컴퓨터공학과 교수
논문접수 : 2006년 11월 3일, 심사완료 : 2007년 4월 19일

2. 관련 연구

디바이스 드라이버와 관련한 기존의 연구는 dtrace, 아이락, visionWARE 등이 있다 [1, 2, 13]. Dtrace 툴에서 디바이스 드라이버 디버깅은 cmn_err() 함수를 이용하여 /var/adm/messages 에 분석 로그를 기록하는 방법을 사용한다 [3, 4, 5, 13, 16]. 이러한 방법을 통해 추측하고 재 컴파일하고 시스템 재부팅을 통해 해결되지 않은 에러를 찾게 된다. 어셈블리에 능통한 개발자는 adb 디버거를 이용하여 mdb를 위한 C 모듈을 작성하여 소프트웨어 에러를 검사하는 방법을 사용한다. 이러한 디버깅 방법은 많은 시간이 소요된다. 아이락은 C 프로그램 실행 중에 발생할 수 있는 버퍼오버런(buffer overrun) 에러를 프로그램 실행 전에 자동으로 빠짐없이 찾아 주는 프로그램 분석기이다 [6, 7, 8, 14, 15]. 이 분석기는 요약 해석의 틀 속에서 설계됐으며, 리눅스 커널, GNU 프로그램, 상용 임베디드 소프트웨어에 적용되어 실제 버그를 찾는 이용이 되고 있다. 또한 아이락은 정적 프로그램 분석으로부터 기인하는 허위 경보에 유연하게 대처할 수 있는 기능을 제공한다.

vison WARE는 기능성과 순응성을 검증하기 위해 단일 쓰레드를 사용하여 진단(diagnostic) 및 생성된 Posix 디바이스 드라이버 모델 개념을 채택한다. 이 방법은 다중 쓰레드된 디버그(multi-threaded debug) 환경의 복잡함을 없애준다 [9, 10, 11, 12]. 이러한 기존의 연구들은 운영체제의 디바이스 드라이버 안정성을 높이기 위해서 이루어지고 있다. 또한 리눅스 운영체제에서 디바이스 드라이버 내의 메모리 관리 뿐만 아니라 커널 하드닝과 관련한 연구를 통하여 디바이스 드라이브의 안정성에 대한 연구를 진행하고 있다 [17].

또한 메모리 할당과 관련한 오류를 검증하는 dmalloc 라이브러리도 있다. Purify 패키지는 사용자 프로그램에 gcc 컴파일러에서 메모리 할당 오류를 검증할 수 있도록 지원하고 있다.

