실험 설계

서버와 클라이언트 프로그램을 C언어로 만든다.

- 두 프로그램 모두 3개의 스레드를 만든다.
- 서버는 각 스레드에서 9001, 9002, 9003 포트의 소켓을 바인딩 및 리스닝을 한다.
- 클라이언트는 서버의 9001, 9002, 9003 포트에 접근하는 소켓을 만든다.
- 연결이 되면 종료신호가 오기 전까지 서버는 recv함수만, 클라이언트는 write함수만 사용한다.
- 서버는 패킷을 받을 때 마다 포트명으로 된(9001.txt, 9002.txt, 9003.txt) 텍스트 파일에 현재 타임스탬프를 찍는다.

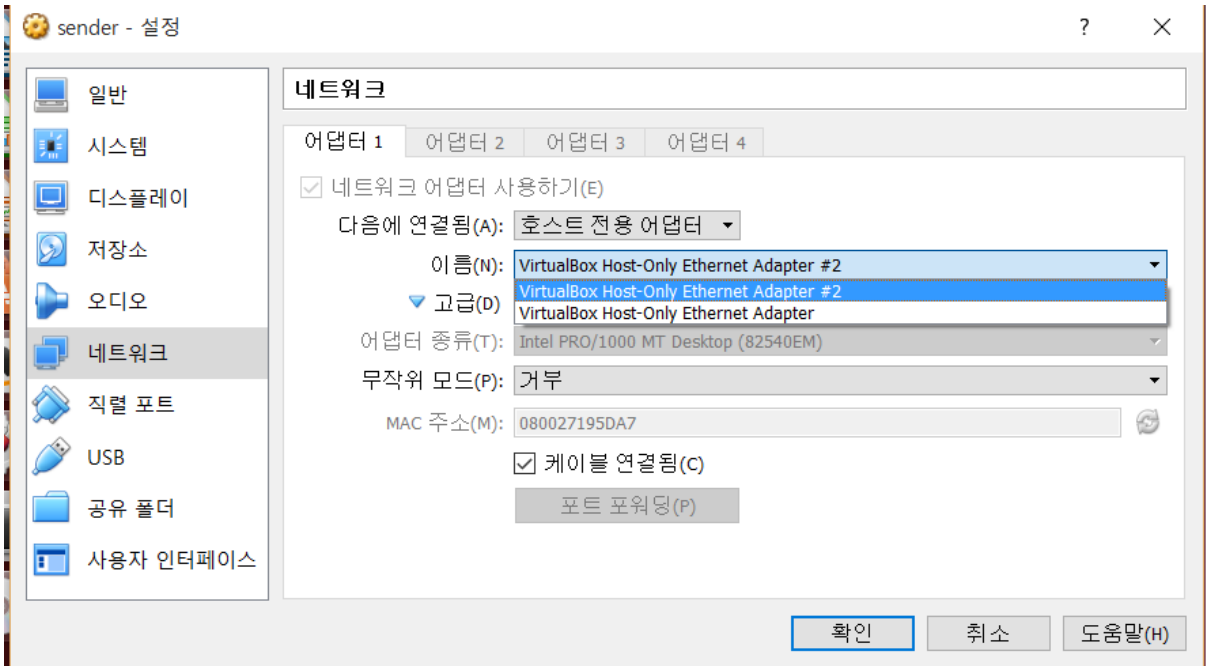
프로그램의 컴파일 및 구동방법은 Readme.txt에 적혀있다.

