커널 2.4 Intro

정원영 suni00@kernel.pe.kr

v0.2, 2000년 8월 30일

개발커널 2.4.0-tset 버전을 컴파일하기위한 기본적인 배경지식과 일반적 사양의 PC에서의 컴파일 옵션 및 방법 그리고 커널 2.4 에서 무엇이 달라졌는지를 다룬다. (이 글을 쓰는 현재 가장 최신 개발 커널인 2.4.0-test7을 기준으로 이론적인면 보단 실제로 어떻게 해야하는지에 중점을 두었다.)

목 차

제 1 절 배경지식

1.1 커널이란?

커널이란 운영체제(Operating System)에서 가장 핵심적인 역할인 자원 (메모리, 프로세서등)을 관리하며 시스템이 원활히 돌아갈 수 있도록 제어해 준다. 현재 우리가 리눅스(OS)라고 하는것도 실제적으론 리눅스란 운영체제의 커널 이름을 말하는것인데 이것이 확장된 의미로 사용되어 현재의 '리눅스'란 OS를 지칭하게도 된것이다. 그러므로 Linux = kernel이란 등식이 성립한다.

1.2 커널은 어디서 구하나?

우선적으로 안정 버전과 가장 최신 개발버전을 원한다면 `<http://kernel.pe.kr>` 로 접속하면 된다. 그 이외에는 다음 미러 사이트에서 구할수 있다.

- 우리나라 : `<ftp://ftp.kr.kernel.org/pub/linux/kernel/>`
- 미국 : `<ftp://ftp.us.kernel.org/pub/linux/kernel/>`
- 일본 : `<ftp://ftp.jp.kernel.org/pub/linux/kernel/>`

1.3 최신 커널 버전은 어떻게 알아보나?

SHELL에서 finger @finger.kernel.org 명령으로 최신 커널버전을 알 수 있다. 이 문서를 작성하는 현재 최신버전은 다음과 같다.

- stable version : 2.2.16
- beta version : 2.4.0-test7
- prepatch version : 2.4.0-test8/pre1

```
커널은 stable version(안정버전)과 beta version(개발버전)으로 나누는데 이
두 버전의 차이는 minor number가 짝수이냐 홀수이냐로 구분한다.
(minor number란 커널 2.2.16이 있다면 가운데 수인 2가 minor number이고
제일 앞의 2는 major number 이다. 마찬가지로 2.3.99에선 major는 2이고 minor는
3이다.)
그렇다면 2.4.0-test 버전들은 minor number가 짝수인데 왜 개발버전이라고 하나?
그 이유는 2.3.99의 패치버전들이 이제 거의 2.4 안정버전에 가까워졌다는 의미로
Linus가 2.4.0-test라 이름 붙인것이다.
```

1.4 커널 컴파일 방법(커널 2.4.0-test7)

여러 가지 방법들이 있지만 저장된 커널옵션설정 파일이 있다면 oldconfig를 저장된 파일이 없다면 menuconfig를 추천한다.

1. make oldconfig 또는 make menucofig

   ```
   make oldconfig를 하기위해선 다음과 같은 작업이 필요하다.
    예전에 설정한 커널옵션이 kernel.config 란 이름으로 저장되어있다면
    이 파일을 새 버전의 커널이 있는 /usr/src/linux 디렉토리에 .config란 이름으로
    카피한다.
   ```

```
# cp kernel.config /usr/src/linux/.config
위와같이 한후 make oldconfig를 실행하면 예전 옵션을 그대로 적용시켜주며
혹시나 새로운 옵션이 있다면 화면에 보여주며 선택할 수 있도록 해준다.
```

2. make dep
3. make clean
4. make bzlilo

```
make bzlilo란 다음과 같은 명령들을 순차적으로 실행한것과 같다.
# make bzImage
# cp /usr/src/linux/arch/i386/boot/bzImage /vmlinuz
# /sbin/lilo
```

5. mv /vmlinuz /boot/vmlinuz-2.4.0-test7
6. mv /System.map /boot/System-2.4.0-test7
7. vi /etc/lilo.conf

```
일반적으로 다음과 같은 화면을 볼 수 있을것이다.
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
default=linux

image=/boot/vmlinuz-2.4.0-test7  ---> 새로 생성한 커널이미지를 linux란 레이블
        label=linux                   로 부팅할 수 있도록 하기위해
        read-only
        root=/dev/hda1

image=/boot/vmlinuz-2.2.16      ---> 예전 커널이 2.2.16이라고 가정했을때
        label=old                    old란 레이블로 정의한다.
        read-only
        root=/dev/hda1
```

8. /sbin/lilo

1.5 모듈 컴파일 방법

모듈이 생성되는 디렉토리는 '/lib/modules/2.4.0-test7'이고 만약 현재 커널의 모듈을 다시 생성하려고 한다면 현재 커널의 모듈 디렉토리를 지우고 다시 컴파일 하기 바란다. (커널 2.4.0-test 버전에서는 모듈의 디렉토리구조가 많이 바뀌었다.)