3. Freed Memory 오류의 정의

Freed Memory 오류는 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 일반적인 경우에 메모리 할당 후에 freed memory가 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: }
```

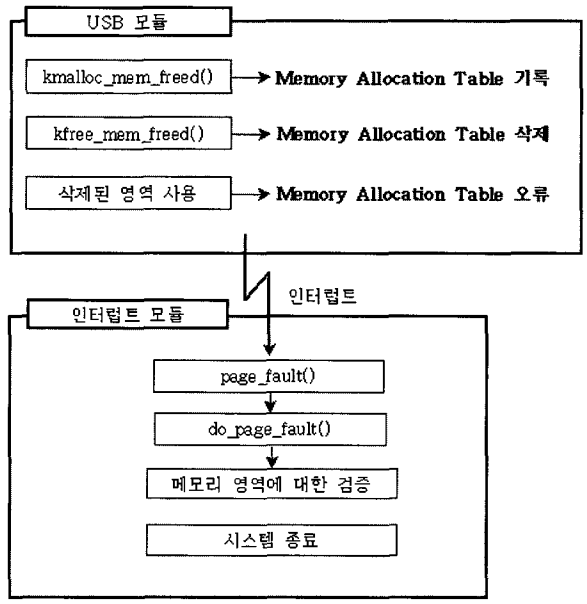
(그림 1) Memory Freed 오류가 발생하는 프로그램 예제

(그림 1)은 Freed Memory 오류가 발생할 수 있는 경우에 대한 프로그램 코드이다. Line 03은 kmalloc()을 사용하여 커널 메모리 영역을 mem이라는 변수에 할당한다. Line 05 은 mem 변수에 할당한 커널 메모리 영역을 해제한다. Line 07는 Line 05에서 이미 할당을 해제한 커널 메모리 영역을 사용하게 된다. 이미 해제된 메모리 영역을 사용하게 되면 이 메모리 영역에 대해서 오류가 발생한다. 즉, 이미 해제된 메모리 공간에 대한 접근을 시도할 경우에 메모리 관련 오류가 발생하게 된다. 이 기능이 리눅스 커널 내에서 발생하게 되면 시스템 panic으로 연결된다.

4. 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)의 경우는 리눅스 커널 내에서 설계된 메모리 free관련 오류를 검증할 수 있는모듈에 대한 동작 흐름이다. 메모리 free 관련 오류를 검증하기 위하여 USB storage를 이용한다. 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.



(그림 2) freed 메모리 오류 검증 모듈 동작 구조

5. linux Kernel Memory Freed 테스트 모듈 구현

5.1 Memory Allocation Table의 구현

리눅스 디바이스 드라이버에서 free된 메모리에 대한 오류를 방지하기 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(MAT) 이다. 이 테이블은 메모리 할당과 해제에 관한 정보를 기록한다. 여기에 기록된 정보를 이용하여 메모리 해제 여부를 확인하도록 한다. 메모리 MAT에는 "메모리 free/allocate, 메모리 할당 주소, PID" 필드로 구성된다. 각 필드의 기능은 다음과 같다.

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 ID를 기록한다.
4) Last 필드는 마지막으로 free된 메모리를 찾기 위해서 설정한 필드로써 마지막에 free된 메모리의 Last 필드를 '1'로 설정함으로써 마지막으로 free된 메모리 주소를 찾을 수 있게 한다.

위의 4가지 필드를 가진 Memory Allocation Table을 구현함으로써 접근하려는 메모리 영역에 대해서 해당 프로세스가 사용가능 여부를 확인할 수 있다. 그리고 MAT테이블은 커널 소스의 /root/linux-2.6.9/drivers/usb/storage 디렉토리의 mem_freed.h 파일에 작성되어 있다.

5.2 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 3)은 kmalloc_mem_freed(), kfree_mem_freed()를 구현하였다. (그림 3)에서 kmalloc_mem_freed()함수는 kmalloc()를 대신한 함수이고 kfree_mem_freed()는 kfree()를 대신한 함수이다.

(그림 3)은 kmalloc_mem_freed(), kfree_mem_freed()를 이용하여 kmalloc(), kfree()를 실행한다. 기존의 kmalloc(), kfree() 처리에 추가된 기능은 [표 1]에서 구현한 Memory Allocation Table에 메모리 영역 할당 정보이다.

1) kmalloc_mem_freed() : 기존의 kmalloc() 함수를 이용

```
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

하여 커널 메모리 영역 할당이 성공하면 Memory Allocation Table에 할당 정보를 기록한다. 메모리 할당이 실패할 경우는 메모리 할당이 되지않았기 때문에 MAT 메모리 할당 테이블에 기록하지 않는다.

2) kfree_mem_freed() : 기존의 kfree()를 이용하여 커널 메모리 영역을 해제하고, kmalloc_mem_freed()함수에서 MAT 테이블에 기록한 정보들을 삭제한다

6. 실험 및 비교 분석

본 논문의 실험에서는 기존의 dmalloc 툴을 이용한 실험을 수행하였다. Dmalloc 툴을 이용한 실험은 (그림 4), (그림 5)와 같다.

```
#include <stdio.h>
#include <stdlib.h>
#ifdef DMALLOC
#include <dmalloc.h>
#endif
int main(void)
{
        char *ptr;
        ptr = malloc(5);
        return 0;
}
```

