TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

2018-04-13 (37 회차)

강사: Innova Lee(이상훈)
gcccompil3r@gmail.com
학생: 정유경
ucong@naver.com

Chap 7. 리눅스 모듈 프로그래밍
Ch8. 디바이스 드라이버 - 8.2 문자 디바이스 드라이버 구조 (36 회차 참고)
Ch9. 네트워킹
부록 A 리눅스와 가상화 그리고 XEN

Chap 7. 리눅스 모듈 프로그래밍

7.1 마이크로 커널

- 모노리틱(monolithic) 커널
: 커널이 제공해야 할 모든 기능이 단일한 커널 공간에 구현된 구조
: 리눅스 커널은 모듈을 지원해서 마이크로 커널의 장점을 제공함
: 모듈을 이용하여 많은 기능을 필요로할 때 적재할 수 있도록 함(커널 공간에 적재)
: 모듈 사용시 커널에 새로운 기능을 추가하면 커널 소스를 컴파일할 필요 없음

- 마이크로 커널
: 커널 공간에 반드시 필요한 기능만 구현
: 문맥 교환, 주소 변환, 시스템 호출 처리, 디바이스 드라이버 중 하드웨어와 밀접 관련 기능만 커널에 구현
: 그외 다른 기능은 사용자 공간에 구현
: 커널의 크기를 작게 할 수 있음, 관리/유지/개선이 쉬움
: 다른 임베디드 시스템에 적용 가능, 분산환경(클라이언트-서버 모델)에 적용
: 최근에 개발되는 운영체제는 대부분 마이크로 커널 구조를 가짐

7.2 모듈프로그래밍 무작정 따라하기

```
/*Hello Module.c*/
#include <linux/kernel.h>
#include <linux/module.h>

int hello_module_init(void)
{
	printk(KERN_EMERG "Hello Module~! I'm Kernel\n");
	return 0;
}

void hello_module_cleanup(void)
{
	printk("<0>Bye Module~! \n");

}

module_init(hello_module_init); // insert module 시에 실행
module_exit(hello_module_cleanup); // remove module 시에 실행

MODULE_LICENSE("GPL");
```

```
/*Makefile*/

obj-m		:= hello_module.o

KERNEL_DIR	:=/lib/modules/$(shell uname -r)/build
PWD		:= $(shell pwd)
```

```
default :
	$(MAKE)	-C	$(KERNEL_DIR)	SUBDIRS=$(PWD)	modules
clean :
	$(MAKE) -C	$(KERNEL_DIR)	SUBDIRS=$(PWD)	clean
```

/*모듈 컴파일 및 수행 (모듈 프로그래밍 관련 명령어)*/
make
sudo insmod hello_module.ko // insmod - 인자로 주어진 파일을 커널에 넣기 위한 공간을 할당받고 모듈 삽입
dmesg (device message 확인)
lsmod | grep hello_module // lsmod - 현재 적재된 모듈의 정보 보여주기
rmmod hello_module // rmmod - 적재된 모듈을 해제

*. insmod, rmmod 시에 printk() 는 출력되지 않는다. → kernel 메시지 확인하는 dmesg 로 확인

7.3 시스템 호출 hooking → open 을 myopen 으로 바꾸자

http://old-releases.ubuntu.com/releases/16.04.3/ubuntu-16.04.1-desktop-amd64.iso
16.04.1 amd64 버전으로 다운로드 하길 바람!

Virtual Box 는 아래에서 다운 받도록 한다.
http://download.virtualbox.org/virtualbox/5.0.40/VirtualBox-5.0.40-115130-Win.exe