1. make modules
2. make modules_install

제 2 절 커널 2.4.0-test 버전

2.1 필요한 패키지

RedHat 기반 6.1이상의 배포판이라면 modutils만 업그레이드 하면 될것이다. (modutils를 업하는데 glibc버전이 낮다면 glibc도 업해야 한다. glibc는 로케일과 timezone, 여러 라이브러리를 포함해서 의존성 문제에 많은 영향을 끼치므로 업그레이드 하는데 주의를 필요로 한다. rpm 버전이 낮다면 rpm도 업해야 한다.) 2.4.0-test 커널을 사용하기위해서 다음 사항을 확인하라.

```
        패키지 명            버전                 확인방법
    o  Gnu C              2.7.2.3            # gcc --version
    o  Gnu make           3.77               # make --version
    o  binutils           2.9.1.0.22         # ld -v
    o  util-linux         2.10o              # kbdrate -v
    o  modutils           2.3.13             # insmod -V
    o  e2fsprogs          1.18               # /sbin/tune2fs --version
    o  pcmcia-cs          3.1.19             # cardmgr -V
    o  PPP                2.4.0              # pppd --version
    o  isdn4k-utils       3.1beta7           # isdnctrl 2>&1|grep version
```

2.2 커널 2.4에서 달라진것과 새로운 것들

- Bogo Mips가 두 배로 올라간다.(AMD-K6 와 확인 못한 Processor 제외)

```
$ cat /proc/cpuinfo 를 해보면 확인할 수 있다.
Bogo(bogos : 가짜)이므로 신뢰할순 없지만 기분만은 좋지않은가?
```

- modutils 업그레이드시 바뀌는 것들

```
만약 modutils 버전이 앞에서 명시한 버전보다 낮은데 모듈 컴파일을 한다면
에러가 난다. 꼭 modutils를 업 해야 모듈 컴파일을 할 수 있다.
modutils를 업하면 예전의 /etc/conf.modules가 /etc/modules.conf로 바뀌고
USB 모듈도 자동로딩이 가능하며 많은 예약어를 지원한다.

다음 ftp 사이트에서 구할수 있다.
ftp://ftp.kernel.org/pub/linux/utils/kernel/modutils/v2.3/
```

- 모듈이 위치한 /lib/modules/2.4.0-test7 디렉토리 구조

```
커널 2.4.0-test6 이상의 버전부터 모듈들이 위치하는 '/lib/modules/커널버전' 의
디렉토리 구조가 더 세분화 되고 다음과 같은 형식으로 바뀌었다.
현재 필자의 디렉토리 구조이다.

[kernel@suni00/lib/modules/2.4.0-test7]$ ls -l

lrwxrwxrwx  1 root  root    26 Sep  4 17:09 build -> /usr/src/linux-2.4.0-test7/
drwxr-xr-x  6 root  root  1024 Sep  4 17:09 kernel/
-rw-r--r--  1 root  root  1340 Sep  5 20:33 modules.dep
-rw-r--r--  1 root  root  3235 Sep  5 20:33 modules.pcimap
drwxr-xr-x  2 root  root  1024 Sep  4 17:09 pcmcia/
```

위의 kernel 디렉토리에 들어가보면 다음과 같이 세분화 되어있다.

```
[kernel@suni00/lib/modules/2.4.0-test7/kernel]$ ls -l

drwxr-xr-x   3 root     root         1024 Sep  4 17:09 arch/
drwxr-xr-x   8 root     root         1024 Sep  4 17:10 drivers/
drwxr-xr-x   9 root     root         1024 Sep  4 17:09 fs/
drwxr-xr-x   4 root     root         1024 Sep  4 17:09 net/
```

drivers 디렉토리 구조를 보면 예전 모듈들의 디렉토리 구조가 여기로 옮겨 왔음을 알 수 있다.

```
[kernel@suni00/lib/modules/2.4.0-test7/kernel/drivers]$ ls -l

drwxr-xr-x   2 root     root         1024 Sep  4 17:09 block/
drwxr-xr-x   3 root     root         1024 Sep  4 17:09 char/
drwxr-xr-x   2 root     root         1024 Sep  4 17:09 ide/
drwxr-xr-x   2 root     root         1024 Sep  4 17:09 net/
drwxr-xr-x   2 root     root         1024 Sep  4 17:09 sound/
drwxr-xr-x   2 root     root         1024 Sep  4 17:10 video/
```

- UID,GID bit가 16bit에서 32bit로 바뀌었다.