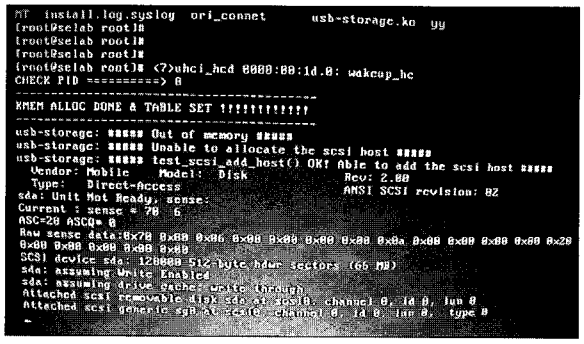
(그림 4) Dmalloc에서 메모리 오류를 일으키는 소스 코드

```
1125406700: 1: Dmalloc version '5.4.2' from 'http://dmalloc.com/'
1125406700: 1: flags = 0x4f4e503, logfile 'logfile'
1125406700: 1: interval = 100, addr = 0, seen # = 0, limit = 0
1125406700: 1: starting time = 1125406700
1125406700: 1: process pid = 21163
1125406700: 1: Dumping Chunk Statistics:
1125406700: 1: basic-block 4096 bytes, alignment 8 bytes
1125406700: 1: heap address range: 0x40017000 to
               0x40020000, 36864 bytes
1125406700: 1:      user blocks: 1 blocks, 4069 bytes (11%)
1125406700: 1:     admin blocks: 8 blocks, 32768 bytes (89%)
1125406700: 1:     total blocks: 9 blocks, 36864 bytes
1125406700: 1: heap checked 1
1125406700: 1: alloc calls: malloc 1, calloc 0, realloc 0, free 0
1125406700: 1: alloc calls: recalloc 0, memalign 0, valloc 0
1125406700: 1: alloc calls: new 0, delete 0
1125406700: 1:   current memory in use: 5 bytes (1 pnts)
1125406700: 1:  total memory allocated: 5 bytes (1 pnts)
1125406700: 1:  max in use at one time: 5 bytes (1 pnts)
1125406700: 1: max alloced with 1 call: 5 bytes
1125406700: 1: max unused memory space: 27 bytes (84%)
1125406700: 1: top 10 allocations:
1125406700: 1:  total-size  count in-use-size  count  source
1125406700: 1:          5      1           5      1  dmalloc.c:9
1125406700: 1:          5      1           5      1  Total of 1
1125406700: 1: Dumping Not-Freed Pointers Changed Since Start:
1125406700: 1:  not freed: '0x40017fe8|s1' (5 bytes) from 'dmalloc.c:9'
1125406700: 1:  total-size  count  source
1125406700: 1:          5      1  dmalloc.c:9
1125406700: 1:          5      1  Total of 1
1125406700: 1: ending time = 1125406700, elapsed since start = 0:00:00
```

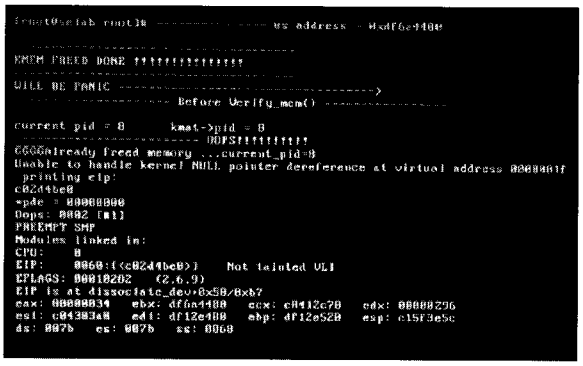
(그림 5) dmalloc 메모리 할당 오류 점검 실험 화면

또한 본 논문은 리눅스 USB 디바이스 드라이버에서 freed memory 기능을 실제 구현하여 검증하기 위한 실험을 수행하였다. 본 논문에서 제시한 freed memory 개념을 리눅스 USB 디바이스 드라이버에 구현하고 실험하였다. 본 논문에서 제시한 개념을 구현 후 실험한 결과 화면은 다음 (그림 6), (그림 7)과 같다.

(그림6), (그림 7)은 리눅스 커널 내의 USB 디바이스 드라이버 내에 본 논문에서 제안하는 freed memory 개념을 구현한 내용에 대해서 실험한 내용이다. 리눅스 시스템에 USB 메모리를 삽입, 삭제했을 때의 실험 결과 화면이다. 이 실험은 본 논문에서 제시한 freed memory 검증 기능을 USB 디바이스 드라이버에 구현한 후에 실험을 한 결과 화면이다. (그림 4)는 USB 메모리를 삽입했을 때 USB 디바이스 드라이버에서 출력해 주는 결과를 보여주는 화면이다. 이 출력 메시지에서 "KMEM ALLOC DONE & TABLE SET !!!!!!!!!!"이라는 메시지는 실제 USB 디바이스 드라이버에서 출력된 것이다. 이 메시지를 통해서 USB 메모리 추가시에 정상적으로 커널 내에서 메모리 할당이 이루어짐을 알 수 있다. (그림 6)에서 "KMEM FREED DONE !!!!!!!!!!!!!!!!!!"는 정상적으로 freed 메모리가 일어남을 확인할 수 있다. 그리고 그 다음 "WILL BE PANIC" 메시지는 freed 된 메모리에 대한 강제 사용을 시도하였다. Freed된 메모리에 대한 강제 쓰기를 시도하였기 때문에 패닉이 발생되는 것을 확인할 수 있었다. 이 실험을 통해서 본 논문에서 제시하는 freed 메모리 기능 검증 모듈이 정상적으로 동작됨을 확인할 수 있었다.