sudo apt-get install vim
sudo apt install git
git clone ~
make
sudo insmod syscall_hook.ko
lsmod | grep syscall_hook 확인해본다
dmesg
rmmod syscall_hook

```
/*syscall_hook.c*/
#include <linux/kernel.h>
#include <linux/module.h>
#include <asm/unistd.h>
#include <linux/syscalls.h>
#include <linux/hugetlb.h>

unsigned long **sys_call_table;

unsigned long **locate_sys_call_table(void)
{
	unsigned long tmp;
	unsigned long *p;
	unsigned long **sys_table;
/* 0xffffffff81000000 ~ 0xffffffffa2000000 까지 찾는다*/
	for(tmp = 0xffffffff81000000; tmp < 0xffffffffa2000000; tmp += sizeof(void *))
	{
		p = (unsigned long *)tmp;
```

```
                if(p[__NR_close] == (unsigned long)sys_close)
                {
                        sys_table = (unsigned long **)p;
                        return &sys_table[0]; // 시스템콜 테이블의 위치 리턴
                }
        }

        return NULL;
}

asmlinkage long (* orig_call)(const char __user *, int, umode_t);

asmlinkage long sys_our_open(const char __user *filename, int flags, umode_t mode)
{
        printk("<0>Open System Call\n");
        return (orig_call(filename, flags, mode));
/*sys_our_open 출력시, 리턴으로 origcall 이 오므로
하나만 실행되는 것이 아니라, 원래 open 도 하고 our_open 도 하게된다.*/
}

static int (*fixed_set_memory_rw)(unsigned long, int);

int syscall_hooking_init(void)
{
        unsigned long cr0;

        if((sys_call_table = locate_sys_call_table()) == NULL) // 시스템콜테이블의 위치 찾기
        {
                printk("<0>Can't find sys_call_table\n");
                return -1;
        }

        printk("<0>sys_call_table is at[%p]\n", sys_call_table);

        cr0 = read_cr0(); // cr0 레지스터 값을 읽는다
        write_cr0(cr0 & ~0x00010000);
/*커널에 위치한 sys_call_table 의 쓰기를 풀어준다
0x00010000 즉 16 번째 비트(page 쓰기를 방지하는 bit)를 0 으로 클리어한다.*/
        fixed_set_memory_rw = (void *)kallsyms_lookup_name("set_memory_rw");
        if(!fixed_set_memory_rw) //함수포인터 fixed_set_memory_rw 를 잘 가져왔니?
        {
                printk("<0>Unable to find set_memory_rw symbol\n");
                return 0;
        }

        fixed_set_memory_rw(PAGE_ALIGN((unsigned long)sys_call_table) - PAGE_SIZE, 3);
/*잘 가져왔다면,
sys_call_table 을 정렬(align) 해서 PAGE_SIZE(4096) 빼고 페이지 3 개를 준다?*/
        orig_call = (void *)sys_call_table[__NR_open]; // 원래 open 의 주소를 orig_call 에 저장
```

```
        sys_call_table[__NR_open] = (void *)sys_our_open;
/*our_open 을 sys_call_table 에 등록해놓는다 (hooking)*/
        write_cr0(cr0); // 다시 쓰기금지해놓는다. (cr0 의 값은 바뀌지 않았다)
        printk("<0>Hooking Success!\n");
        return 0;
}

void syscall_hooking_cleanup(void)
{
#if 1
        unsigned long cr0 = read_cr0();
        write_cr0(cr0 & ~0x00010000); // 쓰기금지 해제
        sys_call_table[__NR_open] = orig_call; //hooking 전으로 바꾸어 놓는다
        write_cr0(cr0);
        printk("<0>Module Cleanup\n");
#endif
}

module_init(syscall_hooking_init);
module_exit(syscall_hooking_cleanup);
MODULE_LICENSE("GPL"); // 사수에게 물어보자:)
```

Ch9. 네트워킹

네트워크

1) Who? → NIC(Network Interface Card)마다 지정되어 있는 IP 주소

(MAC 주소는 로컬망에서 사용)

2) (한 개의 NIC 를 통해 통신하려는)각각의 **사용자 태스크를 구분**? → 포트 번호(port)

통신하기 위해서는 서로 IP 주소와 포트 번호를 알고 있어야 하고, 프로토콜을 지켜야함

TCP/IP 프로토콜

1) BSD 소켓층

- 사용자 태스크는 소켓만을 인식하고 파일 시스템 인터페이스를 통해 일관된 구조로 통신
- 소켓 인터페이스 : socket(), bind() ,connect(), accept(), listen(), send(), recv(), read(), write()
- 프로토콜 패밀리(도메인) - INET, UNIX, IPX, APPLETALK

2) INET 층(BSD 소켓층에서 INET 을 선택할 경우)

- 소켓의 유형: Stream(TCP), Datagram(UDP)

3) TCP 층 (Transmission Control Protocol, INET 층에서 Stream 유형을 선택할 경우)

- 연결 지향 신뢰성 있는 프로토콜

4) IP 층

- IP 주소를 사용해 통신 (숫자열대신 IP 에 대응되는 이름을 가짐(DNS))
- 자신의 IP 주소와 목적지 IP 주소를 이용해 패킷을 만듬
- 에러처리를 위한 체크섬(checksum), 데이터가 너무 클 경우 단편화(fragmentation), 다른 호스트로 재전송(forwarding) 수행
- 전송할 때 큐 정책(queue discipline)에 따라서 패킷 흐름 제어(shaping)

5) 데이터 링크 층(ex. PPP, SLIP, 이더넷)
- 네트워크 디바이스 → net_device 자료구조에 자신의 정보 및 기능을 저장해 IP 층에 제공
- MAC 주소(이더넷 주소) : 모든 이더넷 디바이스는 각각을 구분하기 위한 고유주소를 가짐
- 특정 이더넷 주소는 멀티캐스트 목적으로 예약되어 있음
- IP 주소를 MAC 주소로 변환: ARP(Address Resolution Protocol) 사용
- RARP(Reverse Address Resolution Protocol): 이더넷 주소를 IP 주소로 변환