  2^16은 65536이므로 사용자를 65536명 까지 만들수 있지만 커널 2.4에서는 2^32 즉 4294967296명을 만들수 있다.(시스템 콜의 변경도 필요로 하며 현실적으로 이렇게 많은 유저를 한 서버에 둔다는것은 무리이다.)

- 생성가능한 프로세스의 수의 제한이 없어졌다.

  2.2.x 커널의 소스를 보면 include/linux/tasks.h에서 NR_TASKS의 개수가 512로 정의되어 있다. 512이상의 프로세스를 생성할 수 있도록 하려면 이 값을 고쳐 주어야 했는데 x86 machine에서는 GDT(Global Descriptor Table)의 제한때문에 4092까지 밖에 할 수 없도록 되어있었다.
  2.4.0-test 버전에서는 task 자료구조가 동적으로 할당되어 NR_TASKS 값이 없어졌고 메모리의 한계만이 유일한 제약으로 따른다.

- 공유메모리를 사용하기위해 /var/shm(또는 /dev/shm)을 마운트 해야한다.

  커널 2.3 개발버전을 처음 컴파일하고 부팅했을때 제일 처음 직면했던 문제인데 System V 공유메모리를 사용하지 못해서 부팅시 자동으로 뜨는 아파치 데몬이 뜨지 못하는 문제와 여러 프로그램을 실행했을때 공유메모리를 사용하지 못해 에러가 나는 문제가 발생했었다. 2.4-test 버전을 쓰기위해선 꼭 해줘야 하는 부분이다.

```
# mkdir /var/shm

# vi /etc/fstab
none            /var/shm          shm         defaults        0 0
```

- ipchains가 iptables로 바뀌었다.

  menuconfig 옵션에서 Networking options ---> [*] Network packet filtering (replaces ipchains) 를 선택하고 IP: Netfilter Configuration ---> <M> ipchains (2.2-style) support 를 선택하면 커널 2.2.x 와 같은 방식으로 쓸 수 있다.

- Magic SysRq key 의 변화

```
Magic SysRq key란 시스템의 제어가 불가능한 상태(일반적으로 '다운'되었다고 한다.)
에서도 제어를 가능하게 해주므로 커널 컴파일시 Kernel hacking --->
[*] Magic SysRq key를 체크해주자.

2.4.0-test 개발 커널에서 Magic SysRq key를 사용할려면 다음과 같이
/proc/sys/kernel/sysrq 값을 1로 만들어야 한다.

# echo 1 > /proc/sys/kernel/sysrq

키 조합
     Alt+SysRq+R      use Raw keyboard events
     Alt+SysRq+K      kill current VT in use
     Alt+SysRq+E      tErminate all running processes (except init)
     Alt+SysRq+I      kIll all processes (except init)
     Alt+SysRq+L      kilL all processes (including init)
     Alt+SysRq+B      reBoot
     Alt+SysRq+S      Sync all drives
     Alt+SysRq+U      Umount all filesystems
     Alt+SysRq+O      turn the machine Off
     Alt+SysRq+P      dump Processor's registers
     Alt+SysRq+T      dump current Tasks
     Alt+SysRq+M      dump some Memory info
     Alt+SysRq+[0-9]  set kernel log level and redirect it to console

     x86 머신에서는 Alt를 누른상태에서 Print Screen(Sys Rq)키를 누르고
     각 알파벳키를 누르면 적용된다.

만약 시스템이 다운 되었다면 같은 다음과 순서의 조합으로 시스템에 손상을
주지않고 안전하게 재부팅 할 수 있을것이다.

Alt+SysRq+S : 버퍼에있는 것들을 디스크로 sync(동기화)시킨다.
Alt+SysRq+E : init 프로세스를 제외한 모든 프로세스를 종료시킨다.
Alt+SysRq+U : 파일 시스템을 unmount 시킨다.
Alt+SysRq+B : 시스템 재부팅
```

- Codepage 949 (UnifiedHangul) 지원(커널 2.4.0-test5부터 지원하며 안정버전은 2.2.16에서 지원한다.)

```
리눅스에서 윈도 파티션(vfat)을 마운트 했을때 한글이 '????'이런식으로
보이는 문제를 해결해 준다.
```

