또 다른 리눅스의 세계

이 만용 geoman@nownuri.net 1997년 6월

SGML 편집: 유 성태 alloying@nownuri.net

목 차

제 1 절 SysV 초기화 2

1.1 리눅스의 첫 번째 프로세스는 init . . . 2
1.2 /etc/rc.d 디렉터리 탐험 . . . 3
1.3 제어판의 실행레벨 편집기 사용하기 . . . 6

제 2 절 네트워크 6

2.1 PLIP이란? . . . 6
2.2 IP 매스커레이딩/방화벽 . . . 8
2.3 MS Windows box와 자원 공유 - Samba . . . 10
2.3.1 인터네트워킹 . . . 10
2.3.2 마이크로소프트 제품과 대화하기 . . . 11
2.4 네트워크 기본 보안 설정 . . . 17
2.4.1 리눅스 기본 보안 설정 . . . 18
2.4.2 tcpd 호스트 접근 제한 메커니즘 . . . 21
2.4.3 고질적인 센드메일 문제, RPC 문제 . . . 22
2.5 리눅스, 하나의 완벽한 방화벽 소프트웨어 . . . 22
2.5.1 방화벽(Firewall)이란 무엇인가? . . . 22
2.5.2 준비 사항 . . . 22
2.5.3 리눅스 네트워킹 코드로 할 수 있는 일은? . . . 23
2.5.4 IP 방화벽(IP Firewall) . . . 23
2.6 끝내면서 : 리눅스 보안은 믿을 수 없다? . . . 28

제 3 절 PnP 하드웨어 인식시키기 29

3.1 ISA PnP 도구, 어디서 구할 수 있는가? . . . 29
3.2 ISA PnP 도구의 구성 . . . 29
3.3 ISA PnP 도구의 사용법 . . . 29
3.4 ISA PnP 도구 컴파일하기 . . . 30
3.5 PnP 정보를 덤프하기 . . . 30

3.6 /etc/isapnp.conf 파일 편집하기 . . . 30
3.7 isapnp 명령으로 PnP 설정하기 . . . 33
3.8 원하는 모듈을 띄운다 . . . 33

제 4 절 프로그래밍 33
4.1 쓰레드 프로그래밍을 하기 위해 필요한 것은? . . . 34
4.2 POSIX 쓰레드 . . . 36
4.3 관련된 이야기 . . . 36

제 5 절 리얼타임 리눅스 37
5.1 소프트 리얼 타임 지원 . . . 37
5.2 하드 리얼 타임 지원 . . . 38
5.3 테스트 해보기 . . . 39

제 6 절 리눅스 소프트웨어 개발자들은 요즘 무슨 생각을 하고 있는가? 40
6.1 포팅, 포팅, 포팅 . . . 40
6.2 일반 사용자를 위한 리눅스, 쉬운 리눅스 . . . 40
6.2.1 인터페이스에 대한 개선 고민 . . . 40
6.2.2 공통의 데스크탑 환경? . . . 41
6.2.3 삭막한 리눅스 로고? . . . 41

제 7 절 글을 마치면서 41

제 1 절 SysV 초기화

지금부터는 레드햇 배포판에서 사용하는 SysV 초기화 스타일에 대해서 알아보도록 하겠다. 그 동안 가장 많이 사용되어 왔던 배포판은 역시 패트릭 볼커딩 씨의 슬랙웨어임이 분명하며 통계상으로 볼 때 지금 현재도 그러하다. 하지만 레드햇, 데비안 배포판의 사용자들이 많아지면서 슬랙웨어처럼 캐주얼하고 약간은 아마추어적인 배포판과는 달리 고가의 상용 유닉스 시스템에서나 볼 수 있었던 탄탄한 패키지 관리 시스템, 이로 인해 더 이상 몇 달에 한 번씩 시스템 전체를 다시 설치해야 하는 굴레에서 벗어나게 되었다는 점은 리눅스 세계의 발전이라 하지 않을 수 없다.

슬랙웨어를 쓰고 있는 사람들, 그리고 엉겁결에 슬랙웨어에서 새로운 배포판으로 주목받는 레드햇 배포판으로 옮겨온 사람들은 슬랙웨어와 비교하여 어떠한 부분이 다른 것인지 알고 싶어한다.

전체적으로 배포판 모두는 같은 리눅스이며 다른 리눅스일 수 없다는 점은 분명하다. 많은 사람들이 슬랙웨어, 레드햇, 데비안이 마치 전혀 다른 것인 것처럼 느끼는데 이 모두를 사용해본 필자로서는 결론적으로 '리눅스'라는 점에서 그리고 결국은 모두 관리자(즉 여러분)의 능력에 달렸다고 말하고 싶다. 이미 자신의 리눅스 시스템이 여러분의 부지런한 손길을 받아서 충분히 안정화되어 있다면, 그리고 특별한 이유가 없으면 단지 새로운 배포판이 나왔다고 하여 멀쩡하게 잘 돌아가는 시스템을 날리는 일은 없도록 해야 한다고 생각한다. 배포판 설치쯤이야 이제 남의 도움 없이도 스스로 해낼 수 있는 사람이라면 새로운 배포판을 경험해보는 재미 또한 빼놓을 수 없는 재미일지 모른다.

하지만 필자의 생각으로는 배포판 재미보다는 새로운 응용 기술과 프로그램을 익히는 것이 더 유익한 시간이 되지 않을까 한다. 또는 여가시간 동안 HOWTO 문서를 읽어보거나 한글로 번역되지 않은 문서를 번역해서 다른 한국 리눅서들을 돕는 것이 더 좋은 일이 아닐까 생각한다.

1.1 리눅스의 첫 번째 프로세스는 init

커널이 메모리에 적재되고 제일 먼저 시작하는 프로세스는 바로 init이다. 이 init이 시작하면 리눅스 시스템이 가동되기 시작하는 것이며 이 init이 죽으면 그 시스템은 멈춘 것이다.

```
$ ps aux | less
```

위 명령으로 확인해보라. init의 PID는 1 번이다. 유닉스 시스템을 아는 사람은 이미 알고 있는 내용으로서 모든 프로세스는 결국 init으로부터 복제(fork)되어 생성된 것이다.

```
USER PID %CPU %MEM SIZE RSS  TTY STAT  START TIME
COMMAND
bin   136 0.0 0.0  824   0 ? SW Mar 13  0:00 (portmap)
nobody 189 0.0 0.2 1104  112 ? S  Mar 13  0:00 (httpd)
nobody 190 0.0 0.2 1104  116 ? S  Mar 13  0:00 (httpd)
nobody 191 0.0 0.2 1104  104 ? S  Mar 13  0:00 (httpd)
nobody 192 0.0 0.2 1104  120 ? S  Mar 13  0:00 (httpd)
nobody 193 0.0 0.2 1104  108 ? S  Mar 13  0:00 (httpd)
root   1 0.0 0.4  812  212 ? S  Mar 13  0:04 init
root   2 0.0 0.0   0   0 ? SW Mar 13  0:00 (kflushd)
root   3 0.0 0.0   0   0 ? SW<Mar 13  0:00 (kswapd)
root   4 0.0 0.0   0   0 ? SW Mar 13  0:00 (nfsiod)
root   5 0.0 0.0   0   0 ? SW Mar 13  0:00 (nfsiod)
root   6 0.0 0.0   0   0 ? SW Mar 13  0:00 (nfsiod)
root   7 0.0 0.0   0   0 ? SW Mar 13  0:00 (nfsiod)
root   21 0.0 0.1  796   48 ? S  Mar 13  0:00 /sbin/kerneld
root  103 0.0 0.3  824  164 ? S  Mar 13  0:00 syslogd
root  114 0.0 0.2  972  112 ? S  Mar 13  0:00 klogd
root  125 0.0 0.1  820   84 ? S  Mar 13  0:00 crond
root  147 0.0 0.0  804   16 ? S  Mar 13  0:00 (inetd)
root  158 0.0 0.0  828   4 ? S  Mar 13  0:00 (lpd)
root  170 0.0 0.1  976   76 ? S  Mar 13  0:00 named
root  181 0.0 0.3 1104  160 ? S  Mar 13  0:00 httpd
root  207 0.0 0.4 1220  224 ? S  Mar 13  0:00 sendmail: accepting
```

init의 중요한 임무 중 하나는 시스템을 초기화시키는 것이다. 우선 init은 다음 파일을 살펴보고 자기가 어떤 일을 해야 할지 알아본다.

```
/etc/inittab
```

실행 레벨(Run Level)이라는 개념

리눅스는 실행 레벨이라는 개념을 갖는다. 레드햇 시스템에서 시스템 정지를 의미하는 0 번, 루트만이 로그인할 수 있고 시스템의 문제를 고치게 되는 싱글 유저 모드 1 번, NFS 서비스를 제공하지 않는 다중 사용자 모드 2 번, 그리고 기본값으로 모든 사용자들이 로그인할 수 있고 시스템의 모든 기능이 다 발휘되는 3 번, 초기화하자마자 로그인 화면이 X 윈도 로그인 화면으로 뜨는 5 번, 그리고 마지막으로 리부팅 모드인 6 번이 있다. 4 번은 어디 있는가? 그것은 사용하지 않는다.

initdefault 항목에서 기본값을 정해주게 되어 있는데 절대 0 번이나 6 번으로 정하는 실수를 범하지 말라. 시스템이 부팅되자마자 리부팅되거나 중지해버릴 것이다.

이렇게 실행 레벨이라는 개념을 가지므로 시스템은 여러 가지 상황에 대하여 좀더 유연하게 대처할 수 있고 더욱 논리적인 시스템이 된다.

1.2 /etc/rc.d 디렉터리 탐험

이 디렉터리의 구조가 슬랙웨어와는 전혀 달라 많은 사람들이 어리둥절해 하곤한다.

```
rc.sysinit
```

시스템 초기화시 맨 먼저 딱 한 번 실행되는 초기화 스크립트이다. /etc/sysconfig/network 내용을 불러들인 다음, hostname 명령으로 호스트 이름을 설정한다. 많은 사람들이 레드햇의 제어판(control-panel)에서 네트워크 설정을 하면 어디에 저장되는지 알고자 하는데 바로 기본값이 /etc/sysconfig 밑에 저장되어 있다는 것을 알아두기 바란다.

스왑을 활성화시키고 /etc/fstab 정보에 의거하여 파일 시스템에 이상이 있는지 일단 점검한다. 마운트한다. 만약 심각한 문제가 발생하는 경우에는 (Repair filesystem) # 라는 프롬프트를 내놓고 수동으로 e2fsck를 돌려서 파일 시스템을 점검하도록 요청한다. 레드햇 시스템에서는 파일 시스템을 점검한 후에는 리부팅하도록 되어 있다. 셸 스크립트를 이해하는 사람들은 곰곰히 읽어보기 바란다. 여러분이 다른 용도로 셸 스크립트를 짜는데 많은 도움이 될 것이라 믿는다.

```
rc0.d
rc1.d
rc2.d
rc3.d
rc4.d
rc5.d
rc6.d
```

레드햇 리눅스의 /etc/rc.d 디렉터리의 특징은 바로 여기에 있다. 각 실행 레벨 마다 별도의 디렉터리를 마련해두고 있다. 자, 기본 실행 레벨 3 번에 관계되는 rc3.d 디렉터리에 들어가보라. 그리고 ls -l 해보자.

```
S10network    ->   ../init.d/network
S30syslog     ->   ../init.d/syslog
S40cron       ->   ../init.d/cron.init
S40portmap    ->   ../init.d/portmap.init
S50inet       ->   ../init.d/inet
S50lpd.init   ->   ../init.d/lpd.init
S55named.init ->   ../init.d/named.init
S65httpd      ->   /etc/rc.d/init.d/httpd.init
S75keytable   ->   ../init.d/keytable
S80sendmail   ->   ../init.d/sendmail.init
S85gpm        ->   ../init.d/gpm
S99local      ->   ../rc.local
```

오로지 링크만 존재한다. 그리고 그 링크는 모두 /etc/init.d 디렉터리의 스크립 트들을 가리키고 있다. 다른 디렉터리들에 들어가도 마찬가지이다.

실제 스크립트는 모두 /etc/rc.d/init.d 에 있다.

링크된 이름을 잘 살펴보면 S## 이름의 형식을 띄고 있다는 것을 알 수 있다(여기서 ##는 숫자). rc6.d에 가보면 대부분이 K## 이름의 형식을 가지고 있다는 것도 알 수 있다. 이에 대한 설명은 잠깐 미루기로 하자. 일단 모든 스크립트가 들어있는 /etc/rc.d/init.d 디렉터리로 가본다.

```
cron.init      크론 시작/종료 스크립트
functions      다른 스크립트에 사용되는 유틸리티 함수
gpm            텍스트 모드 마우스 시작/종료 스크립트
halt           halt 또는 reboot 스크립트
```

```
httpd.init     웹 서버 시작/종료 스크립트
inet           inet 수퍼 서버 시작/종료 스크립트
keytable       키보드 테이블 지정 시작/종료 스크립트
killall        죽지 않는 프로세스 강제 종료 스크립트
lpd.init       BSD 프린팅 데몬 시작/종료 스크립트
named.init     로컬 네임서버 데몬 시작/종료 스크립트
network        네트워크 업/다운 스크립트
news           뉴스서버 업/다운 스크립트
nfs            NFS 서비스 시작/종료 스크립트
nfsfs          NFS 마운트 스크립트
pcmcia         PCMCIA 시작/종료 스크립트
portmap.init   포트맵 데몬 시작/종료 스크립트
random         난수 발생 스크립트
sendmail.init  메일 서버 시작/종료 스크립트
single         싱글 모드 진입 스크립트
skeleton       레드햇 init 스크립트 뼈대
syslog         syslog 시작/종료 스크립트
```

만약 웹 서버를 멈추게 하고자 한다면 다음과 같이 한다.

```
# ./httpd.init stop
```

다시 시작하려면 다음과 같이 입력한다.

```
# ./httpd.init start
```

상당히 이성적이지 않은가? 시작하려면 start를, 멈추려면 stop을 셸 스크립트의 전달 인수로 준다. 만약 여러분이 어떤 서비스에 대한 초기화 스크립트를 만들려고 한다면 다음의 skeleton 스크립트를 기본 뼈대로 하고 나머지 스크립트를 참고하여 start에 대해서는 서비스 시작 명령을, 그리고 stop에 대해서는 서비스 종료에 관계된 명령을 적어주도록 한다.

```
# 스크립트에 사용되는 함수 라이브러리 루틴을 소스(source)한다
. /etc/rc.d/init.d/functions

# 이 부분을 유심히 살펴보기 바란다.
case "$1" in
 start)
    touch /var/lock/subsys/skeleton
    ;;
 stop)
    rm -f /var/lock/subsys/skeleton
    ;;
 *)
    echo "Usage: skeleton {start|stop}"
    exit 1
esac

exit 0
```

그럼 rc?.d 디렉터리의 각 심볼릭 링크 파일의 이름은 어떤 의미를 갖는가? S로 시작하는 심볼릭 링크 파일은 가리키는 스크립트에 start가 붙어서 실행된다. 즉 시작 스크립트이다. K로 시작하는 심볼릭 링크는 가리키는 스크립트에 stop이 붙어서 실행된다. 즉 종료 스크립트이며 Kill을 의미한다. 실제로 어떻게 호출되고 있는지를 알아보기 위해서는 /etc/rc.d/rc 스크립트를 읽어보면 된다.

여러분이 원하는 단 한 번의 실행 명령은 rc.local에!

시스템 초기화할 때 딱 한 번 실행되길 원하는 명령은 아무 곳에나 두지 말고 rc.local 파일 뒷부분에 추가한다. 필자는 다음을 추가하였다.

```
modprobe slhc
modprobe ppp            PPP 기능 로딩
modprobe bsd_comp
modprobe ppp_deflate
modprobe ip_masq_ftp    FTP 매스커레이딩 기능 로딩
modprobe aic7xxx        아답텍 SCSI 컨트롤러 인식
modprobe sbpcd          AT버스 사블2배속 인식
```

rc.local 스크립트를 잘 보면 여러분이 로그인할 때 볼 수 있는 Red Hat Linux release 4.1 (Vanderbilt)와 같은 메시지가 설정되는 부분을 이해할 수 있을 것이다.

1.3 제어판의 실행레벨 편집기 사용하기

```
그림 1: 제어판
-------------------------------------------------------------
File                                                     Help
-------------------------------------------------------------
init.d     0 번    1 번    2 번    3 번    4 번    5 번    6 번
           시작    시작    시작    시작    시작    시작    시작
-------------------------------------------------------------
Add
Remove     0 번    1 번    2 번    3 번    4 번    5 번    6 번
Edit       종료    종료    종료    종료    종료    종료    종료
Execute
-------------------------------------------------------------
```

제어판의 화면 구성은 위와 같다. 맨 왼쪽 상단 창에는 /etc/rc.d/init.d 디렉터리에 있는 모든 스크립트가 나타나 있다. 그 스크립트를 몇 번 레벨의 시작 또는 종료에 넣을 것인지 결정하면 된다. 하나를 선택하고 추가(Add) 버튼을 누르면 어느 실행 레벨에 추가할 것인지 물어온다. 그리고 시작 스크립트인지 아니면 종료 스크립트인지 물어오게 될 것이다. 그리고 나서는 두 자리 숫자를 적어주게 되는데 그 숫자가 낮을 수록 우선적으로 실행된다. 제거(Remove) 버튼은 각 실행 레벨에서 선택한 후 선택한다. 물어보지 않고 지우므로 주의하기 바란다.

편집(Edit) 버튼은 각 스크립트의 우선 순위 번호를 편집하는 것이다. 실행 버튼 (Execute)은 일단 init.d 창에서 원하는 스크립트를 선택한 후 start/stop을 선택한다.

소형 시스템에서는 그렇게 매력적으로 보이지 않을지 모르나 대형 서버에서는 관리자로 하여금 체계적인 시스템 관리를 하게 해줄 수 있는 훌륭한 GUI 도구라고 본다. 지금까지 독특한 레드햇 SysV 초기화 시스템을 알아보았다.

제 2 절 네트워크

리눅스에서 제일 재미있게 즐겨볼 수 있는 것이 네트워크가 아닐까? 그리고 요즘은 많은 곳에서 리눅스는 '웹서버 운영체계' 라고 부를 수 있을 만큼 가장 안정적인 웹 서버 머신으로서 각광받고 이제 '리눅스는 훌륭한가? 안정적인가? 믿을 만한가?' 라는 쓸데없는 질문이 사라진지 오래이다. 리눅스의 성공을 싫어하는 많은 사람들에게 얄미운 일인지 모르나 리눅스는 바로 우리 자신의 운영체계이고 언제나 열려있으므로 가장 높은 성장 속도를 보여주고 있으며 자유를 사랑하고 컴퓨터 속에서 자유를 실현하고자 하는 사람들이 있는 한 전진은 멈추지 않을 것이다.

2.1 PLIP이란?

이번에는 리눅스 커널에서 지원하는 아주 독특한 기능으로서 랜 카드없이도 두 대의 컴퓨터를 아주 값싸게 그것도 TCP/IP 프로토콜로 간단히 연결해 텔넷, FTP, NFS 등 모든 기능을 쉽게 사용할 수 있는 방법의 하나인 '페러렐 라인 인터넷 프로토콜 PLIP'에 대해서 알아보겠다.