데이터 전달과 Encapsulation
1) Application data: Application header + user data
2) TCP message: TCP header + Application data
- TCP header: 소스 포트번호, 목적지 포트번호, SEQ, ACK, 윈도우 크기
- 포트 번호(TCP 층에서 응용을 구별): ftp 21, telnet 23, www 80
- 일련 번호(SEQ), 응답 번호(ACK), 윈도우 크기 : 신뢰성 있는 데이터 전달 지원
- 상태 정보(state): 에러 및 예외 처리
3) IP packet: IP header + TCP 메세지
- version
- TTL(Time To Live): 패킷이 목적지에 도달하기 위해 최대 몇번 재전달(forwarding) 될 수 있는지 나타냄, 0 이 되면 더 이상 forwarding 하지 않고 패킷 삭제
- Transport: IP 의 상위 층 (6 이면 TCP, 17 이면 UDP)
- Checksum: 전송 시 에러 발생 여부 파악
- 보내는 호스트 IP 주소, 목적지 호스트 IP 주소, 실제 데이터
4) 이더넷 프레임(frame): 이더넷 header + IP 패킷
- 보내는 호스트 이더넷 주소, 목적지 호스트 이더넷 주소, 프로토콜 필드, CRC(전송 시 발생한 에러 파악)

9.2 주요 커널 내부구조

sk_buff(소켓버퍼)자료구조
- 사용자 태스크에서 데이터가 전달되면 하위 계층으로 전달하여 가장 하위 계층인 드라이버 층까지 내려 보내야 하는데, 매번 실제 복사 작업을 수행하면 매우 비효율적 → 가상 복사 개념을 지원
- 이 자료구조에 237P 의 ethernet frame 을 통째로 넣는다

부록 A 리눅스와 가상화 그리고 XEN

“다중 운영체제를 지원하는 가상화 기술”

A.1 가상화 기법의 이해
가상화는 물리적인 자원과 사용자가 사용하는 자원을 분리하는 것
→ **가상화 층** 도입하여 물리자원의 복잡함을 숨기고 일관된 가상자원을 제공
가상화 기술의 핵심은 물리적인 자원을 가상자원으로 추상화 시키는 것
“가상화는 통합, 고립, 집합, 이동, 에뮬레이션등의 장점을 제공한다”
1) 서버의 이용률을 높이고 관리부하를 줄일 수 있다 → 다양한 응용 및 시스템소프트웨어를 통합운영, 관리
2) 각 사용자의 수행 환경을 다른 환경으로부터 고립시킬 수 있다→ 신회성, 보안성
3) 여러 물리자원을 단일한 가상자원으로 집합할 수 있다 → 물리적으로 분리된 공간을 논리적으로 연결하고 동일한 인터페이스를 통해 접근(컴퓨터 여러대를 한대처럼, 한대를 여러대처럼)
4) 시스템의 이동성을 증가시킨다. ↔ 포팅: hw 종속적(sw 를 hw 에 맞게 구현)
(물리적 자원의 종류에 무관하게 응용프로그램과 OS 가 이주)
5) 모의 실험이 가능

*. 가상화 방법은 가상화 층을 어디에 도입하느냐에 따라 분류할 수 있음 (347p)

→ 하드웨어 기반, 운영체제 기반, 미들웨어 기반, 응용프로그램 수준 가상화
*. 가상화는 목표로 하는 시스템에 따라 분류(348p)
→ 서버, 저장공간, 네트워크, 내장형시스템

A.2 가상화 기술
*. VMM(Virtual Machine Monitor)에서 가상화기술
- 하이퍼바이저(hypervisor)는 호스트 컴퓨터에서 다수의 운영 체제(operating system)를 동시에 실행하기 위한 논리적 플랫폼(platform)을 말한다. 가상화 머신 모니터(virtual machine monitor, 줄여서 VMM)라고도 부른다.

1) CPU 가상화 : 물리 CPU 를 여러 가상 CPU 에게 스케쥴링을 통해 공평, 효율적으로 할당해 주는 것

2), 메모리가상화: 물리메모리를 Guest OS 가 사용하기 쉬운 형태의 가상메모리로 추상화 시켜 지원하고, 각 가상머신의 요구에 맞게 메모리를 동적할당

3) 디스크나 네트워크와 같은 I/O 가상화 기술 : 각 가상머신간의 간섭을 없애고,, 각 가상머신이 예약한 입출력 품질을 보장할수 있도록 가상화

이와 같이 각 자원들(CPU,메모리, I/O)에 대한 추상화 기술은 자원 고립과 효율간에 균형을 맞추어 추상자원들을 구성하고 이 추상자원 위에 여러 운영체제가 동작하게 되는 것이다.

읽어보자: http://www.itworld.co.kr/insight/17199
들어보자: https://www.youtube.com/watch?v=y1ViwbFrNGw