- Devfs(Device Filesystem)

```
devfs란 /proc와 같은 가상 파일시스템으로 Richard Gooch에 의해 작업되었다.
devfs를 사용할려면 다음과 같은 과정을 거쳐야하며 devfs를 쓰지 않아도 2.4.0-test
커널을 사용하는데 상관은 없다.
devfs를 사용하고자 한다면 devfs는 메모리상에 존재하는 가상 파일시스템이므로 만약
/dev 에 특수한 목적의 디바이스 파일 또는 디렉토리를 생성했다면 재부팅시 /dev
디렉토리에 만들었던것이 사라질것이므로 재 작성해줘야 한다.
부팅할때마다 mknod 명령으로 필요한 디바이스를 다시 만드는것 보단 셸 스크립트를
작성하는것이 편리할 것이다.
```

예를들어 X-4.0이상의 NVdirver를 쓴다면 /dev 디렉토리에 nvidia0, nvidia1, nvidia2, nvidia3, nvidiactl과 같은 캐릭터 디바이스 파일이 만들어져 있어야 X를 띄울수 있으므로 부팅시 만들어질수 있도록 셸 스크립트를 작성한다면 약간의 수고를 덜 수 있을것이다.
다음과 같은 방법으로 devfs를 사용할 수 있다.

1. 커널 컴파일 옵션에서 다음 사항을 체크한다.
```
Code maturity level options  --->
    [*] Prompt for development and/or incomplete code/drivers

File systems  --->
    [*] /dev file system support (EXPERIMENTAL)
    [*]   Automatically mount at boot
    [*]   Debug devfs
```

2. devfsd 소스 파일을 구한다.
http://www.atnf.csiro.au/~rgooch/linux/ 에서 devfsd-v1.3.10.tar.gz 이상의 버전을 다운 받아 다음 작업을 한다.
```
# tar xvvzf devfsd-v1.3.10.tar.gz     -> 적당한 디렉토리에 압축을 푼다.
# make                                -> 컴파일해서 devfsd란 파일을 얻는다.
# mv devfsd /sbin/                    -> /sbin 디렉토리로 옮긴다.
# cp modules.devfs /etc/              -> modules.devfs와 devfsd.conf를
# cp devfsd.conf /etc/                   /etc 디렉토리로 옮긴다.
```

3. /etc/rc.d/rc.sysinit 스크립트에 다음 부분을 추가한다.
```
# Mount /proc (done here so volume labels can work with fsck)
action "Starting devfsd" /sbin/devfsd /dev          <--- 이부분을 추가한다.
action "Mounting proc filesystem" mount -n -t proc /proc /proc
```

4. /etc/securetty 파일을 다음과 같이 수정한다.
```
1
2
3
4                           # tty1 이런식으로 되어있는것 모두 tty를 뺀
5                           # 숫자부분만 남도록 해준다.
6
7
8
```

5. /etc/security/console.perms 에서 다음과 같이 바꿔준다.
```
<console>=tty[0-9][0-9]* :[0-9]\.[0-9] :[0-9] 을 주석처리하던지 삭제하고
<console>=tty[0-9][0-9]* [0-9][0-9]* :[0-9]\.[0-9] :[0-9] 으로 바꿔준다.
```

6. /etc/lilo.conf 에 다음을 추가
```
append = "devfs=mount"
```

```
devfs를 쓰지 않겠다면 append = "devfs=nomount" 하면 된다.
```

2.3 커널 옵션 설정

일반적인 데스크탑 PC사양(1 CPU, No SCSI)에서 필요로 하는 커널옵션들을 설명하며 그다지 필요성이 없는 옵션들은 제외시켰다.

Code maturity level options —>

* Prompt for development and/or incomplete code/drivers - *개발 수준의 옵션들도 선택가능하게 해준다.*

Loadable module support —>

* Enable loadable module support - *모듈을 사용할수 있게 해줌.*

* Kernel module loader - *커널이 알아서 모듈을 올려준다.*

Processor type and features —>

- (Pentium-Pro/Celeron/Pentium-II) Processor family
  - (X) Pentium-Pro/Celeron/Pentium-II - *테스트 PC가 Pentium II이므로 ... 시스템에 맞는걸 선택한다.*

* MTRR (Memory Type Range Register) support - *프로세서가 메모리 영역 접근을 제어할수 있음. 그래픽의 쓰기 속도 향상*

General setup —>

* Networking support - *네트워킹 지원*

* PCI support - *PCI 지원*

- (Any) PCI access mode - *커널이 직접 액세스 시도*

* System V IPC - *System V IPC를 지원하게 함, Shared Memory도 여기서 지원한다.*

* BSD Process Accounting - *프로세스 정보를 파일에 저장*

* Sysctl support - *특정 커널의 파라미터와 변수들을 동적으로 변경시킬수 있도록 함*

- (ELF) Kernel core (/proc/kcore) format - *ELF core 포맷*
- <*> Kernel support for a.out binaries - *a.out 바이너리 지원*
- <*> Kernel support for ELF binaries - *ELF 바이너리 지원*