준비물

프린터 포트를 연결하는(일반적으로 도스 시절 FX 케이블이라고 부름) 페러렐 널 케이블을 구입한다. 용산 컴퓨터 상가, 특히 케이블 전문 상점에서 쉽게 구할 수 있을 것이다. 약 2 천원   2 천 5 백원 정도면 구입할 수 있다고 안다. 컴포트를 연결하는 것이 아니라 프린터 포트를 연결하는 케이블임을 정확히 알려주어야 한다.

리눅스 컴퓨터 2 대 또는 리눅스 머신 한 대와 도스 머신 한 대 그리고 커널 컴파일할 시간과 자신감, 끈기가 있으면 된다.

커널 컴파일하기 또는 모듈 로딩하기

```
# cd /usr/src/linux
# make xconfig
```

지금 리눅스 커널 컴파일에서 가장 많이 사용되는 것은 역시 모듈 컴파일이다. 클래식한 설정(Yes 또는 No)과 달리 이제는 주로 사용하지 않는 기능 또는 서로 충돌을 일으키는 기능을 모듈(M)로 설정하여 시스템 운영중 자유자재로 활성화, 비활성화시킬 수 있다.

PLIP의 경우에는 페러렐 포트를 사용하기 때문에 프린터를 사용하는 사람들은 한 번에 두 가지 기능을 모두 사용할 수는 없다는 문제를 갖는다. 물론 페러렐 포트가 두 개 달린 경우에는 문제가 달라지겠지만 대부분의 PC들은 단 하나의 페러렐 포트를 가지고 있다.

바로 여기서 중요한 것 한 가지가 있다. 여러분이 프린터/PLIP을 그때마다 사 용하려면 두 기능을 모두 모듈로 설정해야 한다. 그리고 나서 프린터를 쓸 때 는 PLIP 모듈을 빼내고 프린터 기능을 집어넣고 PLIP을 쓸 때는 반대과정을 거쳐야 한다. 또한 페러렐 버전의 ZIP 드라이브 같은 것을 사용하는 사람들은 당연히 ZIP 기능도 모듈로 설정해야 한다.

```
# make dep ; make clean
# make zImage 또는 make zlilo
# make modules ; make modules_install
```

두 번째 make zImage 과정은 여러분의 커널에 이미 프린터 지원이나 ZIP 기능이 정적으로 들어가 있는 경우이다. 요즘의 배포판들은 대부분 모듈 방식을 취하고 최대한 부차적 기능들을 모두 모듈화시켜놓고 있기 때문에 3 번째 과정만 해주어도 무방하다.

모듈을 만들고 설치하면 /lib/modules/<커널 버전> 디렉터리에 설치된다. PLIP 은 네트워킹 관련이므로 net 디렉터리에 있다.

```
total 119
-rw-r--r-- 1 root root 9112 Feb 20 22:01 3c509.o
-rw-r--r-- 1 root root 5464 Feb 20 22:00 bsd_comp.o
-rw-r--r-- 1 root root 6044 Feb 20 22:01 dlci.o
-rw-r--r-- 1 root root 2160 Feb 20 22:01 dummy.o
-rw-r--r-- 1 root root 4320 Feb 20 22:00 new_tunnel.o
-rw-r--r-- 1 root root 12532 Feb 20 22:00 plip.o
-rw-r--r-- 1 root root 24268 Feb 20 22:00 ppp.o
-rw-r--r-- 1 root root 34176 Feb 20 22:01 ppp_deflate.o
-rw-r--r-- 1 root root 6440 Feb 20 22:00 slhc.o
-rw-r--r-- 1 root root 8548 Feb 20 22:01 slip.o
```

PLIP을 사용하려는 두 대의 컴퓨터를 페러렐 널 케이블로 연결한다. 그리고 나서 두대 모두 insmod, modprobe 명령으로 plip.o 모듈을 커널 기능으로 활성화시킨다.

/etc/hosts 작성하기

TCP/IP 네트워킹 연결이므로 두 대는 서로를 구별할 수 있는 주소를 가져야 한다. 두 대의 컴퓨터 /etc/hosts 파일 뒤에 다음 내용을 추가하자.

```
192.168.1.1     one
192.168.1.2     two
```

케이블로 연결한 한 쪽의 주소를 192.168.1.1로 정하고 이름은 그냥 one이라고 하기로 한다. 나머지 한 쪽은 192.168.1.2를 주소로 하고 two라고 부르기로 하였다.

PLIP 인터페이스 활성화하기

one 즉 200.0.0.1이라는 주소를 부여받은 한 쪽 리눅스 머신에서 다음과 같이 한다.

```
# ifconfig plip1 one pointopoint two up
# route add two plip1
```

ifconfig 명령은 잘 알아두어야 할 명령이다. 어떤 네트워크 인터페이스든 바로 이 명령을 가지고 작동시킨다. plip1이라는 인터페이스를 만든다. one과 two를 일대일(P-t-P)로 연결한다. 그리고 나서 라우팅 정보를 수동으로 부여하는데 plip1에다가 two로 가는 경로를 설정한다. 목적지가 two이면 PLIP 인터페이스로 패킷을 내보내게 된다. two라는 이름의 컴퓨터에서는 반대 과정을 거쳐주면 된다.

```
# ifconfig plip1 two pointopoint one up
# route add one plip1
```

자! 이렇게 하면 두대는 TCP/IP 프로토콜로 연결된 것이다. 서로 텔넷, FTP를 해보자. NFS 설정을 할 수 있는 사람은 서로 NFS 마운트 할 수도 있다. 넷스케이프도 마찬가지이다. 두 쪽에서 운영하는 웹 서버에 연결 하는 것도 가능하다.

```
$ telnet two
$ ftp two
```

PLIP을 마치고 싶을 때는 다음과 같이 한다.

```
# ifconfig plip1 down
```

인터페이스를 활성화시킬 때 마지막에 up이라고 했듯이 이번에는 down이다. 그리고 나서 rmmod 명령으로 모듈을 빼낸다.

```
# rmmod plip
```

몇 가지 남는 문제들

PLIP은 개선의 여지가 정말로 많은 기능이라고 본다. 텔넷 세션에서는 느낄 수 없으나 만약 두 컴퓨터간의 FTP 세션으로 몇 메가씩 되는 파일을 전송할 때는 빈번한 인터럽트 발생으로 인해 두 시스템의 부하가 정말 많이 걸려서 도저히 정상적인 작업을 할 수 없을 정도가 된다.

이는 어찌 보면 당연한지도 모른다. 이서네트 카드와 같은 하드웨어적 장치가 아니기 때문이며 2천원짜리 네트워크의 한계일까? PLIP이 유용하게 쓰일 만한 환경은 아마도 한 대의 데스크탑과 랩탑 컴퓨터 사이의 연결이 아닌가 생각한다.

2.2 IP 매스커레이딩/방화벽

리눅스 커널의 네트워크 기능 중 방화벽(Firewall) 기능을 알아보자. 이것을 사용하면 어떤 일이 가능한지 알아보겠다.

상황

지금 현재 여러분의 사무실에는 4 대의 컴퓨터가 있다. 그리고 그 컴퓨터들은 모두 이서네트 카드로 연결되어 서로 연결 가능하다. 그런데 사무실에 할당된 공적 IP 주소는 단 한 개뿐이다. 가장 성능이 좋고 비싼 컴퓨터에 외부와의 연결선을 이어서 사용중이다. 이 주소를 예를 들어 210.126.77.222라고 하자. 내부 네트워크는 192.168.1.1부터 시작해서 192.168.1.4까지 사용하여 구축되어 있다. 당연히 인터넷 즉, 외부와의 접속은 오로지 외부와 직접 연결되어 210.126.77.222라는 주소를 갖는 컴퓨터에서만 가능하다. 오로지 그 컴퓨터에서만 텔넷, FTP, 웹이 가능한 것이다.

시스템 관리자는 이런 생각을 해본다. 공적인 주소 210.126.77.222, 내부적인 주소 192.168.1.1이라는 주소를 갖는 컴퓨터 뿐 아니라 단순히 그 컴퓨터와 연결되어 있을 뿐인 3 대의 컴퓨터에서도 인터넷이 가능할 수는 없을까? 직접 다른 컴퓨터에서 텔넷, FTP를 해보았다. 하지만 전혀 될 기미가 보이질 않는다. 그 이유는 무엇일까? 192.168.1.2라는 내부 주소만을 갖는 컴퓨터에서 외부로 텔넷을 한다고 치자. 외부로 나가기 위해서는 결국 외부와 연결되어 있는 컴퓨터를 게이트웨이(gateway)로 경유하여 나가게 된다. 그리고는 다시는 돌아올 수 없다. 왜냐하면 외부세계로 나간 192.168.1.2라는 주소의 패킷은 외부 세계에서 볼 때 전혀 알 수 없는 주소이며 도대체 어떤 경로를 통하여 192.168.1.2에게 패킷을 돌려보낼 수 있는지 해결할 수 없기 때문이다.

해결책

해결책은 있다. 외부와 연결되어 210.126.77.222라는 IP 주소를 가진 컴퓨터에서는 패킷이 나가고 들어오는 것이 가능하므로 다른 3 대의 컴퓨터들에서 나가는 패킷들이 마치 210.126.77.222에서 나간 것처럼 가장(Masquerade)시키면 될 것이다. 그러면 일단 패킷이 나갔다가 돌아올 수 있기 때문이다. 여기서 한 가지 문제는 이제 4 대의 컴퓨터 모두가 210.126.77.222라는 주소를 가장하고 패킷을 내보낸 것까지는 좋으나 돌아온 패킷에 대하여 그것이 실제로는 4 대 중 어느 컴퓨터를 위한 것인지 구별해낼 필요가 생긴다. 이것을 구별하지 못하면 결국 보내기는 보내도 받는 측은 오로지 210.126.77.222이므로 우리가 원하는 결과는 얻지 못한 것이다.

이에 대한 해결책을 리눅스가 제시하고 있다. 외부와의 연결 통로에 존재하는 컴퓨터에 리눅스가 설치되어 내부 네트워크에서 나가는 패킷에 대하여 IP를 가장해주고 들어오는 패킷에 대해서는 그것이 내부 네트워크 어디로 보내져야 하는지 판별하는 것이 가능하다.

준비

이미 리눅스를 조금 써본 사람들은 눈치챘을 것이다. 어떻게 보면 리눅스의 모든 일은 커널 컴파일에서 시작해서 커널 컴파일로 끝나는 듯 하다. 여러분이 원하는 어떤 일이 있는가? 그러면 커널 컴파일에서 그 답을 찾으라. 커널 컴파일 중 네트워크 옵션(Network Options) 항목을 들어가보자.

```
[*] Network firewalls
[*] TCP/IP networking
[*] IP: forwarding/gatewaying
[*] IP: firewalling
[*] IP: masquerading (EXPERIMENTAL)
[*] IP: always defragment
[*] Dummy net driver support
```

이것은 네트워크와 외부의 사이에서 방화벽 역할을 해주는 리눅스 머신 하나에서만 해주면 된다.

실행

커널 2.0.0 이상에서는 ipfwadm이라는 유틸리티를 사용한다. 방화벽 역할을 하는 컴퓨터에서 192.168.1.0 네트워크에 대하여 외부와 차단을 시키면서 내부 네트워크에서 나가는 패킷을 마치 방화벽 주소

210.126.77.222에서 나가는 것처럼 속여주고 또한 외부에서 들어오는 패킷이 실제 내부 네트워크 중 어디로 가야 하는지 판별해주는 역할을 하기 위해서는 다음과 같이 한다. 이 유틸리티는 슬랙웨어, 레드햇, 데비안 배포판에 모두 기본적으로 들어있다.

```
# ipfwadm -F -p deny
# ipfwadm -F -a m -S 210.126.77.0/24 -D 0.0.0.0/0
```

리눅스 방화벽 컴퓨터와 연결된 컴퓨터들은 모두 방화벽을 거쳐서 나가게 될 것이므로 각자의 기본 게이트웨이를 리눅스 머신으로 설정하라.

```
route add default gw 192.168.1.1
```

이렇게 해주면 192.168.1.0 네트워크에 있는 모든 컴퓨터에서 자유롭게 인터넷 사용이 가능하다. 외부에서 볼 때는 192.168.1.0이라는 네트워크는 보이지 않으며 오로지 210.126.77.222라는 주소를 갖는 한 대의 컴퓨터만 보일 뿐이다. 물론 모든 패킷이 방화벽을 거쳐 나가므로 방화벽에 걸리는 부하는 이루 말할 수 없다.

주의할 것 하나

```
-rw-r--r-- 1 root root 3012 Feb 20
22:03 ip_masq_ftp.o
-rw-r--r-- 1 root root 2640 Feb 20
22:03 ip_masq_irc.o
-rw-r--r-- 1 root root 2572 Feb 20
22:03 ip_masq_raudio.o
modprobe ip_masq_ftp
```

모듈 컴파일을 하고 나서 /lib/modules/<커널버전>/ipv4 디렉터리에 가보면 위와 같은 모듈을 발견할 수 있다. 처음부터 각각 FTP, IRC, 리얼 오디오 기능에 대한 매스커레이딩을 위한 별도의 모듈이다. 내부 네트워크에서 FTP를 제대로 하기 위해서는 바로 ip_masq_ftp 모듈을 방화벽 리눅스 머신에 적재해주어야 한다. IRC, 리얼 오디오 또한 마찬가지이다.

2.3 MS Windows box와 자원 공유 - Samba

일반 사용자라면 넷스케이프나 메일 읽기 프로그램 등의 애플리케이션 사용법을 약간의 중급 사용자라면 자기 스스로 TCP/IP 접속 설정을 익혀야 한다. 관리자 수준의 고급 사용자라면 연결된 모든 컴퓨터 간의 원활한 정보 교통을 위하여 적절한 라우팅(Routing) 설정, 각종 서버 설정을 익혀야 할 것이다. 이 장에서 다루는 내용의 대상은 바로 중고급 사용자를 위한 내용임을 밝혀둔다.

2.3.1 인터네트워킹

여러분도 모두 몸으로 느낄 수 있을 만큼 지금 컴퓨터는 본격적인 네트워킹의 시대로 돌입한 상태이다. 몇 년 전만 해도 개인용 컴퓨터의 연결은 잠시 파일을 주고받는데 사용하기 위하여 전화선을 이용한 모뎀 연결만이 전부인 듯 알고 있었고 전용회선에 의한 항상적인 연결이란 일반인들이 범접하기 힘든 고가의 유닉스/메인프레임 시스템에서나 볼 수 있는 것이라고 생각한 적이 있었다는 것을 기억할 것이다. 이제는 학교나 사무실에서 이서네트 네트워크 카드를 통해 한 방 내부 또는 한 건물 내부의 모든 컴퓨터들이 연결되어 있는 경우가 많고 여건이 되는 경우에는 ISP로부터 전용 인터넷 회선을 끌어와 인터넷이라 는 망망대해와 연결하는 경우도 종종 볼 수 있다.

이제 컴퓨터를 배운다는 것은 예전처럼 책 하나 놓고 명령어를 익히고 프로그램 몇 개를 익히는 것에서 그치지 않고 네트워크라는 필수 과목까지 포괄하게 되었다.

네트워크는 둘 이상의 존재를 전제로 한다. 따라서 하나 중심의 사고에서 탈피해야 할 필요가 있다. 각 나라 사람들이 자기 나라에서만 산다면 그리고 충분히 자급자족할 수 있고 다른 나라와 교역할 필요 없이 충분히 잘

살 수 있다면 국어 이외의 외국어 과목은 필요치 않을 것이다. 오로지 자기 나라 언어만 잘 하면 생활에 불편함이 없다. 하지만 일이란 게 실제 그렇지 않다. 엄연히 국경선 너머 자기가 사용하는 언어와는 전혀 다른 언어를 사용하는 사람들이 있고 그들은 우리와 다른 자연 환경, 문화적 환경에서 살고 있고 우리에게 없는 것을 가지고 있다.

원하는 무엇을 얻기 위해 그 나라로 넘어갔다고 하자. 상식을 가진 사람이라면 그 나라에 가서는 당연히 그 나라 언어를 사용해야 한다. 그들이 여러분의 언어를 사용하리라고 생각할 수 있는가? 또는 두 나라에서 공통적으로 사용되는 언어를 사용해야 한다. 바로 네트워킹의 세계가 그러하다. 일반적으로 각 운영 체계들은 자신만의 약속체계를 가지고 있다. 마이크로소프트 사의 제품들인 윈도 포 워크그룹 3.11, 윈도 95, 윈도 NT는 그들 서로 간의 약속체계를 가지고 있어서 원하는 메시지를 주고받을 수 있도록 고안되어 있다. 기존의 랜 환경에서 많이 사용되고 있는 노벨 네트웨어는 또한 그들 제품에서만 사용되는 고유의 약속체계를 가지고 있다. 이렇게 회사마다 서로 다른 약속체계를 가지고 있다. 그 회사들 사이에 특별한 노력이 없는 한 그 약속체계는 서로 같을리 없고 비슷하지도 않을 것이다. 약속체계가 서로 다른 컴퓨터를 연결하여 원하는 메시지를 주고받는다는 것도 어불성설이다.

여러분의 귀에 박히도록 듣고 있는 TCP/IP는 그 많은 약속체계 중 하나이다. 약속체계의 범위와 성격이 다르긴 하지만 마이크로소프트 사에서 사용하고 있는 약속체계, 프로토콜은 NetBIOS 또는 랜 매니저(LanManager)라고도 불리는 SMB(Session Message Block)이다. 노벨 사에 의해 사용되고 있는 그들만의 프로토콜은 NCP(Network Control Protocol)이라고 불린다. 애플 사에서는 애플토크(AppleTalk)라는 자신만의 약속체계를 갖는다(여러분이 자신을 중급 사용자라고 생각하기 원한다면 실제로 쓰지는 않는다 할지라도 이 기회에 이런 프로토콜이 있다는 사실만은 기억해두기 바란다). 썬마이크로씨스템즈 사에서 처음 개발된 네트워크 파일 시스템인 NFS(Network File System)는 지금 거의 모든 유닉스 시스템에서 채용되고 있는 프로토콜이다. 단지 TCP/IP는 물론 유닉스 기반의 컴퓨터들에 의해서 사용되는 개별적인 프로토콜이긴 했지만 좀 더 광범위하게 사용될 수 있도록 고안되었고 처음부터 그런 생각을 염두에 두고 만들어진 것이며 지금 현재 어떤 컴퓨터를 막론하고 인터넷이라고 하는 전세계적 네트워크에 연결하고자 하는 컴퓨터라면 당연히 따라야 할 표준이 되어 있다는 점에서 다른 프로토콜과 그 중요성을 달리 할 뿐이다.

한 번 쯤은 이런 생각을 갖게 된다. 왜 이렇게 많은 프로토콜이 존재하는 것일까? 대답은 간단하다. 사람들의 생각에 만장일치란 있을 수 없기 때문이다. 우선 기술적인 면에 있어서 기존의 어떤 프로토콜에 부족한 점이 있다면 그리고 그것을 단순히 개선하는 것만으로는 해결할 수 없다고 판단할 때는 전혀 다른 기초 위에 건설하려고 하기 때문이다. 이렇게 해서 서로 개념도 완전히 다르고 전혀 호환의 여지가 없는 프로토콜들이 생겨난다. 두 번째 이유도 또한 중요한 이유로서 각 회사의 정치적인 입장을 들 수 있다. 철저하게 자기 고객의 발목을 묶어두기 위한 방편으로 현재 충분히 좋은 프로토콜이 있음에도 불구하고 일부러라도 전혀 다른 프로토콜을 고안하는 것이다. 쓸만한 프로토콜이 있어서 상대방 회사의 프로토콜을 사용한다면 경쟁 상대를 인정하게 되는 결과이므로 일부러라도 호환되지 않는 자신만의 프로토콜을 만든다. 프로토콜이란 하나의 언어이고 자신의 정체성을 확립해주는 것이므로 더욱 더 중요하게 느껴진다.