(그림 6) USB 메모리를 추가한 경우의 실험 결과



(그림 7) USB 메모리를 삭제했을 때의 실험 결과

다음 표는 기존의 메모리 할당 오류 점검 툴과 본 논문에서 제안하는 기법을 기능, 성능, 정확도 측면에서 비교하였다.

〈표 2〉 기존 메모리 할당 오류 점검 툴과 비교

툴 종류	기능	성능	정확도
Dmalloc	응용 프로그램 수준에서 메모리 할당 오류 검출	메모리 할당과 관련 모듈 기능으로 인한 성능 저하	사용자 프로그램의 메모리 할당과 관련하여 사용자 수준에서 검증이 이루어짐. 사용자 프로그램에 대해서 정확도를 가지고 있음.
Purify	응용 프로그램에서 gcc 컴파일러와 함께 메모리 할당 오류 검출	메모리 할당과 관련 모듈 기능으로 인한 성능 저하	사용자 프로그램의 메모리 할당과 관련하여 사용자 수준에서 검증이 이루어짐. 사용자 프로그램에 대해서 정확도를 가지고 있음.
본 연구 제안 커널 방식	커널 내에서 메모리 할당과 관련한 오류 검증	메모리 할당 오류 코드를 수행하는 부분에서 성능 저하 발생	커널 내에 메모리 오류 검증 프로그램 코드를 실행할 경우에 메모리 할당 오류 검증이 정확히 수행됨.

7. 결 론

본 논문은 리눅스 디바이스 드라이버에서 freed memory가 발생할 경우에 이 메모리가 정확히 free되었다는 것을 검증할 수 있는 기능을 설계한다. 본 논문은 리눅스 디바이스 드라이버를 위한 freed 메모리 오류 테스트 모듈의 기본 개념을 제시하며, 기본 개념을 바탕으로 리눅스 디바이스 드라이버에 오류 테스트 모듈을 설계한다. freed 메모리 오류 테스트 모듈 설계를 위해 리눅스 USB 디바이스 드라이버에 본 논문에서 제시하는 freed 메모리 개념을 설계하고, 오류가 발생할 가능성이 존재하는 부분에 대해서 freed memory 검증을 위한 코드를 추가하여 테스트 모듈을 작성한다. Free된 메모리에 대한 기능의 실험은 디바이스 드라이버에서 이미 free된 메모리로 접근하는 경우에 어떤 현상이 발생하는 지를 통해서 이루어진다. 이 실험에서 이미 free된 메모리에 대한 접근이 디바이스 드라이버에서 발생할 경우에 커널 패닉이 발생한다. 본 논문에서 제안하는 기능은 리눅스 디바이스 드라이버에 본 논문에서 제안하는 기능을 구현함으로써 free된 메모리에 대한 접근을 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.

참 고 문 헌

[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/

[17] 장승주, 리눅스 커널에서 하드닝 기능 구현, 한국정보처리학회 논문지, pp.227-234, 2004.11.30.

장 승 주

e-mail : sjjang@deu.ac.kr

1985년 부산대학교 계산통계학과(전산학) 학사

1991년 부산대학교 계산통계학과(전산학) 석사

1996년 부산대학교 컴퓨터공학과 박사

1987년~1996년 한국전자통신연구원 시스템 S/W연구실

1993년~1996년 부산대학교 시간강사

2000년~2002년 University of Missouri at Kansas City, visiting professor

1996년~현 재 동의대학교 컴퓨터공학과 교수

관심분야 : 운영체제, 분산시스템, 임베디드 시스템 운영체제, 시스템 보안