Plug and Play configuration —>

- <*> Plug and Play support - *Plug and Play 지원*

Block devices —>

- <*> Normal PC floppy disk support - *플로피 드라이버 장치 지원*

- <M> Loopback device support - *한 파일을 하나의 파일 시스템처럼 인식 시킴*

Networking options —>

- <*> Packet socket - *네트웍 디바이스와의 직접통신을 하게 해준다.*
  - <*> Packet socket: mmapped IO - *더 빠른 통신을 할 수있게 한다.*
- <*> Unix domain sockets

* TCP/IP networking - *TCP/IP 네트워킹 지원*
  * IP: TCP syncookie support (disabled per default) - *서비스 거부 공격을 받을때 대처해줌*

ATA/IDE/MFM/RLL support —>

- <*> ATA/IDE/MFM/RLL support
- IDE, ATA and ATAPI Block devices —>
  - <*> Enhanced IDE/MFM/RLL disk/cdrom/tape/floppy support - *IDE를 사용할수 있도록 해준다.*
    - <*> Include IDE/ATA-2 DISK support - *하드디스크 사용가능하게 함*
    - <*> Include IDE/ATAPI CDROM support - *CD-ROM 사용가능하게 함*
    - <M> Include IDE/ATAPI FLOPPY support - *ZIP 드라이브등을 사용가능하게 함*
    - — IDE chipset support/bugfixes
    - <*> Generic PCI IDE chipset support - *PCI 방식의 IDE 지원*
      - <*> Sharing PCI IDE interrupts support - *IRQ 공유 지원*
      - <*> Generic PCI bus-master DMA support - *DMA 지원*
      - <*> Use PCI DMA by default when available - *VIA VP2 칩셋을 사용한다면 절대 선택하지 말것*
      - <*> HPT366 chipset support - *Ultra DMA 66 지원*

Network device support —>

* Network device support

- Ethernet (10 or 100Mbit) —>
  * Ethernet (10 or 100Mbit) - *자신의 시스템에 맞는걸 선택* (RealTek 8139등을 쓴다면 [*]EISA, VLB, PCI and on board controllers를 선택하면 하부 목록이 나온다.)

Character devices —>

* Virtual terminal - *가상 터미널 지원*
  * Support for console on virtual terminal - *가상터미널을 시스템 콘솔로 쓸수있게 해준다.*
* Unix98 PTY support - *Unix98 가상터미널 지원*
  * (256) Maximum number of Unix98 PTYs in use (0-2048) *Unix98 PTY 개수의 최대값*

- Mice —>

- <*> Mouse Support (not serial and bus mice)
    - * PS/2 mouse (aka "auxiliary device") support - *PS/2 마우스 지원*
- <M> /dev/agpgart (AGP Support) - *AGP 지원 (하부메뉴중 자신의 보드에 맞는거 선택)*

File systems —>

- <*> Kernel automounter support - *원격 파일 시스템을 자동으로 마운트 해준다.(NFS등을 사용할 때) N을선택해도 상관없다.*
- <M> DOS FAT fs support - *FAT 기반의 파일 시스템 지원*
    - <M> MSDOS fs support - *MSDOS 파티션 마운트 지원*
    - <M> VFAT (Windows-95) fs support - *윈도즈 파티션 지원*
- <M> ISO 9660 CDROM file system support - *CD-ROM 파일 시스템 지원*
    - * Microsoft Joliet CDROM extensions - *Joliet CDROM을 읽을 수 있게 한다.*
- * /proc file system support - *프로세스를 위한 가상 파일 시스템*
- * /dev/pts file system for Unix98 PTYs - *위에서 [*] Unix98 PTY support 를 선택했다면 선택해야 한다.*
- <*> Second extended fs support - *현재 리눅스 파일 시스템*
- Network File Systems —>
    - <M> NFS file system support - *네트웍 파일 시스템 지원*
    - <M> SMB file system support (to mount Windows shares etc.) - *윈도즈의 네트웍 기능들을 공유할수 있게 해준다.*
- Native Language Support —>
    - Default NLS Option: "cp949" - *디폴트로 한글이 선택되도록 한다.*
    - <*> Codepage 437 (United States, Canada)
    - <*> Codepage 949 (UnifiedHangul) - *윈도파티션을 마운트 했을때 한글을 볼수있도록 해준다.*
    - <*> NLS ISO 8859-1 (Latin 1; Western European Languages)