각 회사들은 정치적인 이유로 인해 고집스럽게 다른 회사와는 호환되지 않는 자신만의 프로토콜을 갖으려 한다. 모든 회사들은 마음속으로 자신의 프로토콜 이 전세계적 표준이 되길 바랄 것이다. 이런 의미에서 TCP/IP를 제외한 다른 프로토콜은 서로 간의 우열을 가리기 힘들다. 기술적 이유보다는 정치적인 이유에서 다르게 만든 것일 뿐이기 때문이다. 노벨 사의 NCP는 노벨 네트웨어에 가장 알맞은 프로토콜일 것이며 애플 사의 애플토크는 당연히 자신의 운영체계인 시스템 7에 알맞은 것일 테며 마이크로소프트 사의 SMB는 오로지 윈도 제품군에만 해당되는 얘기이다. 그들이 자신의 프로토콜을 고집 하는 이유는 여러분이 금방 이해할 수 있을 것이다. 한 사무실에서 노벨 사 제품만 쓰든지 아니면 애플 사, 마이크로소프트 제품만 쓰라고 우리에게 강요하고 있다.

2.3.2 마이크로소프트 제품과 대화하기

이번에 다루고자 하는 내용은 바로 SMB라는 자체적 프로토콜을 사용하는 마이크로소프트 제품과의 정보 공유 방법이다. 지금 현재 개인용 컴퓨터에서 가장 많이 사용되고 있는 운영체계 제품은 윈도 3.1, 윈도 95 등의 마이크로소프트 제품이다. '사용자 편이성'이 마치 자신만의 트레이드마크인 양 '비주얼, 쉽게쉽게'를 외치고 있는 그들 제품은 그 속사정이 어떠하든 간에 개인용 컴퓨터 데스크탑 시장에서는 독보적인 위치를 차지하고 있음이 분명하다.

여러분의 학교, 사무실에는 이미 그렇게 많은 마이크로소프트 제품들로 운영되는 컴퓨터들이 있을 것이다. 그곳에 리눅스가 들어서고 있다. 여러분도 알다시피 리눅스는 리눅스 나름의 장점과 약점을 가지고 있다. 많은 사람들은 리눅스에게서 리눅스만의 충분한 장점을 살리고 기존의 마이크로소프트 제품에서는 그들만의 장점을 살리길 원한다. 두 가지를 서로 다른 용도로 사용하고 있는데 간혹 우리는 판이하게 다른 그 두 시스템 자원을 서로 공유하여 사용하고자 할 때가 있다. 이럴 때는 어떻게 해야 하는가?

마이크로소프트 제품 중 리눅스와의 공유를 목표로 하는 제품은 하나도 없다. 그들에게서 그런 일을 기대할 필요는 없다. 그들의 관심밖에 있는 일이며 조금의 관심이 있다 하더라도 정치적인 입장으로 인해 그런 일은 상당 기간 일어나지 않을 것이 분명하다. 다행히도 리눅스/유닉스 안에서 해결의 노력이 있어왔고 그 중 하나가 바로 삼바(Samba)라는 패키지이다.

개념은 간단하다. 삼바라는 패키지는 마이크로소프트 제품이 사용하고 있는 SMB라는 프로토콜을 이해하고 있으며 바로 SMB라는 언어로 이야기할 줄 아는 프로그램들의 모음이다. 적절하게 두 컴퓨터가 연결되고(일반적으로 이서네트 연결) 마이크로소프트 제품에서 리눅스 쪽으로 또는 그 반대로 어떤 메시지를 전달할 때 리눅스 쪽에 삼바를 설치해두면 마이크로소프트 제품 입장에서는 자신이 대화하고 있는 상대가 자사 제품과는 전혀 상관없는 리눅스라는 사실조차 알 리 없다. 영어를 쓰는 나라에 가서 유창하게 영어를 사용하는 것과 같다. 주의할 것은 여러분의 윈도 제품에 TCP/IP 프로토콜 지원이 되어 있어야 한다는 점을 들 수 있다.

단도직입적으로 삼바를 사용하며 다음과 같은 일이 가능하다.

1. 리눅스 드라이브를 윈도 머신과 공유한다.
2. 윈도 드라이브를 리눅스 머신과 공유한다.
3. 리눅스 프린터를 윈도 머신과 공유한다.
4. 윈도 프린터를 리눅스 머신과 공유한다.

삼바를 구할 수 있는 방법

삼바는 다음 FTP 사이트에서 구할 수 있다. `nimbus.anu.edu.au/pub/tridge/samba/` 삼바는 현재 리눅스 사용자들에게 있어 중요한 패키지로 인식되어 있기 때문에 레드햇, 데비안을 포함한 중요 배포판에서 기본적인 패키지로 제공하고 있다. 소스의 경우에는 리눅스의 모든 소스들이 그러하듯 README를 꼼꼼하게 읽어보면 된다. 고스트스크립트 컴파일처럼 어렵거나 하지는 않다. 패키지로 제공되는 경우에는 각 배포판의 패키징 시스템에 맞는 명령을 사용하여 설치하도록 한다.

삼바 패키지의 구성요소

삼바 패키지는 삼바 데몬과 삼바 클라이언트로 이뤄져 있다. 우선 삼바 데몬은 다음과 같다.

```
smbd (SMB 데몬)
nmbd (클라이언트에 대한 NetBIOS 네임서버)
```

위 두개의 파일은 일반적으로 /usr/sbin 디렉터리에 위치하며 시스템 시동 스크립트(/etc/rc.d 아래) 또는 inetd 수퍼 서버에 의해 실행된다. 그 외의 삼바 바이너리 파일들에 대해서 알아보자.

```
smbclinet (유닉스용 SMB 클라이언트)
smbprint (SMB 호스트에 연결된 프린터로 출력하기 위한 스크립트)
smbstatus (현재 로컬 SMB 호스트의 연결 상황 출력)
smbrun (SMB 호스트에서 프로그램을 수행하도록 해주는 스크립트)
```

일반적으로 /usr/bin에 위치하며 그 외에 몇 가지 프로그램이나 스크립트가 더 있는 경우도 있다.

삼바 전체 설정 파일의 위치는 /etc 디렉터리이며 파일명은 smb.conf이다. 사실 여러분이 삼바를 제대로 사용할 수 있는가 없는가는 바로 smb.conf라는 파일 을 어떻게 잘 편집하는가에 달려있다 해도 과언이 아니다.

삼바 설정 파일 smb.conf 들여다보기

리눅스에서 삼바 설정은 /etc/smb.conf라는 파일 하나에서 이뤄진다. 이 파일에 외부 세계와 어떤 자원을 공유할 것인지 그리고 공유하는 자원에 대해서 어떠한 제한을 가할 것인지 정의한다. 세부 사항에 처음부터 집착할 필요는 없다. 중요한 설정을 먼저 마치고 천천히 세부적으로 내용으로 들어가는 자세가 필요하다. 살펴보면 알겠지만 [섹션 이름] 뒤에는 '이름 = 값'의 형식으로 정의가 이뤄진다. 또한 유닉스의 일반적인 규칙과는 설정 파일에서는 대소문자를 가리지 않는다. 그 이유는 아주 분명하다. 마이크로소프트 윈도 머신들이 대소문자를 제대로 가리지 않기 때문이다. 정리를 해보자.

- 정의는 '이름 = 값'의 형식을 지닌다.
- 대, 소문자는 가리지 않는다.
- 세미콜론(;)으로 시작하거나 빈줄은 무시한다.
- 유닉스 관례대로 줄 끝에 역슬래시 문자(\)를 사용하며 다음 줄과 이어진다.
- 대괄호로 묶인 각 섹션은 각각의 서비스를 의미한다.
- 서비스에는 파일 영역 서비스와 프린팅 서비스, 두 가지가 있다.

자, 각 [섹션]의 정의 방법은 어떻게 하는지 차근차근 알아보도록 하자. 우선 파일 영역 서비스에 대해서 알아본다.

```
[foo]
  path = /home/bar
  writable = true
```

위에서는 foo라는 서비스를 정의하고 있다. 경로명은 /home/bar이고 쓰기 가능 으로 서비스하고 있다(writable = true 설정). 다음 프린팅 서비스에 대한 정의이다. 서비스명은 aprinter로 정의되어 있다.

```
[aprinter]
   path = /usr/spool/public
   read only = true
   printable = true
   public = true
```

이 서비스는 읽기만 허용하며 프린팅에 관련된 서비스이다. 오로지 스풀(Spool) 파일에 대한 파일 열기/쓰기/닫기를 통한 쓰기 권한이 부여된다.

삼바 설정 파일은 대괄호로 둘러싸인 몇 가지 섹션이름으로 시작한다. 우선 전체적인 설정에 관계되는 [global] 섹션을 보자.

```
[global]
  printing = bsd
  printcap name = /etc/printcap
  load printers = yes
; Uncomment this if you want a guest account
; guest account = pcguest
  log file = /var/log/samba
  lock directory = /var/lock/samba
  share modes = yes
```

이 섹션에서 정의하고 있는 내용은 서버 전체에 관련된 것이며 이하 다른 섹션에서 명시하지 않은 내용에 대한 기본 설정값을 제공한다. 중복되는 설정이 있다면 바로 [global] 섹션에 적는다.

그 다음은 홈 디렉터리 서비스를 제공하는 것에 관련된 설정을 정의하는 [homes] 섹션이다.

```
[homes]
  comment = Home Directories
  browseable = no
  read only = no
  create mode = 0750
```

클라이언트로 하여금 자신의 홈 디렉터리를 사용할 수 있도록 연결해주는 서비스를 정의한다. 접속 요청이 있을 때 서버는 우선 정의되어 있는 서비스명을 검색한다. 만약 해당 서비스명이 발견되면 물론 그것을 사용한다. 하지만 발견되지 않았을 때에는 서비스명으로 사용자명을 사용하고 알맞은 패스워드를 입력하면 [homes] 섹션에 있는 내용에 의거하여 서비스를 제공한다. 여러 명의 사용자가 있을 때 각 사용자 이름으로 서비스 정의를 할 필요없이 간단히 [homes] 섹션만 하나 정의해두면 편리하게 사용할 수 있도록 하고 있다. create mode는 파일이 생성될 때 기본으로 가지게 될 허가권을 의미한다. 여러분이 원하는 값으로 바꾸는 것이 가능하다. 마지막으로 [printers] 섹션이 있다.

```
[printers]
  comment = All Printers
  browseable = no
  printable = yes
  public = no
  writable = no
  create mode = 0700
```

[homes] 섹션과 같다고 생각하면 된다. 단지 파일 영역 서비스가 아닌 프린팅 서비스에 관련된 내용일 뿐이다. 바로 이 [printers] 섹션이 존재하면 /etc/printcap에 정의되어 있는 프린터를 사용할 수 있게 된다. 주의할 것은 프린팅을 가능케 하려면 꼭 printable = yes라는 설정을 포함하고 있어야 한다는 사실이다.

위 세 가지 섹션이 smb.conf에서 가장 중요하고 특별한 의미를 갖는 섹션이다. 여러분의 배포판에 들어있는 샘플 smb.conf 파일이 아무리 복잡하다 할 지라도 여러분은 우선 이 3 가지 파일에 대해서 충실하게 익히고 난 다음 나머지를 처리하라.

여러분이 만든 smb.conf 설정 파일에 어떤 이상이 없는지 살펴보기 위해서는 testparm 명령을 사용한다.

```
# testparm
Load smb config files from /etc/smb.conf
Processing section "[homes]"
Processing section "[printers]"
No path in service printers - using /tmp
Processing section "[public]"
Loaded services file OK.
Press enter to see a dump of your service definitions
```

<엔터>키를 치면 그 뒤로 각각의 서비스 정의구문에 대한 자세한 결과가 출력 된다.

리눅스 박스에 삼바 서버 띄우기

삼바 패키지가 있는 것을 확인한 후 이제 삼바 서버를 리눅스에 띄워볼 차례이다. 웹 서버와 마찬가지로 삼바 서버를 띄울 수 있는 방법은 두 가지이다.

단독 서버로 띄우기

삼바 서버(smbd)와 netbios 네임서버(nmbd)를 단독 데몬으로 띄우는 방법은 직관적이다. smbd와 nmbd를 -D 옵션을 주고 띄우기만 하면 된다. 대부분의 리눅스 배포판에서는 삼바 패키지를 제공하고 있고 시스템 초기화시에 자동으로 삼바 데몬을 띄우도록 되어있는 경우가 많다. 레드햇 시스템의 경우를 보면 삼바 패키지를 설치하면 /etc/rc.d/init.d 디렉터리에 smb라는 삼바 데몬 시작/ 종료 스크립트가 설치됨을 확인할 수 있다.

일반적으로 시스템 초기화시에 자동으로 실행되지만 수동으로 시작/종료시키고 자 할 때는 다음처럼 한다.

```
# cd /etc/rc.d/init.d
# ./smb stop
# ./smb start
```

시작은 경우에는 스크립트에 start를, 종료의 경우에는 stop을 적어준다.

inetd 수퍼 서버를 통해 간접적으로 띄우기

단독 데몬으로 띄우는 것보다는 반응시간이 약간 길게 느껴질지 모르지만 사실 요즘의 엄청나게 빠른 컴퓨팅 환경에서 그렇게 눈에 띌 만큼은 아닌 듯 하다. inetd 수퍼 서버에게 서버 실행을 일임하는 방식이다. 이렇게 하면 SMB 서비스를 사용하지 않는 경우에 SMB 관련 서버들이 프로세스로 떠 있지 않게 되므로 자원의 절약 효과를 낳는다. SMB 서비스 요청이 있을 때만 inetd에 의하여 자동으로 데몬이 작동한다.

inetd에 서버 데몬을 등록하는 모든 절차가 그러하듯 /etc/services 파일에 서비스 정의 구문을 적어주고 /etc/inetd.conf 파일에 관련 서비스에 해당하는 데몬 실행 명령을 적어준다.

/etc/services 파일에 다음 내용이 있는지 확인하고 없으면 맨 뒤에 추가한다.

```
netbios-ns   137/tcp     nbns
netbios-ns   137/udp     nbns
netbios-dgm   138/tcp     nbdgm
netbios-dgm   138/udp     nbdgm
netbios-ssn   139/tcp     nbssn
```

/etc/inetd.conf 파일에 다음 내용이 있는지 확인하고 없으면 맨 뒤에 추가한다.

```
netbios-ssn stream tcp nowait root /usr/sbin/smbd smbd
netbios-ns dgram udp wait root /usr/sbin/nmbd nmbd
```

주의할 사항으로 1. 단독 서버 실행 방법과 2. inetd 수퍼 서버에 일임하기 방법을 동시에 사용할 수는 없다는 점을 들고 싶다. 레드햇 배포판 사용자라면 X 윈도에서 제어판을 열고 실행 레벨 편집기를 연 다음 실행 레벨 3(일반적인 실행레벨)에서 SMB 관련 실행 스크립트를 삭제한다.

실행 레벨 3 번의 시작(Start) 창에서 smb를 선택한 후 제거(Remove) 버튼을 누른다. 자주 SMB 서비스를 사용하지 않는 사람들에게 권하고 싶은 방법이다.

윈도 머신에게 리눅스 드라이브 보여주기

그냥 모든 사람들에게 리눅스 디렉터리 구조 중 일부를 공유시켜주기 위해서는 다음과 같은 설정을 smb.conf에 설정한다.

```
[public]
   comment = Public Stuff
   path = /home/public
   public = yes
   writable = yes
   printable = yes
```

정상적으로 smbd와 nmbd를 -D 옵션과 함께 실행하여 데몬 상태로 뜨게 해놓 고 나서 네트워크로 연결된 클라이언트 윈도 95나 윈도 NT 쪽으로 가서 네트 워크 드라이브 연결을 하면 된다. 드라이브 명을 적어줄 때는 삼바가 설치된 유닉스 호스트의 호스트명과 서비스명을 다음과 같이 적어준다.

```
\\freeyong\yong
```

도메인 명을 쓰지 않는다는 사실에 유의하기 바란다. 서비스명은 삼바 서버의 /etc/smb.conf에 적어준 이름 또는 간단히 자신의 유닉스 호스트 계정명을 적 어주면 된다. 역슬래시를 두 번 적은 다음 호스트명 그리고 역슬래시 한 번, 맨 마지막에 서비스명을 적는다. 예를 들어 공용 디렉터리로 지정한 [public] 서비스 섹션을 접근하기 위해서는 다음과 같이 적는다.

```
\\freeyong\public
```

리눅스에서 삼바 서버를 설치하고 나서 과연 삼바 서버가 제대로 작동하는지는 윈도 95와 윈도 NT 머신에서 연결하지 않아도 직접 확인 가능하다. 왜냐하면 리눅스 삼바 패키지에 서버와 클라이언트가 모두 들어있기 때문이다. 삼바 서버를 실행한 후 서버가 설치된 리눅스에서 다음과 같이 해보자. yong이라는 일반 사용자가 다음과 같이 했다고 치자.

```
$ smbclient \\\\freeyong\\yong
```

또는

```
$ smbclient '\\localhost\yong'
```

암호가 필요한 경우에는 암호를 물어올 것이다. 암호가 없는 경우에는 그냥 엔터키를 치면 된다.

주의해야 할 사항을 한 가지 지적하자. \\\\freeyong\\yong이라고 적은 것은 뭔가 이상하지 않은가? 혹자 오자가 아닌가? 그렇지 않다. 유닉스 셸에서 역슬래쉬 문자는 독특한 의미를 갖기 때문에 어떤 프로그램에게 역슬래시 문자 두 개를 전달하고자 할 때는 셸에서 위와같이 네 번을 적어 주어야 한다. C 프로그래밍 printf 문에서 역슬래시 문자를 출력하기 위해서는 어떻게 적어주어야 하는지 아는 사람은 금방 이해할 것이다. 또는 셸의 문자 해석을 강제로 막는 작은 따옴표로 묶어주면 된다. 이 기회에 책을 찾아서 역따옴표(backquoute, ‘), 작은 따옴표(’), 큰 따옴표(”)가 셸에서 어떤 다른 의미를 갖는지 조사해보기 바란다. 중급자라면 이것에 대해서 충분히 숙지하고 있어야 한다.

smb: \> 프롬프트가 나오면 h라고 쳐서 어떤 명령이 있는지 확인해보기 바란다. ftp에서 사용하는 명령과 거의 흡사하다. 끝낼 때는 q를 누른다. 이제 윈도 머신과 리눅스 머신이 연결되어 있는 경우 디스켓을 들고 방 안에서 돌아다니거나 또는 리눅스 머신으로 ftp해 들어가서 파일을 가져온다든지 하는 일은 없을 것이다. 물론 최후의 수단은 ftp일 수 있다. 하지만 문제는 리눅스가 서버를 하고 리눅스의 드라이브를 공유하는 것이라면 모르지만 리눅스처럼 배포판에 자연스럽게 ftp 서버가 제공되지 않는 윈도 95나 윈도 NT 머신의 경우에는 반대의 경우 즉 리눅스 머신에서 윈도 머신의 파일을 가져오는 일이 여간 귀찮은 것이 아니다. 그렇다고 쓸 데 없이 윈도 95/NT용 텔넷 서버나 FTP 서버를 구입한다는 것은 정말 불필요한 일이지 않은가?