스레드

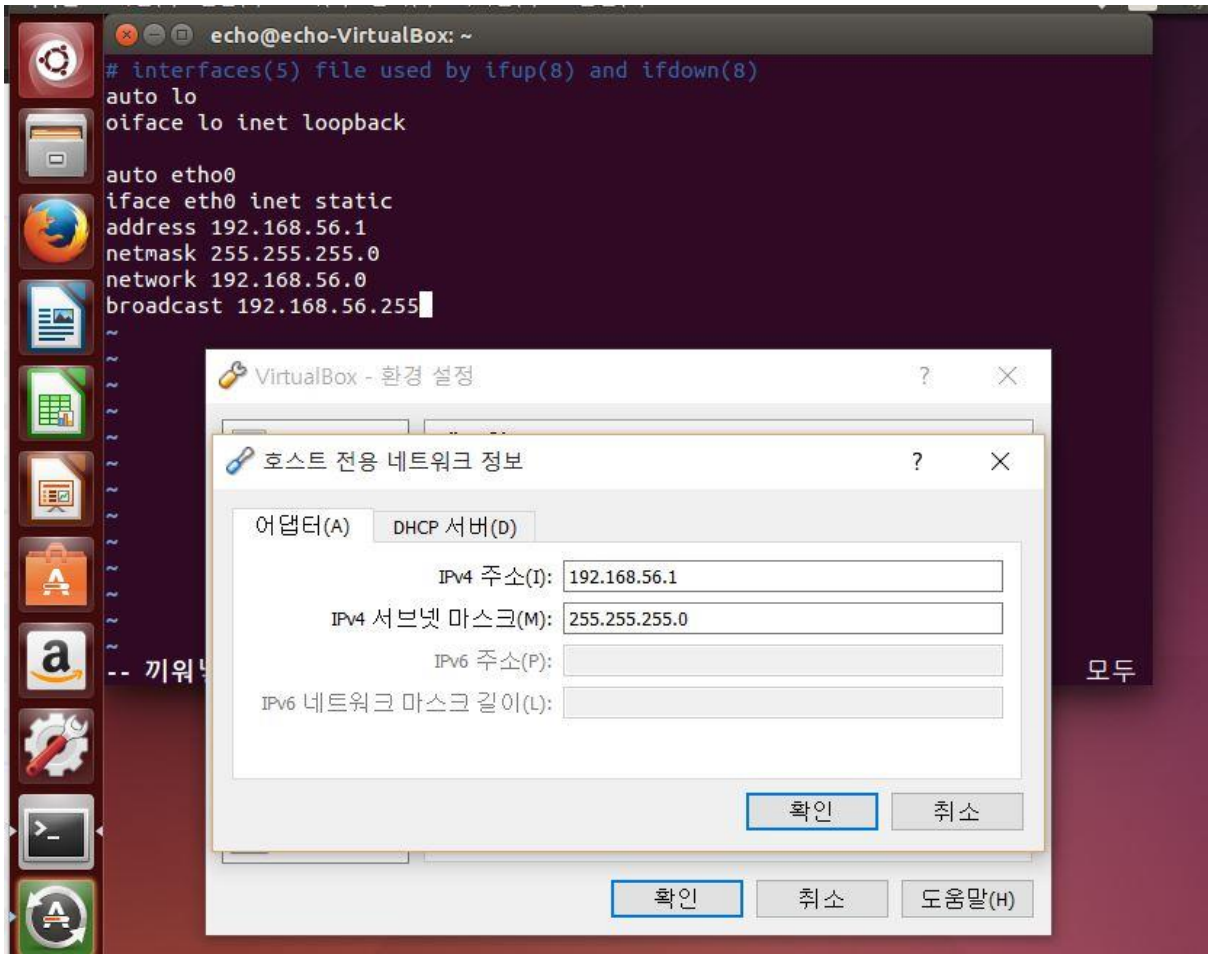
스레드를 만들기 위해서 pthread 라이브러리를 사용하였다. 스레드를 사용하는 이유는 소켓을 blocking 모드로 사용하기 때문에, 3개의 포트를 운용한다고 하였을 경우 동시다발적으로 3개의 포트를 핸들링할 수 없기 때문이다. 스레드는 pthread_create 함수로 생성할 수 있다. 컴파일시에 –lpthread 를 달아줌으로서 라이브러리를 포함하여 정상적으로 컴파일 할 수 있도록 한다.

네트워크

서버와 클라이언트는 각기 다른 가상머신 위에서 돌아간다. 환경을 설정하기 위해서 virtualbox를 다음과 같이 설정한다.



2개의 가상머신을 만든 후, 두 가상머신 모두 네트워크 항목에서 "호스트 전용 어댑터"로 바꾼다. 같은 어댑터로 연결해야 한다.



호스트 전용 네트워크에는 흔히 공유기 머신의 ip주소가 192.168.0.1이듯 192.168.56.1같은 주소로 설정되어있다. 이제 각 가상머신의 ip주소를 192.168.56.N으로 셋팅되도록 하면 된다.

/etc/network/interfaces 문서 내에 아래와 같은 내용을 추가한다.