Console drivers —>

- * VGA text console - *VGA 표준 디스플레이를 통해 텍스트 모드에서 사용가능하게 한다.*

Sound —> *시스템에 맞는걸 선택한다*

Kernel hacking —>

- * Magic SysRq key - *시스템이 다운되더라도 제어할 수있도록 해준다.*

2.4 테스트 PC 사양

커널 컴파일을 할려면 어느정도 자신의 시스템에 관한 하드웨어적 지식이 있어야 한다. 그래픽 카드, 비디오 카드, 사운드 카드, 네트웍 카드등은 커널 컴파일시 각각에 맞는것을 선택하고 앞의 옵션을 참고한다면 무리없이 컴파일 가능할것이다.

- CPU : Intel Pentium II 350 (Deschutes)
- Memory : 128MB
- Main Board : MS-6119 점퍼

  ```
  MS-6119 Award Bios 버전이 2.9이상이 아니라면 메모리를 64MB 밖에
  인식하지 못하는 문제가 있으므로 바이오스를 업데이트 하던지
  /etc/lilo.conf에 append="mem=128M" 이렇게 추가해야 한다.
  ```

- Hard Disk : QUANTUM FIREBALL 8.4GB
- CD-ROM : SAMSUNG 40x
- Network : 3Com PCI 3c905b
- Video : RIVA TNT (16MB)
- Sound : SoundBlaster 64PCI (es1370으로 잡으면 된다.)

제 3 절 그 밖의 것들

다음은 커널 프로젝트 홈으로 문의한 질문들중 몇가지를 추려보았다. 정말 잡다한 것이라 소개하기 부끄러운것도 있고 알아두면 유용한 것도있으니 참고하기 바란다.

3.1 커널 소스의 크기는 얼마나 되나?

초기 리눅스 커널 0.01의 크기가 470KB밖에 되지 않았는데 현재 최신 개발 커널 2.4.0-test5의 크기는 103MB나 된다. 놀랍지 않은가?

3.2 왜 커널 이미지를 압축하는가?

우리는 커널 컴파일을 할 때 make zImage, make bzImage 이런 식으로 커널 이미지를 압축한다.(make zlilo, zdisk, bzlilo 모두 마찬가지...) 아무런 의문 없이 무의식적으로 당연히 이렇게 쓰고 있다.

여기에는 약간의 배경지식이 필요한데 간략히 설명하겠다.

우리가 흔히 아는 매킨토시의 M68계열 processor는 8bit환경에서 32bit환경으로 발전되었지만 인텔의 8086계열은 DOS의 대중성 때문에 바로 32bit 환경으로 가지 못하고 16bit 환경을 가지게 되었다. 대중적으로 많이 쓰이고 있는 DOS를 계속 쓸 수 있도록 하기 위해서였다. 이러한 이유 때문에 리얼모드, 보호모드, 가상86모드가 생겼다.

리얼모드, 보호모드, 가상86모드 모두 세그먼트레지스터와 오프셋 레지스터를 이용하여 주소를 지정하는데 이들 레지스터의 사용방법이 다르다.

리얼모드는 세그먼트x10h + 오프셋으로 주소를 만드는데 16bit에서 세그먼트와 오프셋의 최대 값은 FFFF이다. 그러므로 최대로 지정할수 있는 주소는 FFFF0 + FFFF가 된다. 이걸 계산하면 1MB + 64KB가 된다.

보호모드에선 주소지정 방식이 리얼모드와는 많이 다르며 페이징 등을 이용하여 32bit 모두 주소 값을 만들 수 있으므로 이론적으론 4GB의 메모리를 이용할 수있으며 가상86모드 또한 선형주소를 만들어내는 과정만 다르므로 기본 매커니즘은 보호모드와 같다. (보호모드와 가상86모드의 주소 생성 법은 생략한다.)

프로그램은 code 부분과 data부분으로 나누어지는데 리얼모드에서 프로그램의 code부분은 반드시 위 최대 주소 지정영역 내에 있어야 한다는 것이다. 리얼모드에서는 여러 개의 프로그램이 메모리에 올라와서 수행될 수 없으며 (만약 여러 개의 프로그램이 메모리에서 수행된다면 다른 프로그램영역을 침범 할 수 있기 때문이다.) interrupt를 이용하는 RAM상주 프로그램만이 메모리를 같이 차지할 수 있다.

위에서 설명한 리얼모드의 약 1MB정도의 영역 중 Coventional Memory가 640KB를 차지하고 나머지는 비디오램 이나 기타 디바이스가 차지한다.