리눅스 박스에게 윈도 드라이브 보여주기

이번에는 반대의 경우를 생각해보자. 리눅스 박스가 클라이언트의 입장이고 윈도 머신의 드라이브에 있는 파일을 가져오고자 한다. 클라이언트 프로그램으로는 위에서도 사용한 바 있는 smbclient가 있다.

```
$ smbclient -L freeyong
```

-L 옵션 다음에 윈도 머신의 호스트명(정확히는 랜 매니저 명)을 적어주면 그 쪽 서버에서 여러분에게 제공하는 서비스 명을 모두 보여줄 것이다.

```
[yong@freeyong yong]$ smbclient -L freeyong
Added interface ip=127.0.0.1 bcast=127.255.255.255 nmask=255.0.0.0
Server time is Fri Apr 18 05:59:01 1997
Timezone is UTC+9.0
Domain=[WORKGROUP] OS=[Unix] Server=[Samba 1.9.16p9]

Server=[freeyong]     User=[yong]     Workgroup=[WORKGROUP]
Domain=[WORKGROUP]

    Sharename   Type      Comment
    ---------   ----      -------
    IPC$        IPC       IPC Service (Samba 1.9.16p9)
    lp          Printer
    public      Disk      Public Stuff
    tmp         Disk      Temporary file space
    yong        Disk      Home Directories
```

```
This machine has a browse list:

    Server          Comment
    ---------       -------
    FREEYONG        Samba 1.9.16p9

This machine has a workgroup list:

    Workgroup       Master
    ---------       -------
    WORKGROUP       FREEYONG
```

위 출력 결과는 편의상 리눅스에서 삼바 서버를 운영하고 삼바 클라이언트로 확인한 것이다. 자 위에서 서비스명을 확인했으며 이제 서비스를 사용해보자.

```
$ smbclient \\\\서버호스트명\\서비스명 암호
```

마치 FTP 세션 들어간 것과 같은 유사한 명령행 인터페이스가 나올 것이다.

키타 사항

커널 설정에서 파일시스템(FileSystem)-SMB 파일 시스템 지원을 커널 안에 내장하든가 아니면 모듈로 컴파일한다. 윈도 95를 사용하는 경우에는 SMB 윈도 95 버그 패치 또한 설정하도록 하자. 정확히 말해 이것은 리눅스를 클라이언트로 사용할 때 사용하는 것이다. 삼바 서버를 설치하여 리눅스 박스를 서버로만 사용하고자 하는 사람들에게는 필요하지 않은 내용이다.

윈도 95 서버의 경우 디렉터리 목록을 보여주는데 버그가 있다는 사실이 밝혀졌다. 약간의 속도 저하를 감수하고라도 안정적인 디렉터리 서비스를 받고자 한다면 SMB 윈도 95 버그 패치 옵션을 켜두도록 하자. 커널에서 지원하는 SMB 클라이언트 지원 서비스를 받고자 할 때는 특별한 마운트 프로그램을 가져와야 한다. 위치는 `sunsite.unc.edu:/pub/Linux/system/filesystems/smbfs`으로 smbfs-2.0.1.tgz를 가져와서 컴파일하여 사용한다. 커널 버전은 2.0.29 이상을 사용하도록 하라. make , make install 후에는 smbmount, smbumount라는 특별한 마운트 프로그램이 설치된다.

위에서 계속 예로 보여준 것과 같이 리눅스 자체에 삼바 서버를 띄우고 리눅스 안에서 클라이언트를 돌려서 확인해 보는 것도 좋다. 그래야만 윈도 95/NT로 연결할 때 문제가 생기는 경우 과연 서버의 문제인지 아니면 케이블의 문제인지 윈도 95/NT 설정의 문제인지 쉽게 문제를 파악할 수 있기 때문이다.

2.4 네트워크 기본 보안 설정

인터넷은 상호 신뢰를 기반으로 형성된 전 지구적 네트워크이다. 수많은 거대 BBS 회사들, 특히 미국의 컴푸서브(CompuServe) 또는 AOL과 같은 회사들이 있었다고는 하나 그들의 영향력은 기껏해야 전 지구의 일부인 미국 내에 머물렀을 뿐이다. 인터넷은 처음부터 비영리적인 목적으로 관리자들의 상호 협조 아래 수평적인 관계를 유지해 가면서 급속도로 전파되어 나갔고 특히 여러분도 알다시피 월드 와이드 웹이라는 매체를 통해 일반인에게 다가온 상태이다.

인터넷을 접하는 사람들의 숫자가 증가할 수록 인터넷을 통해 혜택을 누리는 사람들이 많아지는 반면 역시 무시할 수 없을 만큼의 일부 사람들이 피해를 당하고 있다. 여러분이 자주 신문, TV 뉴스에서 읽을 수 있는 크래킹(Cracking) 사건도 있지만 그런 굵직한 사건보다 몇 배로 자그마한 개인들의 피해사례가 발생하고 있는 것은 쉽게 예상할 수 있다.

장난꾸러기들과 범죄자들의 심리가 어떠한지는 모르겠으나 분명한 사실은 그들에게 설교집이나 철학책을 가져준다고 해서 마음을 고쳐먹는다는 보장도 없으며 선도 활동과는 별도로 여러분 자신을 스스로 방어해야 하는 일은 어떠한 경우에도 소홀히 할 수 없을 것이다.

경찰이 있다고 해서 강도와 살인이 없는 것은 아니다. 감시기계가 있다고 해서 도둑이 들지 않는 것 은 아니다. 하지만 이런 기본적인 방책마저 세우지 않는다면 범죄 행위가 너무 쉬워지지 않겠는가?

전문적인 주제로 들어가기에 앞서 이런 이야기를 하는 이유는 ‘네트워크 보안” 이라는 주제에 대하여 이성적인 자세를 갖자는 의도이다. 보안에 대한 양극단의 생각, 즉 보안에 소홀히 하는 방만한 태도 그리고 보안에 대하여 극도로 신경질적으로 대응하는 태도 모두 개인과 단체의 이익에 득이 되지 않는다. 보안이 필요 없는 곳에서 보안에 대해 불필요한 신경을 쓴다는 것은 우스운 일이며 보안이 필요한 곳에서 관리자가 아무 생각 없이 졸고만 있는 것은 위험천만한 일이다.

2.4.1 리눅스 기본 보안 설정

경찰 업무는 매우 힘든 일이다. 고도의 훈련이 필요하며 마음가짐 또한 중요하다는 사실을 잘 알고 있을 것이다. 네트워크 보안 문제 또한 마찬가지이다. 보안 기술 수준은 서로 많은 차이를 보이며 무엇보다도 강조하고 싶은 것은 ‘배우지 않으면 안된다’는 사실이다. 많은 사람들이 서버 운영체계에 대하여 보안 관리마저도 편리했으면 하는 바램을 갖는데 보안의 문제는 애초부터 편리함의 기준으로 해결할 수 있는 문제가 아니라는 사실을 분명히 하겠다. 다른 문제는 모르겠으나 보안 문제에 관하여 편리한 도구를 찾는 네트워크 관리자가 있다면 기업이나 단체의 경우 당장 해고해야 한다고 말하고 싶다. 그는 무술이나 사격술을 배우지 않겠다는 경찰관과 같다. 내근 경관이 아니라면 어찌 이런 경찰관을 믿을 수 있겠는가?

네트워크 보안 문제에 있어 일을 어렵게 만드는 일이 있다면 그 특성 상 네트워크 보안은 항상 ‘방어’라는 사실에 있다. ‘범죄와의 전쟁 선포’같은 일은 없다. 공격이 최선의 방어라는 멋진 말도 통하지 않는다. 공격해 왔다고 해서 이 쪽에서 반격할 수 있는 것도 아니다. 우리는 한 대도 때리지 못하고 막기만 해야 한다는 것이다. 이렇게 생각해보면 조금은 억울한 일임에 틀림없다. 피해를 당하고 나서 범인을 잡고 처벌할 수도 있겠지만 이미 일은 벌어지고 난 다음이다. 기업의 경우 자사의 기밀이 노출되어 이미 회사가 위태로운 지경에 빠지고 나서 무일푼의 젊은이를 구속한들 무슨 소용 있겠는가?

보안 문제에 대한 이야기를 할 때마다 상호 신뢰를 기반으로 형성된 인터넷을 악용하는 사람들이 있다는 엄연한 사실에 화가 날 때가 많다. 이 글을 읽고 있 는 사람이라면 제발 남에게 피해주는 일은 생각하지 말았으면 좋겠다. 세상에 는 남에게 피해를 주는 일 이외에도 재미있는 일들이 얼마든지 있다고 생각한다.

기본 규칙

필요 없는 서버는 실행되지 못하도록 원천 봉쇄한다!

여기서 설명하는 바는 유닉스/리눅스에 공통되는 사항이다. 네트워크 서버들은 크게 2 가지 방식으로 실행되는데 하나는 inetd 수퍼 서버에 의해 자동 관리되는 방식이며 나머지 하나는 유닉스의 여타 프로세스와 마찬가지로 직접 실행시키는 방식이다. 후자의 경우에는 부팅 과정을 면밀히 살펴보고 필요 없는 서버들이 작동하지 않도록 시스템 초기화 파일을 수정하면 된다. 보안 문제에 관하여 극도로 예민한 사람은 간단히 그 서버를 시스템에서 삭제하면 그만이다. 리눅스의 경우 그리고 레드햇, 데비안 등의 배포판에서는 서버 설치/삭제가 자유로우므로 괜한 하드디스크 공간을 차지하게 할 필요 없이 삭제하는 것이 좋다고 본다. 원할 때는 언제든지 시스템 운영 도중 설치하고 그 즉시 가동 가능하기 때문이다. 우리가 놓치기 쉬운 부분이 바로 inetd 수퍼 서버에 의해 관리되는 서버들이다. 우리가 모르는 사이에 보안에 취약한 어떤 서버들이 마구 실행될 수 있다.

/etc/inetd.conf 청소하기

각 행은 하나의 서비스를 정의한다. 형식은 다음과 같다.

```
<서비스명> <소켓 타입> <프로토콜> <플래그> <사용자> <서버 경로> <전달 인수>

#
# These are standard services.
#
ftp  stream tcp nowait root /usr/sbin/tcpd in.ftpd -l -a
telnet stream tcp nowait root /usr/sbin/tcpd in.telnetd
gopher stream tcp nowait root /usr/sbin/tcpd gn
```

ftp, telnet 등의 서비스는 아주 기본적인 서비스이다. 하지만 마지막 줄에 있는 고퍼(gohper) 서비스는 웹의 등장으로 인해 요즘 별 인기도 없어 대부분 사용하지 않을 것이라고 본다. 잘 모른다면 여러분은 이

것을 사용하고 있지 않는 것이며 서비스를 제공할 능력도 되지 않으므로 줄 맨 앞에 샵(#) 문자를 넣어 주석 처리해버리기 바란다.

```
#
# Shell, login, exec and talk are BSD protocols.
#
#shell  stream tcp nowait root /usr/sbin/tcpd in.rshd
#login  stream tcp nowait root /usr/sbin/tcpd in.rlogind
#exec  stream tcp nowait root /usr/sbin/tcpd in.rexecd
talk   dgram  udp wait  root /usr/sbin/tcpd in.talkd
```

위에서 나열한 서비스들은 아주 끔찍한(?) 서비스들이다. 패스워드 없이 원격 로그인하거나 원격에서 서버 상에 프로그램을 실행시킬 수 있는 기능을 제공하기 때문이다. 이런 서비스에 의해 공격당하면 범인도 못 잡는다. shell, login, exec 등은 개인용 서버가 아닌 이상 위에서처럼 주석처리하기 바란다.

```
#
# Finger, systat and netstat give out user information
# which may be valuable to potential "system crackers."
# Many sites choose to disable
# some or all of these services to improve security.
#
# cfinger is for GNU finger, which is currently not
# in use in RHS Linux
#
# finger stream tcp nowait root  /usr/sbin/tcpd in.fingerd
#cfinger stream tcp nowait root  /usr/sbin/tcpd in.cfingerd
#systat stream  tcp nowait guest  /usr/sbin/tcpd /bin/ps -auwwx
#netstat stream tcp nowait guest  /usr/sbin/tcpd /bin/netstat
```

이미 inetd.conf 설정 파일에 잘 적어놓고 있는 내용이다. 여기에 있는 서비스들은 외부에게 여러분 서버, 사용자에 관한 정보를 누출한다. finger, systat, netstat은 주석 처리할 것인가 말 것인가? finger의 예를 들어보자.

```
[yong@freeyong src3]$ finger @localhost
[localhost]
Login Name     Tty Idle Login Time  Office Office Phone
root root     *4  1:18 May  12 12:20
yong Lee Man Yong *1  1:18 May  12 12:20
yong Lee Man Yong *2  1:18 May  12 12:20

[yong@freeyong src3]$ finger @linux.cs.helsinki.fi
[melkki.cs.Helsinki.FI]
----------------------------------------------------  University
of Helsinki - Department of Computer Science
  Local time is Mon May 12 07:45:27 EET DST 1997

  Latest Linux Kernel versions (from ftp.cs.Helsinki.FI)
    v2.0 (stable):   2.0.30
    v2.1 (development): 2.1.36
-------------------------------------------------------
Login Name    Tty Idle Login Time Office Office Phone
ahalme Allan Halme p8 7:22 May 12 00:22 [ hi-hat.in.helsin ]
```

위에서 보다시피 도대체 어떤 사람들이 로컬 호스트에서 작업하고 있는지 아주 분명하게 정보를 누출하고 있다. 다음은 헬싱키 대학 서버에서 어떤 사람이 작업하고 있는지 보여주는 예이다. 리눅스 배포판들은 대부분 finger를 할 수 있도록 해두었는데 이는 보안에 관심 없기 때문이 아니다. 리눅스는 상호 신뢰

에 의해 개발되는 운영체계이기 때문이다. 하지만 중요한 임무를 수행하고 침입자를 막아야 하는 공개 서버들은 이를 막아야 한다. finger를 막은 호스트인 경우에는 다음과 같은 메시지가 나온다.

```
[yong@freeyong src3]$ finger @localhost
[localhost]
finger: connect: Connection refused
```

어떻게 보면 finger와 같이 즐거운 정보를 제공하는 서비스를 몇몇 악의를 가진 인간들 때문에 쓰지 못한다는 것은 불행한 일이 아닐 수 없다. 일단 이 정도만 봉쇄해도 여러분의 시스템을 가지고 놀고 싶어하는 사람들에게 불쾌감을 안겨줄 것이다. 또는 도전의욕을 더욱 부추키는 것일까?

/etc/ftpusers

이 파일은 이름이 의미하는 바와는 달리 이 파일 안에 나열한 사용자들은 ftp 서비스를 사용할 수 없도록 하는데 그 목적이 있다.

```
root
bin
daemon
adm
lp
sync
shutdown
halt
mail
news
uucp
operator
games
nobody
```

파일 내용을 살펴보면 잘 알겠지만 나열된 사용자들은 일반적인 사용자들이 아니고 시스템 운영에 필요한 특수한 목적의 사용자 ID들이다. 이들은 ftp 로그인 말고도 telnet 같은 것으로도 절대 들어와서는 안되는 사용자들이다. 주목할 만한 것으로는 맨 첫 줄의 root를 들 수 있는데 root 사용자로 ftp하는 일은 극도로 위험한 일이기 때문이다. root가 파일 전송 프로토콜인 ftp로 들어와서 중요한 파일을 몽땅 가져간다거나 중요 파일을 지운다든지 또는 중요한 서버를 트로이 목마와 교체한다든지 할 때는 어떻게 하겠는가?

/etc/securetty

일반적으로 root는 자신이 관리하는 서버 콘솔 앞에 앉아서 작업해야 한다. root가 전세계 어디에서든 접속할 수 있도록 해둔다면 이는 크래커들에게 여러분의 시스템을 제물로 갖다바치는 일과 다를 바 없다.

```
tty1
tty2
tty3
tty4
tty5
tty6
tty7
tty8
```

기본 설정치는 리눅스 가상 콘솔 8 개일 것이다. 극도로 예민한 사람들은 tty1 하나만 남겨두기도 한다. 지난 9 월 배포되었던 알짜 슬랙웨어 3.1은 바로 이 파일에 커다란 문제를 갖고 있다. 콘솔 뿐 아니라 외부에서 텔넷 로그인할 수 있도록 잘못 설정되어 있기 때문이다. 만약 여러분이 알짜 슬랙웨어 3.1 사용자라면 이 파일을 지금 즉시 점검해보기 바란다. 그 외의 버전에는 이런 문제가 없다. 많은 사람들이 외부에서 root로 로그인할 수 없다면 잘못된 것 아니냐고 호소하는 사람들이 종종 있다. 잘못된 것이 아니고

지극히 정상적인 설정이다. 굳이 원격지에서 접속하여 root 작업을 하고 싶다면 일반 계정으로 로그인한 후 su 명령을 사용하라.

2.4.2 tcpd 호스트 접근 제한 메커니즘

위에서 본 inetd.conf 설정 파일을 한 번 자세히 살펴보면 다음과 같은 줄을 발 견할 수 있을 것이다.

```
telnet stream tcp nowait root /usr/sbin/tcpd in.telnetd
```

텔넷 서비스에 대한 실제 서버 프로그램은 in.telnetd이다. 그런데 설정 파일 형식의 서버 자리에는 in.telnetd가 있는 것이 아니라 /usr/sbin/tcpd라는 프로그램이 놓여 있다. ftp나 다른 설정을 보아도 tcpd가 자리를 차지하고 있다는 것을 확인할 수 있다.

바로 tcpd는 그 뒤에 적은 서버에 대하여 로그인, 접근 제한 메커니즘을 제공하는 프로그램이다. inetd에 의해 실행될 때마다 tcpd는 2 개의 설정 파일을 읽은 다음 그 안에 정의되어 있는 규칙에 의거하여 서비스에 대한 접근 허락, 거부 여부를 결정한다.

설정 파일을 읽는 순서는 /etc/hosts.allow 다음 /etc/ho sts.deny 이다. 여러분이 명심해야 할 사항은 tcpd는 설정 파일을 읽으면서 맨 처음 부합되는 규칙을 만나면 거기서 멈춘다는 사실이다. 어떤 것을 주의해야 하는지에 대해서는 각 설정 파일에 대한 설명을 할 때 언급하기로 하겠다. 만약 부합되는 규칙을 발견하지 못하면 허락으로 간주한다.

/etc/hosts.allow

이 파일은 허용 규칙을 정의한다. 설정 형식은 매우 간단하다.

```
<서비스 목록>: <호스트 목록> [: 명령]
```

<서비스 목록>은 규칙을 적용할 서비스의 목록으로서 각각의 항목은 콤마(,)로 구분하여 나열한다. 서버명으로는 ftpd, telnetd, fingerd 등을 들 수 있다. <호스트 목록>은 호스트명을 나열한 것인데 역시 콤마로 구분한다. 호스트명 이외에도 IP 주소를 사용할 수 있다. 또한 호스트명과 IP 주소에 대하여 와일드카드 문자를 사용하는 것이 가능하다.

예를 들어 gw.vk2ktj.ampr.org와 같이 하나의 호스트를 지정하는 방법도 있고 .uts.edu.au와 같이 호스트명이 저렇게 끝나는 모든 호스트에 대하여 일괄 지정하는 것도 가능하다. 만약 44.이라고 표현하면 이 숫자로 시작하는 모든 IP주소를 가진 호스트를 가리키는 것이다.

