TI DSP,MCU 및 Xilinux Zynq FPGA

프로그래밍 전문가 과정

이름	문지희
학생 이메일	mjh8127@naver.com
날짜	2018/4/13
수업일수	37 일차
담당강사	Innova Lee(이상훈)
강사 이메일	gcccompil3r@gmail.com

목차

Chapter 9 네트워킹

(1) 계층구조

: 통신 프로토콜은 약속이다.

: NIC(Network Interface Card)마다 지정되어 있는 IP 주소를 통해 누구와 통신을 할 지 지정한다. 한 컴퓨터는 복수개의 NIC 가 장착 가능하고 IP 주소를 복수 개 가질 수 있다.

: 여러 태스크가 한 개의 NIC 를 통해 통신할 수 있으니 각각의 사용자 태스크를 구분하기 위한 포트(port)번호가 필요하다.

프로토콜의 구성

: 표준 프로토콜인 OSI 프로토콜은 7 층을 가지지만 실제로는 4 계층만 구현 가능하다.

: 가장 상위층은 사용자에게 소켓 인터페이스를 제공하는 BSD 소켓 층이다. 사용자 태스크에게 소켓이라는 객체를 제공한다. 대표적인 소켓 인터페이스는 socket(), bind(), connect(), accept(), listen(), send(), recv()등이 있다.

: IP 계층 아래에는 PPP, SLIP 또는 이더넷과 같은 데이터 링크 층이 존재한다. 디바이스는 net_device 라는 자료구조에서 자신의 정보 및 기능을 저장.

: IP 계층 아래에는 PPP, SLIP 또는 이더넷과 같은 데이터 링크 층이 존재한다. 디바이스는 net_device 라는 자료구조에서 자신의 정보 및기능을 저장하여 IP 층에 제공하게 된다.

이더넷에는 많은 호스트가 실제 케이블 하나에서 동시에 접속할 수 있다. 모든 이더넷 디바이스는 각각을 구분하기 위한 고유한 주소인 이더넷 주소 또는 MAC 주소라고 한다.

: IP 주소를 이더넷주소(MAC 주소)로 변환하기 위해 ARP(Address Resolutiojnj Protocol)을 사용하고, 이더넷 주소를 IP 주소로 변환하기 위해 RARP(Reverse Address Resolution protocol)를 사용한다.

: ftp 는 21, telnet 은 23, www 는 80 의 포트를 사용한다.

데이터 encapsulation

Ethernet frame	Destination ethernet address	Source address	protocol	data	CRC

IP packet	Version	TTL	Transport	Checksum	Source IP address	Destination IP address	data

TCP message	Source port	Destination port	SEQ	ACK	data

Application data	Application header	User data

(2) 주요 커널 내부 구조

: 통신 프로토콜은 계층구조를 갖는다. 상위 층은 여러 하위 층과 연동할 수 있다. 통신 프로토콜 층을 내려오면서 다양한 곳으로 제어 흐름이 분기할 수 있다는 의미이다.

: 리눅스는 소켓을 생성하면 fd 가 생성되고 네트워크도 결국 파일로 전달하게 된다. 즉, 리눅스는 제어 흐름이 다양한 곳으로 분기할 수 있는 상황을 위해 층 사이에서 제어가 전달될 때 자료구조를 이용한 간접 호출 방법으로 통신 프로토콜을 구현한다. 파일연산(file_operations) 자료구조를 이용해 디바이스 드라이버 함수를 호출한 것과 유사하다.

: sk_buff 에 ethernet_frame 에 대한 정보들을 채워 넣는다. 위의 데이터 encapsulation.

부록 A 리눅스와 가상화 그리고 XEN

가상화 기법 : OS 도 결국 프로세스이고, 중첩페이징을 이용하여 윈도우 커널에서 페이지를 받아와 3 번(10/10/12bit)를 페이지하고 리눅스 내에서도 또 다시 3 번 페이지를 하여 총 9 번하게 되면 윈도우에서 리눅스를 띄울 수 있음. 하지만 속도가 느리다는 단점이 있다.

가상화 기술의 장점

1. 가상화는 서버의 이용률(utilization)을 높이고 관리 부하를 줄일 수 있다.
2. 각 사용자의 수행 환경을 다른 환경들로부터 고립시킬 수 있어 보안수준을 높일 수 있다.
3. 여러 물리자원들을 단일한 가상자원으로 집합할 수 있다.
4. 시스템의 이동성을 증가시킨다.
5. 새로운 시스템이나 아직 개발되지 않는 하드웨어를 모의실험(emulation)하는 기능을 제공한다.

Chapter 7 리눅스 모듈 프로그래밍

(3) 시스템 호출 hooking

```
 #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;

    for(tmp = 0xffffffff81000000; tmp < 0xffffffffa2000000; tmp += sizeof(void *))
    {//stack 의 시작점부터 8 바이트씩 증가하여 위치를 찾음
        p = (unsigned long *)tmp;
        if(p[__NR_close] == (unsigned long)sys_close)
            //__NR_close 와 sys_close 가 같으면
        {
            sys_table = (unsigned long **)p;
            return &sys_table[0];//sys_tavle[0]의 주소를 반환
        }
    }

    return NULL;//못찾으면 NULL 반환
```

```
}

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));
}

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)
                //system call table 의 위치를 찾는 함수
    {
        printk("<0>Can't find sys_call_table₩n");//못찾았을 경우 출력
        return -1;
    }

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

    cr0 = read_cr0();
    write_cr0(cr0 & ~0x00010000);//cr0 의 16 번째 비트를 0 으로 초기화시킴, 페이지쓰기를 방지하는 비트

    fixed_set_memory_rw = (void *)kallsyms_lookup_name("set_memory_rw");
    //set_memory_rw 라는 함수를 가져옴
    if(!fixed_set_memory_rw)//함수를 못가져왔을 경우 오류메세지 출력
    {
```

```
        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);
    //sys_call_table 을 PAGE_ALIGN 시켜서 PAGE_SIZE(4096)만큼 빼고 페이지 3 을 할당해준다.
    //PAGE_ALIGN : 페이지단위로 정렬하는것
    //페이지 쓰기를 가능하게 해줌
    orig_call = (void *)sys_call_table[__NR_open];//함수포인터를 가져와 대체
    sys_call_table[__NR_open] = (void *)sys_our_open;
    //open 할 때 마다 sys_out_open 가 동작
    write_cr0(cr0);
    printk("<0>Hooking Success!₩n");
    return 0;
}

void syscall_hooking_cleanup(void)
{
#if 1
    unsigned long cr0 = read_cr0();
    write_cr0(cr0 & ~0x00010000);
    sys_call_table[__NR_open] = orig_call;
    write_cr0(cr0);
    printk("<0>Module Cleanup₩n");
#endif
}

module_init(syscall_hooking_init);
module_exit(syscall_hooking_cleanup);
MODULE_LICENSE("GPL");
```