그러므로 커널 이미지는 Coventional Memory 즉 640KB내에 들어가야 하므로 커널 이미지의 크기가 640KB 보나 작아야 한다. 커널 이미지의 압축으로 이러한 제약을 부분적으로 극복하였으나 640KB의 일부는 여러 가지 버퍼(DMA buffer)나 특정 주소가 시스템에 예약되어 있으므로 640KB보다 더 작은 크기의 커널이 요구된다. 이러한 문제의 대안으로 커널을 Extended Memory에 적재하는 방법을 생각해 볼 수 있다. Extended Memory 영역에 자유로이 적재하기 위해서는 보호모드를 사용해야하는데 보호모드에서는 BIOS와 같이 시스템이 완전히 준비되기 전의 기본적인 기능들을 사용할 수 없다. 이럴 경우 디스크를 액세스하는 자체적인 함수를 준비하여 커널을 Extended Memory 에 적재하거나 아니면 커널을 줄이는 수밖에 없는 것이다.

3.3 LILO에서 최대 커널 이미지 개수

여기에 대해선 전혀 모르고 있었는데 어느 날 2.4.0-test1-ac18 버전을 더더욱 최적화하기 위해 커널 컴파일을 하고 lilo.conf에 이미지를 등록시키고 lilo란 명령으로 커널 이미지를 write하는데 다음과 같은 메세지가 나왔다.

```
Only 19 image names can be defined
```

그때까지 나의 lilo.conf에는 19개의 커널 이미지가 정의되어있었는데 하나 더 추가해서 20개 가 되자 위와 같은 메세지를 출력한 것이었다. 커널 이미지를 20개 이상 등록(?)하지 않았다면 아마 몰랐을 것이다.

3.4 스왑파일도 있는데 스왑 파티션을 쓰는이유?

결론적으론 속도 때문이다. 스왑파일을 쓴다면 파일에 접근하기위해 VFS와 파일 시스템 드라이버 그리고 파일정보를 담고있는 여러 가지를 읽어오고 쓸때도 일반 파일이 writing 되는 경로를 거칠 것이다. 스왑 파티션에서는 위의 과정없이 특정 블럭에 바로 접급할 수 있으므로 불 필요한 오버헤드가 발생하지 않아 속도가 더 빠르기 때문이다.

3.5 서브 디렉토리를 60000개 이상 만들고 싶다.

현재 리눅스의 기본 파일 시스템은 ext2이다. (SuSE는 Reiser 파일시스템이다.) ext2란 Second Extended File System의 약어이며 초기 리눅스 파일 시스템이었던 MINIX에서는 최대 파일이름 14문자, 최대 파티션 64MB밖에 되지 않았고 이런 여러 가지 문제로 파일 시스템이 오늘날의 ext2까지 발전하였다. ext2에서는 255문자의 파일명, 최대 2GB의 파일, 4TB의 디스크 용량을 지원하며 서브디렉토리는 테스트해본결과 32000개까지만 만들어졌다. 다음을 참고하라.

```
#!/bin/sh
count=70000
i=1
while [ $i -lt $count ]
do
```

```
  mkdir $i && echo "$i"
  i=$(($i+1))
done
```

위는 서브 디렉토리를 만들어주는 shell script이며 서브 디렉토리 이름은 1번부터 count로 지정한 값만큼 만들어 준다.

- ext2 파일 시스템에서의 서브 디렉토리 생성

```
커널 버전 : 2.2.14-5.0

$ ls -l
drwxrwxr-x  32000 suni00   suni00     475136  7월 12 15:41 tmp/

$du -sh
128M    .

디렉토리 링크의 inode 때문에 ./와 ../를 포함하여 32000 까지 밖에 생성되지
않는걸 알 수 있다.
또 ext2의 블럭 사이즈는 4KB이므로 생성디렉토리들의 총 용량이 128MB나
차지하였습니다.
```

- Reiser 파일 시스템에서의 서브 디렉토리 생성

```
커널 버전 : 2.2.16과 2.4.0-test1-ac18

$ls -l
drwxrwxr-x 64535 suni00   suni00    1548888 Jul 12 15:48 tmp/

$du -sh
30M    .

위에서 보다시피 64535개가 생성되었고 디렉토리들의 용량은 ext2보다 훨씬 적은
30MB 밖에 차지하지 않는걸 알 수 있다.
(Reiser 파일 시스템에서의 빈 디렉토리의 사이즈는 48Byte를 차지했다.)
```

디렉토리 링크가 둘 다 2Byte 범위까지 표현할 수 있는 것 같은데 파일 시스템에 따라 위와 같은 차이가 났다.