이외에는 특별한 심볼이 있는데 ALL은 글자 그대로 인터넷 상의 모든 호스트를 의미하며 LOCAL은 도트(.) 문자를 포함하지 않는 모든 호스트명을 가리키는데 즉 여러분의 호스트 머신과 같은 도메인에 속한 모든 호스트들을 가리킨다. PARANOID는 많이 사용되지 않지만 주소 위조자들을 경계하기 위하여 주소와 호스트명이 일치하지 않는 호스트를 가리킬 때 사용한다. 마지막으로 EXCEPT는 영어 뜻 그대로 전체 호스트에서 어떤 호스트를 제외할 때 사용되는 키워드이다.

<명령> 부분은 일반적으로 적어주지 않는데 적어준다면 tcpd에 의해 그 규칙이 적용될 때마다 주어진 명령을 수행하도록 하도록 되어 있다. 명령은 완전한 패스명으로 적어주어야 한다. 보통 누가 호스트에 접속하려고 시도하는지 확인하는 명령, 특정인 접속할 때 시스템 관리자에게 자동으로 메일이나 경고 메시지를 보내도록 하는 명령을 적는다.

```
# /etc/hosts.allow
#
# 메일을 모든 사람에게 허용
in.smtpd: ALL
# telnet과 ftp는 같은 도메인에 속한 사람과 집에 있는 본인의 호스트
# 에만 허용
telnetd, ftpd: LOCAL, myhost.athome.org.au
# finger를 허용하되 누가 요청했는지 기록을 남긴다.
fingerd: ALL: (finger @%h | mail -s "finger from %h" root)
```

/etc/hosts.deny

이 파일은 서비스 거부에 대한 파일이다. 일반적으로 ALL: ALL이라는 줄 하나를 넣어서 /etc/hosts.allow에서 허가되지 않는 모든 것에 대하여 기본적으로 거부 정책을 사용하는 것이 좋다고 본다. /etc/hosts.deny에서 기본 정책으로 모든 서비스에 대하여 그리고 모든 사용자에 대하여 거부를 해 놓는 이유는 tcpd가 /etc/hosts.allow와 /etc/hosts .deny에서 어떠한 알맞은 규칙도 발견하지 못할 때는 허가한다는 정책을 가지고 있기 때문이다. 기본적인 거부 정책 다음에 /etc/hosts.allow에 하나씩 허가하는 것이 올바른 절차라고 본다.

주의할 것은 절대 /etc/hosts.allow에 ALL: ALL과 같은 무지한 설정은 하지 말아달라는 것이다. 그리고 tcpd의 규칙 점검 특성 상 특수한 규칙을 앞에 두고 일반적인 규칙을 뒤에 둔다. 순서가 중요하기 때문이다. 여러분이 tcpd 설정만 잘해두어도 훌륭한 관리자라는 평가를 받을 수 있으며 크래커 입장에서는 아주 신경 쓰이는 서버 축에 들것이다. 인터넷 상에 수많은 서버 중 뚫기 힘든 여러분의 서버 주위에서 더 이상 배회하려 들지 않을 것이다.

2.4.3 고질적인 센드메일 문제, RPC 문제

크래커들의 네트워크 지식 수준이 높다는 생각을 버려야 한다. 그들은 사실 여러분보다 더 아는 것도 별로 없다. 대부분의 크래커들은 CERT와 같이 보안 문제에 대한 정기적인 발표 사이트에 가서 이미 알려진 보안 문제를 알아보고 게으른 관리자를 두고 있는 서버들에게 치근덕거린다. 이미 보안 상의 버그가 있다고 발표된 버전에 대해서는 즉시 새로운 센드메일 버전을 사용하든지 아니면 데비안 배포판처럼 smail이라는 간단한 메일 서버를 사용하는 것도 좋다.

NFS와 Sun RPC 서비스를 제공하는 경우에도 각별한 주의를 기울여야 한다. 이 서비스를 악용할 수 있을만한 소지가 많기 때문이다. NFS의 경우에는 마운트 허용 대상에 대한 주의 깊은 설정이 필요하다. 읽기만 허용해야 할 사람에게 읽기/쓰기를 허용한다든지 하는 일은 절대 있어선 안된다.

2.5 리눅스, 하나의 완벽한 방화벽 소프트웨어

리눅스 인터넷/인트라넷 서버 활용 분야에 있어 리눅스 커널과 간단한 관리 도구 하나만으로도 아주 훌륭한 방화벽 기능을 해낼 수 있다는 사실에 많은 사람들이 놀라워하고 있다. 아직은 실험적인 수준의 코드라고 하지만 리눅스에서 실험적이지 않은 것이 얼마나 있었던가? 그리고 리눅스에서 말하는 실험적 코드는 여러분이 이미 알고 있는 상용 소프트웨어의 베타 버전과는 성격이 다르다. 또한 실험적 코드라 할지라도 여러분의 실제 평가에 따라 쓸만한 것으로 판명되면 좋은 일이고 만약 그렇지 못하다면 채택하지 않으면 그만이다.

2.5.1 방화벽(Firewall)이란 무엇인가?

방화벽이라 함은 자동차 분야의 용어였다고 한다. 방화벽은 자동차 엔진과 승객 사이의 차폐막 역할을 해주는 장치를 가리킨다. 자동차 엔진에 불이 붙는 상황이 벌어져도 화재로부터 승객을 보호하기 위해서 고안되었다. 이 용어가 그대로 네트워크 분야로 와서 컴퓨터 네트워크 관련의 의미를 가지게 되었는데 여기서의 방화벽이란 사적인 네트워크를 일반적으로 인터넷과 같은 공공의 네트워크로부터 보호하는 장치를 통틀어 말한다.

앞으로 방화벽 역할을 해내는 컴퓨터를 그냥 간단히 방화벽이라고 부른다. 이 방화벽은 여러분이 보호하고자 하는 인트라넷과 같은 사적인 네트워크와 인터넷 둘 다 접근할 수 있는 특별한 위치에 놓인 컴퓨터이다. 간단히 말해 인트라넷은 인터넷을 접근할 수 없으며 또한 인터넷은 여러분의 인터넷을 접근할 수 없도록 하는 것이다. 이렇게 인트라넷과 인터넷의 연결 부분에서 방화벽이 차단하고 있으므로 만약 인트라넷에 있는 어떤 사람이 인터넷을 사용하고자 한다면 우선은 방화벽에 텔넷으로 로그인한 후 그 곳에서 인터넷을 사용해야 한다. 보통 가장 간단한 형태의 방화벽은 이중 네트워크로서 서로 다른 네트워크 두 개에 대하여 각각 연결되어 있다. 이서네트 카드를 2개 달고 있거나 하나는 이서네트, 하나는 전화선이나 전용선을 통한 PPP 접속인 경우가 많다.

2.5.2 준비 사항

- 커널 컴파일을 해야 하므로 커널 소스가 필수적이다. 각 배포판에는 커널 소스가 기본적으로 포함되어 있다(커널 버전 2.0.0 이상).
- ipfwadm 이라는 네트워크 관리 도구가 있어야 한다. 최신 배포판에는 모두 포함되어 있다. 만약 여러분의 배포판에서 찾을 수 없다면 `ftp.xos.nl/pub/linux/ipfwadm`에 가서 ipfwadm-2.3.0. tar.gz 파일을 받아 와서 간단히 make 그리고 make install 과정만으로 쉽게 설치할 수 있을 것이다.

2.5.3 리눅스 네트워킹 코드로 할 수 있는 일은?

우선 IP 방화벽을 구축할 수 있다. IP 방화벽이라 함은 여러분이 정하는 규칙 (어떤 호스트 또는 네트워크로부터 오는 또는 가는 패킷을 차단할 것인가? 어 떤 서비스에 관한 패킷을 차단할 것인가?)에 따라 패킷을 허가하거나 거부할 수 있는 기능이다. 이렇게 아예 패킷 수준에서부터 차단함으로써 기존의 각 서 버 소프트웨어 자체의 보안 기능보다 훨씬 높은 수준의 보안 대책이 마련될 수 있게 되었다. 이미 커다란 업체들에서는 모두 방화벽 하드웨어나 소프트웨어를 사용하고 있다.

리눅스는 이 방화벽 기능을 커널 네트워킹 수준에서 처리하고 있다. 이는 여러분이 고가의 라우터 장비나 방화벽 전문 소프트웨어를 구입하지 않고 단순히 리눅스를 설치하는 것 하나만으로도 게이트웨이, 라우터 역할의 해낼 수 있게 되었다는 것을 의미한다. 돈 많은 회사들은 일단 제치고라도 학교나 비영리 사회단체들은 충분히 리눅스의 이 기능으로부터 혜택을 받을 준비가 되어 있다고 할 수 있다.

상황은 일반적으로 다음과 같다. 하나의 이서네트 또는 기타 방식의 인트라넷이 구축되어 있는 상태이다. 그리고 그 인트라넷과 인터넷의 접촉 지점에 리눅스를 탑재한 게이트웨이 머신이 놓여 있다. 인트라넷이 인터넷과 연결하면서 제일 중요한 면이 바로 인터넷의 자원을 풍부히 활용하면서도 인트라넷의 정보가 외부로 유출되지 않고 인터넷의 잠재적인 공격에 대한 불안을 해소하는 것 이다. 바로 리눅스가 인터넷과 인트라넷의 중간에 놓여 패킷 필터링(Filtering) 작업을 해냄으로써 문제를 해결하게 될 것이다.

그 다음 할 수 있는 일로는 IP 교통량에 대한 분석(IP Acco unting) 그리고 하나의 공식적인 인터넷 IP를 인트라넷의 여러 컴퓨터들이 공유하여 사용할 수 있도록 하는 IP 매스커레딩(IP Masquerading) 기능을 들 수 있다. IP 분석 기능을 같이 활용하면 관리자의 입장에서 얘기치 않은 방화벽 보안 구멍을 발견하는 것이 가능하다.

2.5.4 IP 방화벽(IP Firewall)

우선 커널 컴파일을 해야 할 필요가 있을지 모른다.

```
Networking options --->
[*] Network firewalls
....
[*] IP: forwarding/gatewaying
....
[*] IP: firewalling
[ ] IP: firewall packet logging
```

위에서 나타낸 옵션이 커널 기능에 들어 있어야만 가능하다. 만약 앞으로 수행하는 명령이 제대로 실행되지 않을 때는 커널 컴파일 과정을 재차 확인해보기 바란다. 노파심이지만 정말로 많은 사람들이 커널을 컴파일해두고 제 위치에 커널을 복사하고 lilo를 다시 실행시키지 않는 경우가 허다하다. 레드햇 배포판의 경우 lilo.conf가 가리키고 있는 커널의 위치는 / 디렉터리가 아니라 /boot 디렉터리이다. 커널 컴파일 과정중 make zlilo는 커널을 /boot 위치가 아닌 / 위치에 복사하고 lilo를 실행시키므로 각별히 주의하라.

IP 패킷 필터링 방화벽 방화벽을 구축하는데는 크게 2 가지 방법이 있다고 한다. 하나는 지금부터 설명하고자 하는 패킷 필터링 방화벽(IP Packet Filtering Firewall)이고 나머지 하나는 프록시(Proxy, 대리인) 서버 방법이 있다. 패킷 필터링이라는 말이 의미하듯 이런 형태의 방화벽은 패킷을 여러분이 정하는 특정 규칙에 따라

걸러내는 방식이다. 어떤 패킷은 그냥 거부해버릴 수 있고 어떤 패킷에 대해서는 허가할 수 있다. 여기서 중요한 것은 바로 그 어떤 패킷이라는 기준이 어떻게 성립되는가하는 것이다. 패킷 내부에는 패킷을 보낸 발신지의 주소, 패킷이 도착하고자 하는 목적지 주소, 패킷이 도착하고자 하는 목적지의 포트(Port) 그리고 TCP/UDP 등의 패킷 형태 등에 대한 정보가 기입되어 있다. IP 패킷 필터링 방화벽은 바로 패킷의 내부를 조사한다.

IP 패킷 필터링 규칙

발신지/목적지 주소

그 패킷이 어디로부터 오는지 그리고 어느 곳으로 가려 하는지 점검하여 결정한다. 몇몇 지역에 지사를 두고 있는 회사가 있다고 하자. 각 지역에는 인트라넷이 구축되어 있고 그 인트라넷은 인터넷을 통해 연결되어 있다. 각 인트라넷은 인터넷의 연결 지점에 리눅스 방화벽을 설치하고 각각의 인트라넷 주소만 허용하도록 설정하고 나머지 주소에서 오는 패킷은 거부해버린다.

프로토콜의 종류

TCP, UDP, IGMP

IP 옵션

IP 패킷에 몇 가지 사적인 네트워크에게는 위험한 옵션들이 있다. 그 중 하나는 는 소스 라우트(source route) 옵션인데 이 패킷이 여러분의 네트워크에 돌아다니면 외부에서 여러분의 네트워크가 어떻게 구축되어 있는지 파악할 수 있기 때문이다. 일반적으로 이런 패킷은 거부해야 한다. 외부에서 여러분의 네트워크 구성을 알 필요가 없지 않은가?

발신지/목적지 포트 번호

TCP/UDP 헤더 부분에는 포트 번호가 기입되어 있다. TCP/IP 네트워크에서는 바로 포트 번호를 가지고 서비스를 구분한다. 어떤 호스트에 도착한 같은 패킷이라 할 지라도 23번 포트를 향해 달려온 패킷은 텔넷 서비스에 관계된 패킷이며 25번 포트는 메일 서비스에 관련된 것이다. 여러분이 잘 알고 있는 웹 서비스는 일반적으로 80번 포트를 사용한다. 특정 포트 번호가 기입된 패킷을 필터링 함으로써 여러분은 아주 쉽게 서비스 제한을 할 수 있다. 감이 잘 오지 않으면 /etc/services 파일을 곰곰이 살펴보기 바란다. 각 서비스에 관하여 그것이 TCP를 사용하는지 UDP를 사용하는지 그리고 사용하는 포트는 무엇인지 적혀 있다.

TCP 플래그

연결지향(Connection-Oriented) 방식의 TCP 패킷에는 연결 확립을 위한 ACK/SYN 등의 비트가 있다. 이것을 필터링하면 외부에서는 여러분의 인트라넷에 접속할 수조차 없도록 할 수 있으며 오로지 내부에서 외부로의 접속만 허가할 수 있다. 왜냐하면 연결 확립을 요청하는 패킷을 모두 제거해버리기 때문이다.

ICMP 메시지 유형

ICMP 패킷에는 ICMP 메시지의 유형에 대한 정보가 담겨있다. 이 메시지의 유형을 걸러낼 수 있다. 예를 들어 어떤 호스트가 살아있는지 확인해보는 ping 명령에서 발신되는 반향 요청(Echo Request) 패킷을 거부하게 되면 외부 침입자 입장에서는 도대체 호스트 컴퓨터가 켜져 있는지 꺼져 있는지 직접 보지 않고서는 확인할 길이 거의 없다.

패킷에 든 자료, 네트워크 장치, 패킷의 날짜/시간

이러한 정보에 의거하여 여러분은 판단을 내려야 한다. 리눅스 커널에서 지원하는 필터링 정책(Policy)은 다음 3 가지이다.

- 허가(Accept)
- 부인(Deny)
- 거절(Reject)

여기서 설명이 필요한 부분은 바로 부인과 거절의 차이일 것이다. 부인은 규칙에 의거하여 필터를 통과할 수 없는 패킷을 아무 소리 없이 버리는데 반해 거절은 필터를 통과할 수 없는 패킷을 버리면서 패킷을 보낸 측에 목적지에 도달할 수 없음 메시지(ICMP Destination Unreachable)를 친절하게 보내준다. 이렇게 하면 발신지 측에서 아무런 이유도 모른 체 한 없이 기다리는 일은 없게 된다. 부인 방법과 거절 방법에 대해서는 여러분의 마음에 달렸다.

ipfwadm 명령 사용법 네트워크 보안에 관한 거의 모든 설정을 이 명령 하나로 해결한다. 따라서 여러분이 능력 있는 리눅스 네트워크 관리자가 되려면 사용법에 능숙해야 한다. 여러분도 예상하겠지만 하나의 명령에서 많은 것을 해결하기 때문에 약간 많은 옵션들을 가지고 있다. 하지만 그렇게 처음부터 어려워 할 필요는 없다. 어떤 어려운 일이든 개념만 잘 잡고 들어가면 인간이 만든 이상 이해할 수 없는 것은 없다고 자신에게 장담해보자.

사용형식

```
ipfwadm -A command parameters [options]
ipfwadm -I command parameters [options]
ipfwadm -O command parameters [options]
ipfwadm -F command parameters [options]
ipfwadm -M [ -l | -s ] [options]
```

범주

총 4개의 범주로 나눌 수 있는데 IP 회계분석(IP Accou nting), IP 입력 방화벽 (IP Input Firewall), IP 출력 방화벽(IP Ouput Firewall) 그리고 마지막으로 IP 전달 방화벽(IP Forward Firewall)이 그것이다. 영문자의 앞 자와 ipfwadm 명령의 -A, -I, -O, -F 와의 관계를 잘 보아두기 바란다.

명령(command)

ipfwadm 명령 사용형식에 있어 범주를 고른 후 그 다음 명령이 따른다.

-p <정책>

선택한 방화벽 유형에 대한 기본 정책을 설정하거나 설정 값이 있을 때는 변화 시킨다. <정책> 위치에 올 수 있는 값은 위에서도 설명한 바 있는 accept, deny, reject이다. 만약 아무런 규칙에도 해당되지 않는 패킷이 발견되면 바로 기본 정책을 사용한다. -I, -O, -F 플래그와 같이 사용할 수 있다(policy).

-f

규칙을 모두 지워버린다(flush).

-l

규칙을 화면에 표시한다(list).

-a <정책>

규칙을 추가한다(append).

-i <정책>

규칙을 맨 앞에 삽입한다(insert).

-d <정책>

규칙을 삭제한다(delete).

전달인수(parameters)

-P 프로토콜

패킷 형태를 선택한다. tcp, udp, icmp 그리고 모든 패킷을 가리키는 all을 선택 할 수 있다. 생략하면 all이라고 간주한다(Protocol).

-S 주소[/마스크 [포트 ...]]

주소에 올 수 있는 표현은 호스트명, 네트워크명 그리고 평범한 IP 주소이다. 넷 마스크를 적어주거나 또는 그냥 숫자를 적어주는 것도 가능한데 예를 들어 24는 255.255.255.0과 같다. 24의 의미는 여기서 IP 주소 32 비트에 대하여 왼쪽 24개 비트가 1이고 나머지는 0이라는 말이다. 포트에는 23, 25와 같은 포트 번호를 적거나 telnet, smtp 같은 서비스명칭 또는 ICMP 타입을 적는다. 포트의 범위를 표현할 때는 1023:65535와 같이 포트:포트라는 형식을 사용한다. 생략하면 0.0.0.0/0이라는 값을 사용하는데 이는 모든 주소를 뜻한다(Source).

-D 주소[/마스크 [포트 ...]]

-S와 사용법이 같다. 단 ICMP 포트는 명시할 수 없다(Destination).