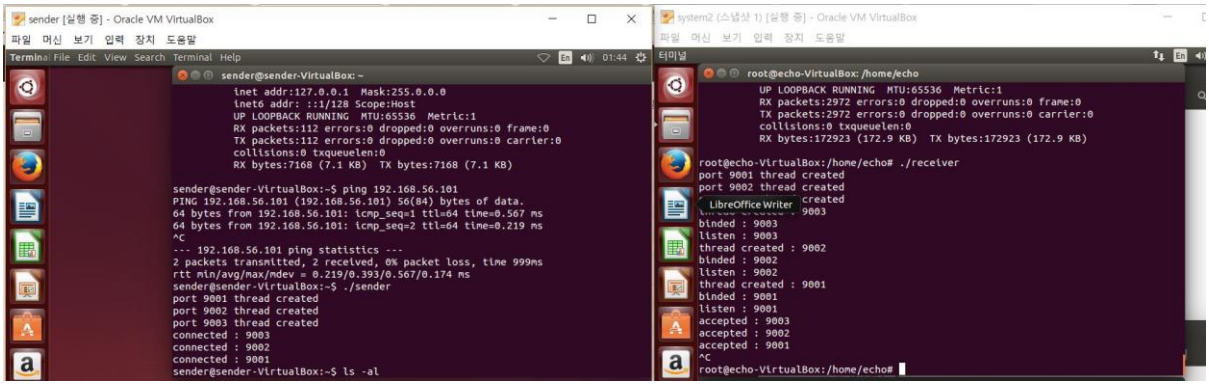
auto eth0

iface eth0 inet static

address 192.168.56.N

netmask 255.255.255.0

본 실험에서는 서버는 192.168.56.3 으로 설정하였다. 클라이언트는 192.168.56.2이다.



서버와 클라이언트 간 정상적으로 연결됨을 확인할 수 있다.

넷필터 커널 모듈

제 1 차 과제에서 우리는 proc 파일을 사용하는 방법을 배웠다. 그것에 착안하여 proc 파일에 port 번호를 넘기면 처리할 수 있는 코드를 만들었다.

```c
static ssize_t my_write(struct file *file, const char __user *user_buffer, size_t count, loff_t *ppos)
{
    // if buffer is full, don't copy them and just return count. it's fake.
    if(bufferOffset + count > MAX_LOG_LENGTH)
        return count;
    if( copy_from_user(stringBuffer + bufferOffset, user_buffer, count) ) {
        return -EFAULT;
    }
    bufferOffset += count;

    // port only >=1000
    if(bufferOffset >= 4)
    {
        stringBuffer[4] = 0;
        long port;
        kstrtol(stringBuffer, 10, &port);
        printk(KERN_INFO "port : %ld was %d, than become %d\n", port, isPortBlocked[port], isPortBlocked[port] != 0 ? 0 : 1);
        if(port >= 1000){
            isPortBlocked[port] = isPortBlocked[port] != 0 ? 0 : 1;
        }
        bufferOffset = 0;
    }

    return count;
}
```

특정 포트 번호가 들어오면 isPortBlocked 라는 배열의 해당 포트번째 변수에 체킹을 한다. 후에 넷필터 후킹함수에서 dest-port 에 해당하는 isPortBlocked 배열내 변수가 체킹되었는지를 확인하는 방식으로 원하는 포트를 잠그기로 한다.

struct nf_hook_ops 의 변수를 하나 선언 한 후, 내부의 값에 적절한 값을 넣음으로서 어떻게 후킹할 것인지를 설정할 수 있다.

본 실험에서는

- main_hook 라는 함수로 후킹이 핸들링되도록
- INET 패킷을
- LOCAL_IN 일 때
- 가장 빨리(PRI_FIRST)

라는 위 4 가지를 셋팅하였다. 그 후 그 변수를 nf_register_hook) 함수를 이용하여 후킹을 등록한다.

```c
netfilter_ops.hook = main_hook;
netfilter_ops.pf = PF_INET;
netfilter_ops.hooknum = NF_INET_LOCAL_IN;
netfilter_ops.priority = NF_IP_PRI_FIRST;

nf_register_hook(&netfilter_ops);
```

Main_hook 함수에서는

- TCP 패킷이며
- Dest-port 가 isPortBlocked 에 체킹되어있으면

위 두가지를 만족하면 drop 이 되도록 하였다. 해당 함수에서 주의해야 할 점은 네트워크 패킷은 엔디안이 다르기 때문에 꼭 htons 혹은 ntohs 함수를 적절히 사용해야 한다.

```
unsigned int main_hook(unsigned int hooknum,
                       struct sk_buff *skb,
                       const struct net_device *in,
                       const struct net_device *out,
                       int (*okfn)(struct sk_buff*))
{

    if(!skb) return NF_ACCEPT;

    struct iphdr* ipHeader = (struct iphdr *)skb_network_header(skb);

    if (!ipHeader) return NF_ACCEPT;

    // not tcp -> pass
    if (ipHeader->protocol != IPPROTO_TCP) {
        return NF_ACCEPT;
    }

    struct tcphdr *tcpHeader= (struct tcphdr *)((__u32 *)ipHeader+ ipHeader->ihl);
    unsigned short int destPort = htons((unsigned short int) tcpHeader->dest);

    printk(KERN_INFO "port : %d\n", destPort);
    if (isPortBlocked[destPort]) {
        printk(KERN_INFO "port %lu droped\n",destPort);
        return NF_DROP;
    }
    return NF_ACCEPT;
}
```