그 밖의 옵션들(Options)

-m

전달하도록 허가된 패킷에 대하여 매스커레이딩을 실행한다. 이 옵션은 전달 방화벽의 허가(accept) 규칙하고만 같이 사용할 수 있으며 커널 컴파일 시 매스커레이딩 지원이 들어있어야 한다(masquerade).

ipfwadm 명령 사용 예 예를 2 가지 들어보겠다. 이 두 가지 예는 거의 비슷하다. 하지만 두 가지 예로 부터 서로 다른 분위기를 느낄 수 있을 것이라고 생각하여 여기 소개한다. 첫 번째 상황은 인터넷과 연결된 인터페이스에 199.1.2.10이라는 주소를 갖고 인트라넷 측과는 192.168.2.1이라는 주소를 갖는 상황이다.

```
#
# IP 패킷 회계분석과 전달에 관한 설정
#
#  전달(Forwarding)
#
# 기본적으로 모든 서비스를 부인한다.
ipfwadm -F -p deny
# 전달, 입력, 출력에 대하여 규칙을 비운다.
ipfwadm -F -f
ipfwadm -I -f
ipfwadm -O -f
# 전자우편을 여러분의 서버로 전달
ipfwadm -F -a accept -b -P tcp -S 0.0.0.0/0 1024:65535 - D 192.1.2.10 25
# 외부 전자우편 서버로의 접속요청을 전달
ipfwadm -F -a accept -b -P tcp -S 196.1.2.10 25 -D 0.0.0.0/0 1024:65535

# 웹 접속 요청을 여러분의 웹 서버로 전달
/sbin/ipfwadm -F -a accept -b -P tcp -S 0.0.0.0 /0 1024:65535 -D 196.1.2.11 80

# 외부 웹 서버에 대한 요청을 전달
/sbin/ipfwadm -F -a accept -b -P tcp -S 196.1.2. * 80 -D 0.0.0.0/0 1024:65535

# DNS 패킷을 전달
/sbin/ipfwadm -F -a accept -b -P udp -S 0.0.0.0 /0 53 -D 196.1.2.0/24

# 현재의 분석 규칙을 비운다
ipfwadm -A -f
# 회계 분석(Accounting)
/sbin/ipfwadm -A -f
/sbin/ipfwadm -A out -i -S 196.1.2.0/24 -D 0.0.0.0/0
/sbin/ipfwadm -A out -i -S 0.0.0.0/0 -D 196.1.2.0/24
/sbin/ipfwadm -A in -i -S 196.1.2.0/24 -D 0.0.0.0/0
```

```
/sbin/ipfwadm -A in -i -S 0.0.0.0/0 -D 196.1.2.0/24
```

인터넷과 PPP로 연결되어 있고 내부 네트워크와는 이서네트로 연결되어 있는 리눅스 방화벽의 방화벽 설정 예이다. 하나의 셸 스크립트에 저장해두고 부팅 할 때마다 /etc/rc.d/rc.local과 같은 곳에서 한 번씩 실행해주면 좋다.

```
#!/bin/sh
# '전달(Forwarding)' 규칙 테이블을 비운다.
# 전달에 대한 기본정책을 '허가(accept)'로 변경한다.
#
/sbin/ipfwadm -F -f
/sbin/ipfwadm -F -p accept
#
# .. 이번에는 '들어오는' 패킷에 대하여
#
/sbin/ipfwadm -I -f
/sbin/ipfwadm -I -p accept

# 우선 PPP 인터페이스를 봉쇄한다.
# 개인적으로는 '-a reject -y' 대신 '-a deny'를 사용하고 싶지만 그
# 렇게하면 인터페이스 접속 자체가 불가능하게 된다.
# -o 는 거부된 데이터그램을 기록해두도록 한다.설정 실수에 대한 어떤 공
# 격이 있었는가를 알 수 있으나 디스크 공간을 필요로 한다.
#
/sbin/ipfwadm -I -a reject -y -o -P tcp -S 0/0 -D
 172.16.174.30/sbin/ipfwadm -I -p accept

# 위조된 패킷은 그 즉시 버린다:
# 멀티캐스트/애니캐스트/브로드캐스트 주소로부터 어떠한 것도 와서는 안
# 된다.
#
/sbin/ipfwadm -F -a deny -o -S 224.0/3 -D 172.16.37.0/24
#
# 루프백 네트워크의 패킷은 회선 상을 돌아다니면 안된다.
#
/sbin/ipfwadm -F -a deny -o -S 127.0/8 -D 172.16.37.0/24

# 외부로부터 오는 SMTP, DNS 접속은 받아들이지만 오로지 메일/네임 서버
# 에게만 전달한다.
#
/sbin/ipfwadm -F -a accept -P tcp -S 0/0 -D 172.16.37. 19 25 53
#
# DNS는 TCP 뿐 아니라 UDP도 사용한다. 따라서 네임 서버에 대해서는
# UDP도 허용해야 한다.
#
/sbin/ipfwadm -F -a accept -P udp -S 0/0 -D 172.16.37. 19 53
#
# 하지만 NFS 또는 Larry McVoy의 NSF 확장 기능 등 위험한 포트로부터
# 나오는 답변은 전달하지 못하게 한다. squid를 실행하고 있다면 그 포트
# 또한 추가하기 바란다.
/sbin/ipfwadm -F -a deny -o -P udp -S 0/0 53 -D 172.16.37.0/24 2049 2050

# 그 외의 포트에 대해서는 괜찮다.
#
/sbin/ipfwadm -F -a accept -P udp -S 0/0 53 -D 172.16.37.0/24 53 1024:65535
```

```
# identd에 대한 접속 요청은 거절(reject)
# 우리는 여기서 'reject'를 사용함으로써 접속하는 호스트에게 즉시 접속
# 해도 소용없다는 반응을 보내준다. 그렇지 않으면 ident가 타임아웃 걸리
# 는 동안 지연이 생긴다.
#
/sbin/ipfwadm -F -a reject -o -P tcp -S 0/0 -D 172.16.37.0/24 113

# 192.168.64 그리고 192.168.65 네트워크는 우리가 믿는 친구들이므로
# 몇 가지 일반적인 서비스를 허가한다.

#
/sbin/ipfwadm -F -a accept -P tcp -S 192.168.64.0/23 -D 172.16.37.0/24 20:23

# 내부에서 발생하는 일에 대해서는 허가하고 통과시킨다.
#
/sbin/ipfwadm -F -a accept -P tcp -S 172.16.37.0/24 -D 0/0

# 다른 대부분의 TCP 접속 요청은 거부하고 기록해둔다.
# (만약 ftp가 제대로 작동하지 않는다면 1:1023 을 추가하라)
#
/sbin/ipfwadm -F -a deny -o -y -P tcp -S 0/0 -D 172.16.37.0/24

# UDP도 마찬가지이다.
#
/sbin/ipfwadm -F -a deny -o -P udp -S 0/0 -D 172.16.37.0/24
```

상당히 복잡한 예이긴 하지만 ipfwadm 명령을 어떻게 사용하는지 감을 잡을 수 있을 것이다.

2.6 끝내면서 : 리눅스 보안은 믿을 수 없다?

여기서 설명한 패킷 필터링 방화벽 방법 말고 프록시(Proxy) 서버 구축을 통한 방화벽 구현 방법이 있다. 리눅스를 프록시 서버로 사용하기 위한 방법 그리고 여기에 소개된 모든 내용은 NET-3 하우투, Firewall 하우투 문서에 자세히 나와 있으니 전문적인 설정이 필요한 사람들은 시간을 내서라도 꼭 읽어보기 바란다.

리눅스에 관한 수많은 정보는 이미 하우투에 있다는 것을 명심해주기 바란다. 몇몇 리눅스를 기피하는 사람들은 리눅스의 소스가 공개되어 있으므로 보안에 취약할 것이라고 단정한다. 소스를 공개하는 정책과 보안 문제는 직결되지 않는다. 여러분의 시스템을 장난기 있는 해커들이 마음껏 망칠 수 있도록 하기 위하여 소스를 공개하는 것은 아니다. 보안 문제에 있어 많은 사람들이 다음과 같이 반론한다. 적지 않은 사람들이 윈도 NT는 리눅스보다 더 안전하다고 믿는 듯 하다. 그 이유는 하나의 회사가 소스를 소유하고 공개하고 있지 않기 때문인 것 같다. 하지만 그 사람들은 커다란 착각을 하고 있다. 그들은 보안의 문제를 보여주지 않으면 모를 것이다 ' 라고 단순하게 생각하고 있으며 실제로 일반인들은 그렇게 생각하고 있는 것 같다. 윈도 NT든 리눅스든 보안 문제의 결함은 항상 드러나기 마련이다. 그것을 공개와 비공개의 문제가 아닌 소프트웨어의 원초적 특성 때문이다. 공개하지 않아서 안전하다면 누구나 그 정책을 사용할 것이다. 하지만 소스 비공개는 상업적인 이유이지 여러분의 보안을 고려하는 마음의 배려라고 착각하지 말기 바란다.

소스가 공개되어 있으므로 소스를 보고 열심히 보안 구멍을 찾는 사람이 있을 것이라고 말하는 사람도 있다. 하지만 필자가 보기에 크래커들은 소스를 열심히 보는 인간들이 아니라고 본다. 그렇게 착실하게 앉아서 소스를 보는 사람이라면 프로그래밍으로 공헌할 가능성이 많다. 여러분이 신문을 통해 접할 수 있는 사고들을 보면 하나같이 가장 쉬운 방법, 상대방의 패스워드 알아내기 같은 지루하고 비생산적인 작업을 많이 하고 있다는 것을 알 수 있다. 필자가 생각 하기에 여러분의 상식을 영화가 망치는 것 같다. 컴퓨터 해킹에 대하여 하나도 모르는 감독들이 그려내는 해커들의 모습은 신기에 가깝다.

만약 윈도 NT와 리눅스에서 똑같이 보안 상의 허점이 발견되었다고 치자. 리눅스 사용자는 그 즉시 소스 레벨

에서의 패치가 가능하다. 또한 어느 누구든 발표한 사람이 그 즉시 유즈넷 뉴스그룹 창구를 통해 빠르게 공고할 수 있다. 윈도 NT 사용자라면 마이크로소프트 사에서 공식 발표가 이뤄지기 전까지 알지 못하는 경우가 대부분이다. 그리고 패치가 나오기를 기다리면서 불안에 떨꺼나 서비스를 중단하는 수밖에 없다.

정말로 보안에 결벽 증세가 있다면 여러분의 회사에서 몇 명의 프로그래머를 고용하여 리눅스 네트워크 소프트웨어 중 몇 가지 중요한 부분을 임의로 고쳐서 사용하면 되지 않겠는가? 하지만 고용한 프로그래머는 또 어떻게 믿는가? 이 많은 불신을 리눅스 방화벽 구축을 통해 해소할 수 있다.

제 3 절 PnP 하드웨어 인식시키기

수많은 훌륭한 기능에도 불구하고 리눅스는 아직도 플러그 앤 플레이(PnP) 하드웨어를 제대로 지원하지 못하고 있다. 여러분도 알다시피 플러그 앤 플레이는 IBM 아키텍처의 하드웨어에서 주변기기를 설치할 때의 아주 고질적인 I/O 주소, IRQ, DMA 설정 문제에서의 불편함을 해소하기 위해서 나온 개념이며 이미 상당히 오래전부터 필요한 필수 기능이었다.

실제 PnP 하드웨어 인식/설정 기능이 얼마나 효율적인가의 문제는 제쳐두고라도 컴퓨터 산업의 큰 힘들에 의하여 이미 표준적인 기능이 되어버린 상태이므로 반론의 여지없이 PnP 기능은 리눅스 커널의 주요 기능으로 자리잡아야 한다고 본다.

물론 대부분의 장치에 대하여 하드웨어 인식시에 잘 찾아내어 사용하지만 표준적인 I/O 주소 또는 IRQ를 사용하지 않는 경우에는 제대로 인식하지 못하는 경우가 많다. 주변장치를 얼마 갖고 있지 않은 사람들에게는 별 문제되지 않지만 이서네트 카드를 두 개 인식시켜야 한다든지 할 때는 골치 아픈 일이 아닐 수 없다.

대부분은 주변장치와 같이 포장되어 있는 PnP 기능을 끄는 도스용 소프트웨어를 가지고 특정 주소와 IRQ를 맞춰서 사용하기도 하지만 필자의 경험으로는 제대로 되지 않는 경우가 허다했다. 이에 대한 답이 있다. 아직 리눅스 커널에서 PnP가 지원되지는 않지만 ISA PnP 툴이라는 것을 사용하면 거의 대부분의 문제를 해결할 수 있다.

3.1 ISA PnP 도구, 어디서 구할 수 있는가?

- 웹 페이지 `www.redhat.com/linux-info/pnp/`
- FTP 사이트 `ftp.redhat.com` `ftp.demaon.co.uk`
- 리눅스 동호회 자료실

그리고 여러분이 프로그램세계 1월호 부록인 알짜 레드햇 4.0 CD를 가지고 있다면 다면 CD롬을 마운트한 후 마운트 디렉터리로부터 misc/pnp/utils 위치에서 isapnptools-1.8.tgz 파일을 찾아볼 수 있을 것이다. 필자가 본 최신 소스는 1.9 버전이었고 알찬 맨 페이지도 들어있었다.

3.2 ISA PnP 도구의 구성

```
/etc/isapnp.conf
/usr/man/man5/isapnp.conf.5
/usr/man/man8/isapnp.8
/usr/sbin/isapnp
/usr/sbin/pnpdump
```

3.3 ISA PnP 도구의 사용법

몇 가지 간단한 규칙만 익히면 된다. ISA PnP 도구의 사용법은 다음과 같은 절차를 거친다.

1. pnpdump로 현재의 ISA용 PnP 하드웨어에 대한 정보를 입수한다.
2. /etc/isapnp.conf 파일을 보면서 설정하고 싶은 I/O 주소, IRQ 등을 설정한다.
3. isapnp를 실행시켜 PnP 하드웨어를 설정한다.
4. 모듈을 로딩한다.

여기서 한 가지 중요하게 짚고 넘어가야 할 점이 있다. ISA PnP 도구를 사용하여 설정할 수 있는 하드웨어는 리눅스에서 그 기능을 모듈로 제공하는 것에만 유효하다. 이미 커널 안으로 들어가 버린 기능에 대해서는 ISA PnP 도구를 쓸 기회가 주어지지 않기 때문이다. 하지만 리눅스 커널 기능의 대부분이 모듈로 독립하여 사용할 수 있기에 문제되는 경우는 거의 없을 것이다.

3.4 ISA PnP 도구 컴파일하기

필자가 컴파일한 환경은 커널 버전 2.0.30, C 라이브러리 5.4.23, 컴파일러는 gcc 2.7.2.1 이다. 간단히 소스를 풀고 make 명령을 내리면 다음과 같은 에러 메시지가 나온다.

```
[yong@freeyong isapnptools-1.8]$ make
cc -O2 -Wall -DREALTIME -DNEEDSETSCHEDULER
 -DNEEDNANOSLEEP  -c pnpdump.c -o pnpdump.o
pnpdump.c:36: conflicting types for ‘nanosleep’
/usr/include/time.h:124: previous declaration
 of ‘nanosleep’
make: *** [pnpdump.o] Error 1
```

개발자의 개발환경이 어떤지는 모르겠으나 Makefile에서 PNPFLAGS 부분에서 -DNEEDNANOSLEEP을 삭제한다. 이 방법은 README에 적혀있는 대로이다.

```
[yong@freeyong isapnptools-1.8]$ make
cc -O2 -Wall -DREALTIME -DNEEDSETSCHEDULER
 -c pnpdump.c -o pnpdump.o
cc  pnpdump.o  -o pnpdump
flex -i -t isapnp.y > isapnp.c
cc -O2 -Wall   -c isapnp.c -o isapnp.o
isapnp.y:934: warning: prototype for ‘yywrap’ follows
isapnp.y:76: warning: non-prototype definition here
isapnp.y:1360: warning: ‘yyunput’ defined but not used
cc  isapnp.o  -o isapnp
```

make install 과정은 없으므로 pnpdump와 isapnp 바이너리를 /sbin 디렉터리에 직접 복사한다.

3.5 PnP 정보를 덤프하기

```
# pnpdump > /etc/isapnp.conf
```

pnpdump 명령을 내리면 여러분의 PnP BIOS를 점검하고 PnP 하드웨어를 발견하게 되면 리다이렉션으로 주어진 /etc/isapnp.conf 파일에 결과를 출력한다.

3.6 /etc/isapnp.conf 파일 편집하기

리눅스/유닉스에서의 작업이 다 그러하다는 것을 알 것이다. 편집기를 가지고 난해하기 짝이 없는 설정 파일을 여러분의 상황에 알맞게 편집한다. 파일 형식 이 어떠한지 알아보도록 하자.

```
# $Id: pnpdump.c,v 1.8 1997/01/14 21:05:35 fox Exp $
# This is free software, see the sources for details.
# This software has NO WARRANTY, use at your OWN RISK
#
# For details of this file format, see isapnp.conf(5)
#
# Compiler flags: -DREALTIME -DNEEDSETSCHEDULER -DNEED
# NANOSLEEP
#
# Trying port address 0203
# Trying port address 0207
# Trying port address 020b
# Board 1 has serial identifier 5e 00 19 1c 0b 19 80
# 8c 4a

# (DEBUG)
(READPORT 0x020b)
(ISOLATE)
(IDENTIFY *)

# Card 1: (serial identifier 5e 00 19 1c 0b 19 80 8c 4a)
# RTL8019 Serial No 1645579 [checksum 5e]
# Version 1.0, Vendor version 1.0
# ANSI string -->Plug & Play Ethernet Cardb --
#
# Logical device id RTL8019
#   Device support I/O range check register
#
# Edit the entries below to uncomment out the
# configuration required.
# Note that only the first value of any range is given,
# this may be changed if r
# Don't forget to uncomment the activate (ACT Y)
# when happy

(CONFIGURE RTL8019/1645579 (LD 0
#   Compatible device id PNP80d6
#   Logical device decodes 10 bit IO address lines
#     Minimum IO base address 0x0220
#     Maximum IO base address 0x0380
#     IO base alignment 32 bytes
#     Number of IO addresses required: 32
(IO 0 (BASE 0x0340))
#   IRQ 3, 4, 5, 9, 10, 11, 12 or 15.
#     High true, edge sensitive interrupt
(INT 0 (IRQ 10 (MODE +E)))
(ACT Y)
))
# End tag... Checksum 0x00 (OK)
```

설정 파일이 상당히 복잡해 보이기는 하지만 그 모든 것을 다 이해할 필요는 없다. 필자도 복잡하기 짝이 없는 isapnp.conf 맨 페이지를 모두 다 읽어보지는 않았다. 설정 파일에서 샵(#) 문자로 시작하는 줄은 여러분도 예상하다시피 주석문이다. 따라서 위의 설정 파일에서 실제로 사용되는 부분은 다음과 같다.

```
(READPORT 0x020b)
(ISOLATE)
(IDENTIFY *)
(CONFIGURE RTL8019/1645579 (LD 0
(IO 0 (BASE 0x0340))
(INT 0 (IRQ 10 (MODE +E)))
(ACT Y)
))
```

처음에는 (ACT Y)부터 시작해서 거의 모든 것이 주석문 처리되어 있을 것이다. 여러분의 작업은 주석문 처리되어 있는 것 중에 알맞은 것을 선택하여 # 문자를 지워 활성화시키는 일, 주석문에 자세히 적힌 I/O 주소, IRQ 범위에서 다른 주변기기와 충돌하지 않는 주소와 IRQ를 선택하는 일이다. 주석문이 영어로 되어 있기는 하나 필자가 생각하기에는 맨 페이지보다도 더 중요하고 실전적인 지식을 알려준다.

```
#     Compatible device id PNP80d6
#     Logical device decodes 10 bit IO address lines
#     Minimum IO base address 0x0220
#     Maximum IO base address 0x0380
#     IO base alignment 32 bytes
#     Number of IO addresses required: 32
```

위와 같은 문장을 볼 수 있을 것이다. 지정할 수 있는 I/O 최소 번지는 0x0220 이고 최대 번지는 0x0380이라고 가르쳐 주고 있다. 설정하기 전에 이미 리눅스가 부팅되고 나서 사용 중인 I/O 주소를 피해야 하므로 현재 사용중인 I/O 주소에 대하여 알아야 할 것이다. 이럴 때는 리눅스의 재미있는 파일 시스템인 /proc 시스템을 이용하자.

```
# cd /proc
# cat ioports
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
0070-007f : rtc
0080-009f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
00f0-00ff : npu
0170-0177 : ide1
01f0-01f7 : ide0
0220-022f : sound blaster
0230-0233 : sbpcd
02f8-02ff : serial(auto)
0376-0376 : ide1
0378-037f : lp
0388-038b : OPL3/OPL2
```

중복되는 주소만 피해서 설정하면 된다.

```
#     IRQ 3, 4, 5, 9, 10, 11, 12 or 15.
#     High true, edge sensitive interrupt
(INT 0 (IRQ 10 (MODE +E)))
```

이번에는 IRQ를 설정하는 부분이다. 친절하게 설정할 수 있는 IRQ로는 3, 4, 5, 9 등이 있다고 알려주고 있다. 역시 이미 사용중인 IRQ를 지정할 수는 없으므로 현재 사용중인 IRQ를 점검해보자.

```
# cd /proc
# cat interrupts
 0:  23254768  timer
 1:   312846  keyboard
 2:      0  cascade
 3:  6309207 + serial
 4:      1 + serial
 5:      1  sound blaster
 8:      0 + rtc
11:     60 + aic7xxx
12:   779889  PS/2 Mouse
13:      1  math error
14:  1616870 + ide0
15:   208796 + ide1
```

빈 IRQ 번호를 정하라.

3.7 isapnp 명령으로 PnP 설정하기

가장 어려운 고비인 /etc/isapnp.conf 파일 설정을 마쳤다.

```
# isapnp /etc/isapnp.conf
```

이렇게 하면 시스템이 잠깐 멈추는 기색을 보이고 나서 성공적으로 PnP 설정을 마칠 것이다. 만약 여러분이 제공한 정보가 올바르지 않아 PnP BIOS에 의해 설정이 거부되면 에러 메시지가 뜰 것이다.

3.8 원하는 모듈을 띄운다

필자는 대만제 PnP NE2000 호환 이서네트 카드를 인식시키려고 isapnp 도구를 사용하였다.

```
# modprobe ne io=0x340,irq=10
```

또는 커널 데몬(kerneld)을 사용하는 경우 /etc/conf.modules 파일에 다음 줄을 한 줄 추가하면 된다. 다음은 필자의 예이다.

```
alias scsi_hostadapter aic7xxx
alias eth1   ne
options sbpcd  sbpcd=0x230,1
options ne   io=0x340,irq=10
options wdt   irq=6
```

한 때 사운드블래스터 32 기종과 3Com509b의 인식 문제가 크게 대두된 적 있다. 사운드블래스터 최신 기종들이 IDE 인터페이스를 달고 나오며 그 IDE 인터페이스가 우리도 모르게 IRQ 10 또는 11번을 장악하는 사실을 몰랐기 때문이다. 안다 할지라도 사운드블래스터의 PnP 기능, IDE 인터페이스 기능을 마비시키는 것이 제대로 먹혀들지 않는 것 같았다. 필자도 알짜 레드햇 4.1 제작 당시 한 대의 테스트 머신에서 사운드블래스터 32를 사용하는 바람에 3Com509b를 포기한 적이 있었다.

끝으로 정말로 힘든 일을 해준 Peter Fox씨에게 감사하는 마음 갖는다. 그는 마이크로소프트 사의 PnP 스펙을 보고 힘겹게 리눅스용 프로그램을 만들었다고 한다. 한 사람의 노력으로 전세계의 사람들이 혜택을 누릴 수 있다는 것, 그것만큼 값진 것이 어디 있을까? 여러분도 프로그래머라면 금전적 이익 이외에 이러한 명예를 누려보는 것이 어떤가? 리눅스에 기여할 수 있는 분야는 정말 많다.

제 4 절 프로그래밍

엄청난 하드웨어의 성능 발전을 소프트웨어 기술이 따라가지 못하고 있다는 지적이 많다. 눈 깜짝할 사이에 이미 펜티엄 프로 200을 넘어서고 있고 멀티미디어 관련 MMX다 뭐다 해서 인텔사는 새로운 구매 욕구에 충동질을 하고 있다. 필자가 원하든 원하지 않든 또 한 번 하드웨어 갈아치우기 전쟁이 일어날 듯 하다.

하드웨어적 발전 중에서도 이번에는 SMP(Symmetrical MultiProcessing) 그리고 병렬 처리 개념을 사용하는 쓰레드(thread) 프로그래밍에 대한 얘기를 잠깐 해보고자 한다. 이번에 다루는 내용은 소개 수준 밖에 안된다는 것을 미리 일러두고자 한다.

병렬처리 개념을 사용하면 이익을 얻을 수 있는 분야는 역시 인터넷 서버라고 할 수 있다. 지금 현재는 웹 서버의 경우 HTTP 서비스를 요구하는 클라이언트의 요구가 있을 때마다 자기 스스로를 복제(fork)하여 그 복제 프로세스로 하여금 클라이언트에게 서비스를 제공하고 메인 프로세스는 계속적으로 특정 포트(일반적으로 웹 서버는 80번 포트 또는 8080번 포트)에 귀기울이는 형태를 지닌다. 이미 유닉스 시스템에는 웹 서버와 같이 클라이언트들의 동시 접속, 동시 서비스 제공을 해결할 능력을 가지고 있다. 그럼에도 불구하고 쓰레드 프로그래밍은 좀 더 나아가려고 한다. 기술적인 얘기로는 유닉스의 프로세스 복제에 걸리는 시스템의 부하가 많기 때문에 그보다는 개선된 형태 즉 쓰레드 (thread)라고 불리우는 경량급(light-weight) 프로세스 모델을 사용하여 빈번한 서버 처리 업무에 효율성을 기하자는 내용이다. 더군다나 ATM 교환기 등 초 고속 네트워크가 건설되면 약간의 시간 차이라 할지라도 서버의 처리 능력은 커다란 문제로 떠오르지 않을 수 없다. ATM 교환기를 통해서 쏟아져 오는 패킷을 제대로 처리하지 못한다면 비싼 돈 들여 건설한 네트워크 하드웨어가 무슨 소용 있겠는가?

쓰레드가 새롭게 만들어지는 것과 프로세스가 새롭게 만들어진 것 사이에는 약간의 차이가 있다고 한다. 프로세스가 복제될 때는 복제를 행하는 부모 프로세스와 상대적으로 적은 양의 정보를 공유한다고 한다. 하지만 쓰레드의 경우에는 예를 들어 전역 변수(global variable), 정적 지역 변수(static local variable), 그리고 열려진 파일 기술자, 프로세스 ID 등 더 많은 정보를 공유한다고 한다. 프로세스의 경우 개별적인 복제물을 만들어내는데 쓰레드에 비해 상대적으로 부하가 많이 걸리며 한 프로세스에서 다른 프로세스로 제어권을 넘기는데 (context switch라고 한다) 걸리는 시간이 한 쓰레드에서 다른 쓰레드로 이동하는 시간보다 상대적으로 더 많이 걸린다고 한다.

유즈넷 뉴스그룹에 가보면 쓰레드 프로그래밍에 대한 찬반이 엇갈리지만 어찌 되었든 쓰레드 방식의 프로그래밍은 표준적인 프로그래밍 라이브러리의 위치로 들어오기 시작했다. 리눅스 커널은 2.0 버전이 되면서부터 멀티-쓰레드 프로세스를 다룰 수 있게 되었고 따라서 쓰레드 프로그래밍에 길을 열어주고 있다.

4.1 쓰레드 프로그래밍을 하기 위해 필요한 것은?

리눅스 C 라이브러리 버전 5.x 대의 라이브러리에서는 사용자 레벨 (User-Level)의 쓰레드 라이브러리가 들어있으나 그렇게 쓸만 한 것은 아니라 는 말들이 많다. 몇 번 시험해본 결과 구현되지 않은 것들도 몇 가지 있어서 아예 컴파일 자체가 안되는 경우도 허다했다.

여러분이 구해야 할 것은 Xavier Leroy씨의 커널 수준(Kernel-Level) 쓰레드 라이브러리이다. 커널 수준의 쓰레드란 리누스씨가 커널 버전 2.0 이하에서 제공하고 있는 clone()이라고 하는 새로운 기능에 의거한 쓰레드를 말한다. 현재 소개 수준에 그치는 이 글에서 사용자 레벨의 쓰레드와 커널 수준의 쓰레드는 크게 구분할 필요없으며 API는 같으므로 상관하지 않아도 좋다.

소스를 구하여 수동으로 설치해야 하는데 그 위치는 다음과 같다. `pauillac.inria.fr/\~{}xleroy/linuxthreads/` 레드햇 배포판 사용자는 손쉽게 커널 쓰레드 라이브러리를 설치할 수 있다. 레드햇 배포판 버전 4.1에 패키지가 추가되어 있기 때문이다. 여러분이 설치해야 할 패키지명은 linuxthreads, linuxthreads-devel 이렇게 2 개의 패키지이다. 마찬가지로 Leroy씨의 커널 쓰레드 라이브러리이다.

다음 예제를 컴파일해보자. (이 소스는 리눅스 저널 97년 2월호 ISSUE 34호, Martin McCarthy씨의 원고에서 Listing 6.에서 인용한 것입니다)

```
{{{{#define _REENTRANT
#include <stdio.h>
#include <pthread.h>
```

```
#define MATSIZE     4

/* 함수 원형 */
void* matMult ( void* );

/* 전역 행렬 자료 */

int mat2[MATSIZE][MATSIZE] =
  {  {1, 2, 3, 4 },
     {4, 5, 6, 7 },
     {7, 8, 9, 10 },
    {10, 11, 12, 13 } };
int mat1[MATSIZE][MATSIZE] =
  {  { 9, 8, 7, 6 },
    { 6, 5, 4, 3 },
    { 3, 2, 1, 0 },
    { 0, -1, -2, -3 } };
int result[MATSIZE][MATSIZE];

int
main( void )
{
    pthread_t thr[MATSIZE];

    int i, j;

    for ( i = 0 ; i < MATSIZE ; ++i )    {
        pthread_create ( &thr[i], NULL, matMult, (void*)i );
    }
    for ( i = 0 ; i < MATSIZE ; ++i )    {
        pthread_join ( thr[i], NULL);
    }
/* 소스 다음에 계속 */
/* 소스 앞에서 이어짐 */

    for ( i = 0 ; i < MATSIZE ; ++i )    {
        printf ("|");
        for ( j = 0 ; j < MATSIZE ; ++j )
            printf ("%3d ", mat1[i][j] );
        printf ("|%c|", ( i==MATSIZE/2 ? 'x' : ' ') );
        for ( j = 0 ; j < MATSIZE ; ++j )
            printf ("%3d ", mat2[i][j] );
        printf ("|%c|", ( i==MATSIZE/2 ? '=' : ' ') );
        for ( j = 0 ; j < MATSIZE ; ++j )
            printf ("%3d ", result[i][j]);
        printf ("|\n");
    }

    return 0;
}

void*
matMult ( void* col )
{
```

```
    int i, j;
    int val;

    for ( i = 0 ; i < MATSIZE ; ++i )
    {
        result[i][(int)col] = 0;
        for ( j = 0 ; j < MATSIZE ; ++j )
            result[i][(int)col] += mat1[i][j] *
                mat2[j][(int)col];
    }

    return NULL;
}
```

아래는 실행 결과를 보여준다.

```
$ gcc thread_ex1.c -o thr_ex1 -lpthread
$ ./thr_ex1
| 9  8  7  6 |   | 1  2  3  4  |   | 150 180 210 240 |
| 6  5  4  3 |   | 4  5  6  7  |   |  84 102 120 138 |
| 3  2  1  0 | x | 7  8  9 10  | = |  18 24 30 36    |
| 0 -1 -2 -3 |   | 10 11 12 13 |   | -48 -54 -60 -66 |
```

쓰레드 프로그래밍 방식으로 행렬을 계산한 예이다. 사실 쓰레드 프로그래밍을 배우기 어렵다기 보다는 어디에 활용할 것인가를 찾는 것이 더 어렵다고 말할 수 있을 것 같다. 쓰레드의 활용 영역은 앞서 얘기한 다중 클라이언트 지원 서버 프로그래밍 그리고 여기서 보는 것처럼 행렬 연산 그리고 둠과 같이 독립적으로 움직이는 적들이 많이 등장해야 하는 게임을 들 수 있다. 가까운 예로는 지난달에 소개했던 MPEG Layer3 디코더/플레이어인 splay가 바로 pthread 라이브러리를 사용하고 있다.

행렬 연산에 대해서 알아보자. 위에서는 4x4 정방행렬 2 개의 행렬 곱을 처리하는 과정을 보여주고 있다. 여러분이 고등학교 수학을 마쳤다면 행렬 곱이 어떤 식으로 이뤄지는지 그 규칙을 알고 있을 것이다. 앞 행렬의 m 번째 행과 뒤 행렬의 n 번째 열을 계산하여 결과 행렬의 m 행 n 열의 원소가 된다. 행렬 곱 규칙을 잘 살펴보면 행과 열의 곱은 서로의 결과에 영향을 받지 않고 독립적으로 행해지는 연산임을 알 수 있다. 따라서 각 행과 열의 곱은 병렬 처리하기가 아주 좋다.

필자는 쓰레드 프로그래밍을 보고 있노라면 손오공이 적을 물리치기 위하여 자신의 머리털을 뽑아 자그마한 분신들을 만들어 공격하는 모습을 연상하곤 한다.

4.2 POSIX 쓰레드

멀티쓰레딩에 관하여 POSIX 표준이 이미 마련되어 있다. 따라서 POSIX 쓰레드 표준에 따른 프로그래밍을 한다면 소스 코드 수준에서 리눅스에서뿐 아니라 다른 비슷한 유닉스에서 같은 프로그램을 운영할 수 있다. 리눅스 쓰레드 라이브러리들은 POSIX 표준을 따르므로 걱정할 필요없다.

4.3 관련된 이야기

쓰레드에 대하여 이야기하면서 빠뜨릴 수 없는 주제들이 있다. 바로 병렬 다중 프로세서 SMP와 마크(Mach) 커널이 바로 그것이다. 요즘 우리는 심심지 않게 2 개의 프로세서를 장착할 수 있는 보드를 볼 수 있다. 그리고 기존의 유닉스 커널과는 다른 설계 방식의 마크(Mach) 커널에 대한 얘기를 심심치 않게 듣게 된다. 쓰레드 프로그래밍은 바로 이런 환경에서 가장 뛰어난 효율을 발휘한다고 한다.

카네기 멜런 대학(CMU)에서 시작한 새로운 운영체계 연구 프로젝트의 의 하나인 마크 프로젝트는 1985년부터 시작되었다고 하니 꽤 오랜 시간을 거친 시스템이라고 할 수 있다. 1994년 CMU의 마크 커널 개발은 중단되

었고 마크 커널의 개발은 Open Software Foundation, 유타 대학의 Flexmach, 헬싱키 대학의 LITES 시스템, 그리고 FSF의 Hurd 시스템으로 개발이 진행 중이다. 마크 커널에 대한 자세한 정보는 다음 사이트에서 얻을 수 있다. www.cs.cmu.edu/afs/cs.cmu.edu/project/mach/public/www/mach.html

안타깝게도 리눅스는 마크 커널 위에 만들어진 것이 아니며 전통적인 유닉스 스타일로 만들어진 것이다. 하지만 이미 애플 사에 의해 만들어진 MkLinux는 마크 커널 위에 리눅스를 올려놓는데 성공하였다. 아직 현실적인 힘을 발휘하기에는 많은 시간을 기다려야 할 것으로 예상되는 운영체계 GNU는 마크 커널 방식이며 Hurd라는 커널을 사용한다. 최근 들어 GNU 측에서도 마크 커널을 내놓았다고 한다. 마크 커널에 대한 관심은 무엇보다도 애플 사로 인수된 넥스트스텝의 하부 기술이기에 더욱 커지리라 본다. GNU Hurd에 관한 정보는 다음 사이트를 방문해보기 바란다. www.gnu.ai.mit.edu/software/hurd/hurd.html

제 5 절 리얼타임 리눅스

지금부터 소개할 내용은 RT-Linux라고 불리는 것으로서 멀티태스킹 OS에서 해결하기 힘든 리얼 타임(Real-Time) 지원을 리눅스에 가져온 독특한 프로젝트이다. 일반적으로 선점형 멀티태스킹 시스템에서 정확한 시간 주기를 가지고 일을 해야 하는 프로세스에게 커널이 정확한 타이밍을 맞춰준다는 것은 거의 불가능한 일처럼 보인다.

거의 대부분의 작업들(예를 들어 워드 프로세싱, 웹 브라우징 등)은 짧은 시간 간격, 정확한 주기와는 별 상관없는 것들이다. 몹시 부하가 걸려 워드 프로세서에게 넘겨져야 할 제어권이 잠시 지연된다던가 해서 워드 작업을 망치거나 하지는 않는다. 네트워크 사정 때문에 멈추기 일쑤인 브라우징 작업이 1-2 초 다른 작업에 의해 멈춰진다 해도 문제될 것은 없다.

필자로서는 정확한 주기를 가지고 시그널을 보내야 한다든지 하는 중요한 작업에 어떤 일이 있는지 잘 상상이 안되지만 어찌 되었든 실시간 지원 OS에 대한 중요성을 빠뜨릴 수 없는 것 중 하나라고 한다. 이 문제를 리눅스에서 해결해야만 관찰/감시 카메라, 로봇 또는 여러 과학 관련 도구들을 리눅스를 운영하는 PC에 연결하여 활용하는 것이 가능해진다.

리눅스에서 정확한 타이밍이 이뤄지지 않는 순간에 대해서 서술하라면 멀리서 찾지 않아도 된다. 여러분은 대부분 덩치도 크고 뜨는데 시간도 많이 걸리는 넷스케이프를 리눅스/X 윈도에서 띄워본 적이 있을 것이다. 뜨는 동안 마우스 조차 움직일 생각을 하지 않고 모든 프로그램들이 헤매는 모습을 쉽게 확인할 수 있다.

5.1 소프트 리얼 타임 지원

어떠한 패치 없이 표준 커널 안에 들어있는 리얼 타임 지원을 사용하는 것을 소프트 리얼 타임 지원이라고 부르고 있다. 커널 컴파일 설정시 문자 장치(Character Devices)–>Enhanced Real Time Clock Support를 설정하고 나서 커널 컴파일을 해보도록 하자. 그리고 나서 메이저 번호 10, 마이너 번호 135의 문자 장치 파일 /dev/rtc 파일을 만든다.

```
# mknod /dev/rtc c 10 135

[yong@freeyong 9705]$ cat /proc/interrupts
 0:  2859962  timer
 1:   140285  keyboard
 2:      0  cascade
 3:  1447093 + serial
 4:      1 + serial
 5:      1  sound blaster
 8:      0 + rtc
10:     430 + aic7xxx
12:   202636  PS/2 Mouse
13:      1  math error
14:   252100 + ide0
```

```
15:   421489 + ide1
```

리얼 타임 지원 기능을 갖는 커널로 부팅하고 나서 인터럽트 사용 현황을 살펴보면 8번 인터럽트를 rtc 기능에서 사용하고 있다는 것을 볼 수 있다.