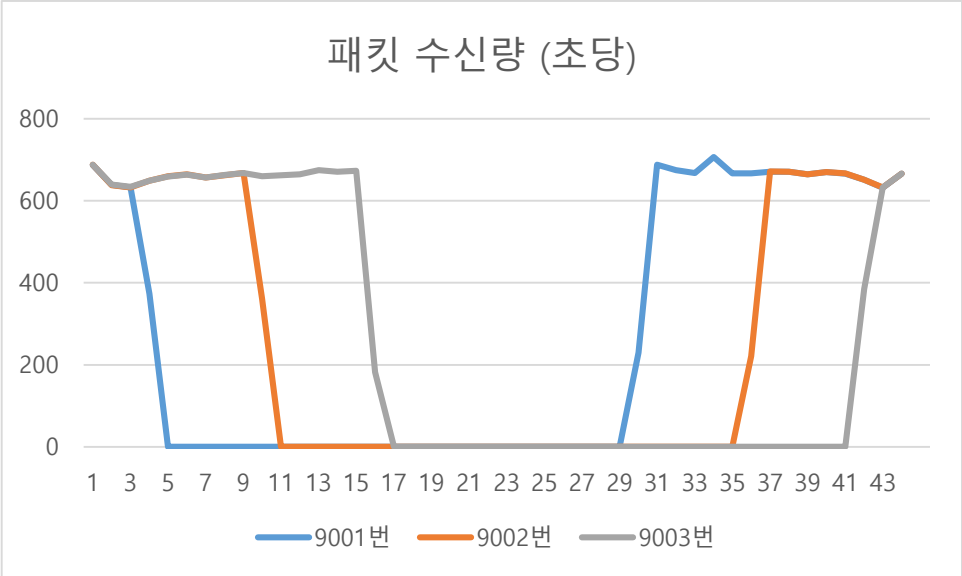
만들어진 모듈을 insmod 명령어로 등록하며, echo 9001 > /proc/myproc/myproc 명령어로 원하는 포트번호를 입력한다.

```
root@sender-VirtualBox:/home/sender/module# insmod simpleModule.ko
root@sender-VirtualBox:/home/sender/module# echo 9001 > /proc/myproc/myproc
root@sender-VirtualBox:/home/sender/module# echo 9002 > /proc/myproc/myproc
root@sender-VirtualBox:/home/sender/module# echo 9003 > /proc/myproc/myproc
root@sender-VirtualBox:/home/sender/module# echo 9001 > /proc/myproc/myproc
root@sender-VirtualBox:/home/sender/module# echo 9002 > /proc/myproc/myproc
root@sender-VirtualBox:/home/sender/module# echo 9003 > /proc/myproc/myproc
```

```
[ 7021.215644] Simple Module Init!!
[ 7023.399112] module file open
[ 7023.399122] port : 9001 was 0, than become 1
[ 7027.631058] module file open
[ 7027.631068] port : 9002 was 0, than become 1
[ 7031.687397] module file open
[ 7031.687410] port : 9003 was 0, than become 1
[ 7034.695011] module file open
[ 7034.695022] port : 9001 was 1, than become 0
[ 7037.151010] module file open
[ 7037.151021] port : 9002 was 1, than become 0
[ 7039.471089] module file open
[ 7039.471100] port : 9003 was 1, than become 0
```

결과물

각 포트별 패킷이 들어온 타임스탬프가 찍혀있는 파일이 3 개(9001.txt, 9002.txt, 9003.txt)가 있다. 이제 이 3 개의 파일을 하나로 묶을 것이다. 1 초에 각 포트당 몇개의 패킷이 들어왔는지를 정리하였다. 그 결과물은 다음과 같다.



본 실험에서는

1. 9001 번 포트를 끄고
2. 9002 번 포트를 끄고
3. 9003 번 포트를 끄고
4. 9001 번 포트를 키고
5. 9002 번 포트를 키고
6. 9003 번 포트를 키고

위 같은 순서대로 진행하였다. 그에 따라 패킷이 정상적으로 도착됨을 알 수 있다. 본 보고서와 함께 위 데이터의 raw text file 을 첨부하였다.