자, 리눅스 소프트 리얼 타임 지원을 시험해보기 위한 프로그램을 하나 구해와야 한다. 다음은 바로 리눅스 커널 내부 기능을 사용하는 rt 라는 프로그램 실행기의 LSM(Linux Software Map) 파일이다.

```
Legin3
Title:          rt - run program on realtime priority
Version:        1.0
Entered-date:   12 Jan 1997
Description:    rt starts any program on realtime priority, like nice
                does with lower priority. Lets you choose scheduler
                class (SCHED_RR vs SCHED_FIFO policy) and priority.
Keywords:       linux, real time, scheduler
Author:         Boris Tobotras <boris@xtalk.msk.su>
Maintained-by:  Boris Tobotras <boris@xtalk.msk.su>
Primary-site:   sunsite.unc.edu /pub/Linux/system/Admin
                10 kb rt-1.0.tar.gz
Alternate-site: ftp.macsimum.ru /pub/Linux/local
                10 kb rt-1.0.tar.gz
Platform:       Linux 1.3.87 and later
Copying-policy: GPL
End
```

소스를 가져와 컴파일하고 설치하는 과정을 소개할 필요없이 아주 간단하다. 결과로 나오는 바이너리명은 rt이다.

```
Usage: rt [options] command

Recognized options are:
    -r  Use SCHED_RR scheduling policy (default);
    -f  Use SCHED_FIFO scheduling policy;
    -p  N Set priority level N. If N<0, set priority level
        to sched_get_priority_min(POLICY)-N.
        Default is (sched_get_priority_max()+sched_get_priority_min())/2;
    --  End of options;
    -v  Be verbose;
    -h  This help screen.
```

지금은 splay와 같은 훌륭한 MPEG Layer3 플레이어가 나왔으므로 사용할 필요가 없어졌지만 시험상 예전에는 바이너리로만 주어진 셰어웨어 버전의 mp3play를 사용한 적이 있었다. mp3play는 사실 스크립트에 불과하고 l3dec라는 프로그램과 wavplay를 불러다 사용하였다. 문제는 펜티엄 120 이상에서도 조금만 다른 프로세스가 동작한다든지 할 때는 심하게 음이 끊기는 것이었다. 소스 코드가 공개되어 있지 않으므로 별다른 해결 방법을 찾지 못하다가 rt라는 프로그램을 알게 되었고 리눅스 커널에 이미 어느 정도의 리얼 타임 지원이 들어있었다는 것을 그제서야 알게 되었다.

```
rt [옵션들] -- <리얼타임 지원을 원하는 명령>
( 위에서 -- 는 rt에 대한 옵션의 끝임일 알린다 )
```

이렇게 수행시키면 예를 들어 mp3play의 경우 넷스케이프와 같은 프로그램이 시작해도 rt를 통해 수행시킨 프로세스에게 상당한 우선권이 부여되어 있으므로 전혀 끊기지 않는 현상을 목격할 수 있었다. 루트 사용자만 사용할 수 있다. 당연히 한 프로세스에 대하여 특별한 우선권이 주어졌다는 것은 전체적으로 다른 프로세스들에게 악영향이 간다는 것을 의미한다.

5.2 하드 리얼 타임 지원

현재 리눅스 커널 안에서 기본적으로 제공하고 있는 리얼 타임 지원을 소프트라고 부르는 것은 Michael Barabanov씨와 Victor Yodaiken씨의 RT-Linux 프로젝트에 대한 상대적인 개념으로서 그렇게 부르는 것이다. 커널 패치의 형태로 주어지는 RT-Linux는 더욱 더 강력한 형태의 리얼 타임 지원을 해주도록 리눅스 커널 구조에 약간의 변화를 준 것이다. 관련 사이트는 다음과 같다. luz.cs.nmt.edu/\~{}rtlinux/ 여러분은 여기서 일단 커널 2.0.27에 대한 패치 형식으로 된 rtlinux-0.5.tgz 파일을 받아와야 한다. 커널 2.0.29에 대해서는 웹 페이지에서 별도의 패치 파일을 받아오면 된다. 파일을 풀어보면 rtlinux-0.5라는 디렉터리가 만들어지고 kernel_patch라는 파일을 하나 볼 수 있을 것이다. 그 파일을 /usr/src 디렉터리 밑으로 복사한다. 레드햇 4.1 사용자라면 이미 커널 버전은 2.0.27 일 것이다.

```
cd /usr/src/linux
patch -p2 < ../kernel_patch
```

patch 프로그램에 -p2 옵션을 주는 것에 유의하자.

```
make config
make dep ; make clean
make zlilo
make modules ; make modules_install
```

일반적인 커널 컴파일 과정을 마치고 나면 /lib/modules/2.0.27/misc 디렉터리에 rt_prio_sched.o 라는 모듈 파일이 하나 생긴다. 리얼타임 지원을 사용하려면 바로 저 모듈을 로딩해주어야 한다.그리고 나서 rt_fifo_new.o 모듈도 로딩한다.

```
modprobe rt_prio_sched.o
modprobe rt_fifo_new.o

[root@freeyong /root]# lsmod
Module         Pages      Used by
rt_fifo_new      2      [rt_process]  0
rt_prio_sched    1                    0
sbpcd           14                    1
aic7xxx         10                    0
bsd_comp         1                    0
ppp              5      [bsd_comp]    1
slhc             2      [ppp]         1
lp               2                    0
```

모듈이 로딩되면 리눅스 커널과 하드웨어 레벨 사이에 리얼 타임 커널층을 끼워지게 되어 둘 간의 처리를 중간에 가로채어 처리하게 되는 형태이다. 그 다음에 해주어야 하는 일은 /dev 디렉터리에 적절한 장치를 만드는 일이다. 배시 사용자라면 명령행에서 다음과 같이 입력해주면 된다.

```
for i in 0 1 2 3; do mknod /dev/rtf$i c 63 $i; done
```

펜티엄 120 머신에서 RT-Linux는 150 마이크로초 수준의 정확도를 가지고 스케쥴링을 해줄 수 있다고 한다.

여러분의 시스템에 리얼 타임 지원이 필요한가? 대부분의 사람들에게는 물론 논외의 문제일 것이다. 그리고 소프트 리얼 타임 지원만으로도 충분할 것이라고 본다.

5.3 테스트 해보기

rtlinux-0.5 디렉터리에 보면 testing이라는 디렉터리를 볼 수 있을 것이다. 각 디렉터리의 README를 읽어보고 테스팅을 해보기 바란다. sound 디렉터리에 있는 예를 실행해보면 리얼 타임 지원이 어떠한지 감을 잡을 수 있을 것이다. 필자가 글을 쓰고 있는 순간 rtlinux-0.5 버전 패키지에 든 소스는 제대로 동작하지 않으므로 웹 페이지에서 패치파일을 가져와서 패치한 후 컴파일해야 한다. sound 디렉터리 그 위치에서 patch < soundpatch 하고 나서 README에 따라 테스트해본다.

리눅스를 창조력을 가지고 도전하는 사람들로 인해 그 빛을 발휘하는 것이라는 생각이 든다.

제 6 절 리눅스 소프트웨어 개발자들은 요즘 무슨 생각을 하고 있는가?

6.1 포팅, 포팅, 포팅

리눅스는 정말로 수많은 아키텍처로 포팅되고 있다. '리눅스 저널' 번역 기사에서도 나갔듯이 인텔 CPU로부터 시작한 리눅스는 아미가(Amiga), 아타리 (Atari) - 우리 나라에서는 참 보기 힘든 컴퓨터들이다 -, 알파(DEC/Alpha), ARM, MIPS, 파워PC, SPARC 등으로 포팅되어 가장 많은 운영 플랫폼을 갖는 운영체계가 되어버렸다.

이 외에도 286 그리고 XT 컴퓨터에서 돌아갈 수 있도록 리눅스를 다운사이징하여 내장 시스템(Embeddable System)으로 사용할 수 있게 하는 프로젝트도 진행중이다. 리눅스를 작게 만드는 작업은 CE 환경에도 필수적이라고 본다. 이 와는 정반대로 VAXLinux를 만들려는 움직임도 있다.

리누스(Linus) 자신은 자신의 코드에 대하여 리눅스가 너무도 인텔 x86 코드를 많이 사용했기 때문에 다른 플랫폼으로의 포팅은 거의 불가능할 것이라고 말한 것과는 정반대의 길을 걷고 있다. 리눅스는 더 이상 리누스 자신만의 것이 아니며 혼자 만드는 것도 아니다.

이제 적지 않은 사람들이 인텔 리눅스 이외에도 스팍 리눅스를 사용하기 시작하는 것 같다. 유닉스 진영에 대해서도 충분히 위협적인 존재가 될 것인지 두고 보기로 하자. 리누스가 꿈꾸는 '세계 정복(World Domination)'이 그냥 허황된 꿈은 아니라는 생각을 갖게 한다.

리눅스 관련 프로젝트에 대하여 쉽게 알아볼 수 있는 페이지는 역시 `sunsite.unc.edu/linux` 페이지이다.

6.2 일반 사용자를 위한 리눅스, 쉬운 리눅스

리눅스가 많은 사용자층을 확보하면서 '일반 사용자를 위한 리눅스', '지금 보다 더 쉽게 다가갈 수 있는 리눅스' 에 대한 얘기가 뉴스그룹에서 상당히 많이 오고간다. 쉬운 리눅스란 전적으로 윈도 95와 같은 사용자 직관성, 폭넓은 프로그램의 존재를 의미하는 것처럼 느껴진다. 모티프는 질렸다. 일관된 인터페이스가 필요하다.

많은 사람들이 이구동성으로 이야기하는 바이다. 여러분이 보고 있는 모티프 (Motif) 인터페이스는 윈도 95 또는 OS/2에 비하여 구식이며 전혀 어필되지 않는다. 심하게는 윈도 3.1 시대의 망령을 보는 듯 하다고 말하는 사람도 있다. 리눅스/X 윈도서 GUI 프로그래밍을 하려는 사람들에게 있어서 자유롭게 느껴지기도 하지만 힘들게도 느껴지는 면은 바로 이것이 바로 리눅스/X 윈도의 모습이다라고 말할 만한 구체적이고 표준적인 인터페이스가 없다는 사실이다. 리눅스의 무정부적인 특성상 이런 모습으로 통일하는 강제성이 통하지 않는데다가 사실 이런데 관심을 갖기 시작한 것도 그리 오래되지는 않았기 때문이다. 요즘 들어서야 겨우 리눅스의 GUI화에 대하여 고민하는 사람들이 많아졌을 뿐 이다.

6.2.1 인터페이스에 대한 개선 고민

SVGA 그래픽 라이브러리와 X 윈도 시스템을 그래픽 환경으로 사용하는 리눅스는 서서히 자신이 사용하고 있는 그래픽 시스템에 대하여 회의를 느끼기 시작한 것 같다. 우선 X 윈도 서버는 그래픽 자원에 관한 한 무제한적인 권력을 행사하고 있고 리눅스 커널이 관여하지 못하고 있다. 가장 쉽게 볼 수 있는 현상으로는 X 윈도 시스템이 종료하면서 콘솔 화면을 완전히 망가뜨리고 어떤 때는 복구조차 할 수 없어서 리부팅하는 것이 편할 때

가 있었다. 또한 보안 상 X 서버가 setuid 바이너리 상태로 작동하므로 큰 문제가 아닐 수 없다. 이에 따라 리눅스에서 X 서버의 무제한 권력을 빼앗고 리눅스 커널에 그래픽 제어 권한을 돌려주는 프로젝트인 GGI가 나와 있고 소커의 성과를 가진 상태이다. 홈 페이지는 `synergy.foo.net/\~{}ggi` 여기서 재미있는 또 다른 펭귄을 만날 수 있다.

그 외에도 게임 SDK 개발 프로젝트 등이 있는데 그 성공 여부를 떠나서 결과물은 충분히 리눅스 공동체에 환원되리라 본다.

리눅스 인터페이스 개선(LIP)라는 것도 있다. 요즘은 많은 위젯들이 쏟아져 나오고 있으니 기대할 만도 하다. 홈 페이지는 `blank.pages.de/lip` LIP는 Xt 층을 사용하지 않고 직접 Xlib를 사용하여 구현한 리눅스 인터페이스를 위한 위젯세트이다. 여러분 중 도전할 생각을 가진 사람은 없는가?

리눅스에게 칙칙한 유닉스 터미널이라는 옷 이외에 정말로 산뜻한 새로운 옷 을 지어줄 재봉사는 없는가?"

6.2.2 공통의 데스크탑 환경?

리눅스/X 윈도 환경에는 수많은 윈도 관리자가 있다. 제일 먼저 사랑 받았던 것이 fvwm 류라면 그 다음은 애프터스텝과 같은 넥스트스텝 분위기의 관리자를 들 수 있고 완전히 윈도 95와 같은 흉내를 내주는 fvwm95라는 관리자도 있다. 여기에 KDE(K Desktop Environment)라는 것이 가세했는데 이것은 윈도 관리자 하나만이 아니라 데스크탑 환경을 구성하는 거의 모든 요소를 가지고 있다는데에서 놀라움을 금치 못한다.

설치해보면 알겠지만 여러분은 마치 윈도 95나 OS/2와 같은 환경에서 작업하고 있다는 착각이 들 정도이다. 윈도 95에서 볼 수 있는 많은 친숙한 요소들을 여기서 발견할 수 있으며 심지어 새로운 파일관리자인 리눅스 익스플로러까지 있다.

KDE 프로젝트의 관리자는 설정 방식, 파일을 두는 위치, 키보드 액셀레이터 사용법 등 일관된 환경을 갖추기 위해 노력하고 있다. 여러분이 KDE 프로젝트의 한 구성요소를 만들고 싶다면 먼저 'The Style Guide'라는 것을 읽어보아야 한다.

KDE는 트롤 테크(Troll Tech)라는 회사 `www.troll.no` 에서 내놓은 반공개 X 윈도 위젯인 Qt라는 것에 의거하고 있다. 그들이 내놓는 Qt 위젯은 X 윈도/윈도 95/NT 멀티플랫폼 용으로서 윈도 95/NT 환경에 대해선 라이선스료를 받는 반면 X 윈도 시스템에 대해서는 자유 소프트웨어를 만드는 데 사용할 때는 소스까지 무료로 제공한다. 하지만 여러분이 소스를 고칠 권한은 없다. KDE라는 훌륭한 데스크탑 통합 패키지가 있음에도 불구하고 많은 사람들이 반대하는 이유가 바로 GPL이 아니라는 점인 것 같다.

KDE에 대하여 레드햇이나 데비안과 같은 배포판 제작자들이 어떤 반응을 보이는지가 중요한 관건으로 보인다. 필자로서는 이 성과가 참 아쉬울 뿐이다 (한 가지 공통적인 것인 인터페이스에 관한 프로젝트들이 대부분 독일인들에 의해 주도되고 있다는 사실이다. 부럽게 느껴진다). 이 문제들에 관해서는 5월 21 23일 3일 동안 독일에서 열린 '제 4차 국제 리눅스 회의'에서 중점적으로 다뤄졌다.

6.2.3 삭막한 리눅스 로고?

리눅스에서 한 가지 활발한 작업을 들라면 로고 그리기 작업이 아닐까 한다. 약간의 그래픽 실력을 가진 사람들이라면 리눅스를 표현하는 재미있는 로고를 그려서 공헌하기를 좋아한다. 리눅스에 있어 모든 것은 사용자의 직접적인 선택에 달린 것이므로 그 많은 로고 중 몇몇은 많은 사랑을 누리고 있다. 단적인 예가 바로 펭귄 로고가 아닌가 한다.

중세의 삭막한 무기와 리눅스가 어떤 연관을 갖는지는 모르겠으나 Brian Hammond 씨가 생각하는 리눅스는 강력하고 거친 컴퓨팅 환경인 것 같다. 이번 호에서는 방화벽과 같은 힘든 주제를 다루었으니 그림을 보고 조금 쉬어 보는 것이 어떨까? 사이트는 다음과 같다. `www.wco.com/\~{}png/greg/greg\_lnxpics.html`

제 7 절 글을 마치면서

필자가 생각하는 것보다 리눅스는 전세계적으로 그리고 국내에서도 널리 알려진 것 같다. 네트워크 관련 종사

자나 관심을 갖는 사람이라면 앞으로 당분간 리눅스라는 이름을 듣지 않고는 살 수 없을 것이라고 본다. 외국의 경우 이미 많은 ISP들이 리눅스를 서버로 사용하고 있다는 사실만 봐도 성능면에서나 가격면에서 그리고 무엇보다도 '우리의 것'이라는 주체적인 면에서 성공하고 있다는 것을 알 수 있다.

리눅스에 대한 많은 입장이 존재하지만 몇 가지 당부하고픈 내용이 있다. 리눅스를 어느 정도 사용해왔고 남들에게서 컴퓨터 좀 할 수 있다는 이야기를 듣는 여러분이 다른 초보자들에게 리눅스를 소개할 때 '리눅스가 공짜다'라는 측면을 제일 먼저 강조하지는 말아주었으면 한다. 많은 사람들이 리눅스가 무료라는 관점을 너무 강조한 나머지 리눅스의 중요한 면들에 대해서 이해하는데 실패하기 때문이다. 리눅스에서 여러분이 얻을 수 있는 것은 정말로 많다. 그 이유는 바로 리눅스가 어떤 소프트웨어 회사의 프로그래머 인력보다도 비교할 수 없을 만큼 수많은 자발적인 해커들의 노력 속에서 커가고 있기 때문이다. 그들의 모든 지혜가 여러분 것이 된다. 하지만 이 모든 것이 저절로 주어지는 것은 아니며 그 어떤 환경에서보다 귀찮을 정도로 여러분 스스로의 노력을 필요로 한다는 점을 강조하고 싶다.

리눅서들 사이에 의무 개념이란 없다. 많은 초보자들은 다른 리눅서들이 자신의 질문에 항상 대답해야 할 의무가 있는 것처럼 다그치기 좋아한다. 그 이유를 이해할 수는 없으나 아마도 리눅스가 무료이므로 질문에 대한 답변도 당연히 무료(?)여야 한다고 생각하는 것 같다. 사실 리눅스에서 무료인 것은 없다. 가장 귀중한 여러분의 노력이 필요한 곳이다. 여러분이 나서서 먼저 찾고 해결하고 도와주려는 생각을 갖지 않으면 리눅스는 발전할 수 없는 컴퓨팅 환경이다.

마지막으로 리눅스는 무엇보다도 여러분에게 즐거움을 줄 수 있는 컴퓨팅 환경이다. 여러분이 생각하기에 도저히 재미도 없고 적성에 맞지 않다고 생각하면 리눅스에 대해 강박관념을 가질 필요없다. 리눅스는 무엇보다도 즐거운 창조 작업으로 만들어진 것이며 지금도 그러하고 앞으로도 그러할 것이다. 그리고 사용자인 여러분도 그러하길 바라면 개발자인 여러분도 그렇게 즐기길 바란다. 아무리 어려워 보이는 것일지라도 여러분이 즐길 마음의 준비만 되어 있다면 충분하다.