3.6 Reiser 파일시스템 만들기

현재 리눅스의 기본 파일 시스템인 ext2는 신뢰성, 충돌 복구 등의 문제가 있는데 이런 부분들을 해결 할 수 있는 것이 저널링 파일 시스템이다. Reiser 파일 시스템 또한 이런 저널링 파일 시스템의 일종으로 이미 SuSE에서는 기본 파일 시스템으로 Reiser를 선택하고 있다. 머지않아 리눅스의 기본 파일 시스템도 저널링 파일 시스템으로 바뀔 것이다.

1. 패치파일 받아오기

<http://devlinux.com/projects/reiserfs/> 또는 <http://kernel.pe.kr/data.php> 에서 자신의 버전에 맞는 패치를 받아온다.

2. 패치를 적용시킨다.(/usr/src/linux 에서 패치한다.)

```
# patch -p1 < 패치파일명 (압축을 풀었을때)
또는
# gzip -cd 패치파일명.gz | patch -p1
```

3. 커널옵션 선택

```
Code maturity level options  --->
   [*] Prompt for development and/or incomplete code/drivers 를 선택하고
File systems  ---> <*> Reiserfs support 선택한다.
```

4. 커널 컴파일및 모듈 컴파일, lilo 정보도 고쳐준다.

5. ReiserFS 유틸 컴파일(mkreiserfs을 만들기위해)

```
# cd /usr/src/linux/fs/reiserfs/utils/
# make
# make install
```

6. ReiserFS가 적용된 커널 이미지로 재부팅한다.

7. ReiserFS 만들기

```
(/dev/hdb1 에 만든다고 가정하며 기존 ext2 파일시스템의
 데이터는 지워지므로 백업하던지 아님 새 하드를 달았다고 가정한다.)

# umount /dev/hdb1 (mount 되어있지 않다면 이건 필요없다.)
# mkreiserfs /dev/hdb1
# mount /dev/hdb1 /TEST(mount 시킬 디렉토리는 각자 알아서..)
```

8. /etc/fstab 수정하기

```
/dev/hdb1            /TEST             reiserfs    defaults        1 2
ext2를 reiserfs로 바꾸면 된다.
```

9. 확인하기

```
$ mount
...
/dev/hdb1 on /TEST type reiserfs (rw)
...
위와 같이 reiserfs 라고 나오면 성공
```

Reiser 파일 시스템을 만들고 마운트 시키면 기본으로 32MB의 용량을 잡아먹는다. 이 공간에다가 자체적인 정보를 기록하므로 Reiser 파일 시스템을 위해선 최소 32MB이상의 파티션을 필요로 한다.(나의 /boot 파티션은 16MB여서 reiserfs를 적용시키지 못하였다.)

3.7 커널 2.4.0-test버전에서 X 4.0을 쓰고싶다.(RIVA TNT)

현재 nVidia에서 나오는 X 4.0 드라이버들은 커널 2.2.x의 안정버전에만 적용된다. 개발버전에 적용시킬려면 <http://www.darkrock.co.uk/nv/>에서 NVIDIA_kernel-0.9-2.3.x.tar.gz를 받아 컴파일 해야한다.(테스트해본 결과 커널 2.4.0-test1-ac18까지만 적용된다.)

3.8 커널 버전을 어떻게 바꾸는가?

어떤 학교의 전산관련 과목 리포트인지 이런 질문을 많이 받았다. 커널 버전만을 가짜로 바꿀려는 이유를 모르겠다.

```
# vi /usr/src/linux/Makefile

        VERSION = 2
        PATCHLEVEL = 4
        SUBLEVEL = 0
        EXTRAVERSION = -test1-ac18


# vi /usr/src/linux/include/linux/version.h

        #define UTS_RELEASE "2.4.0-test1-ac18"
        #define LINUX_VERSION_CODE 132096
        #define KERNEL_VERSION(a,b,c) (((a) << 16) + ((b) << 8) + (c))

위의 Makefile에서 버전 정보를 고쳤다면 컴파일할때 version.h는 자동으로 바뀐다.
```

이 두 파일들을 원하는 버전으로 바꾸고 컴파일 하면 된다. 잘못 바꾸어서 커널 버전이 맞지 않을 경우 System.map에서 버전이 틀리다고 메세지를 보낼 것이다. 그럴 경우 해당 커널 이미지의 System.map을 없애버려도 무관하다. (System.map이 없다면 어디서 문제가 발생했는지 알 수 없을 것이다.)

3.9 CPU의 정보를 바꾸고 싶다, Bogo Mips를 높이고 싶다

이건 그냥 내가 하고 싶어서 해봤다.

```
# vi /usr/src/linux/arch/i386/kernel/setup.c
        $ cat /proc/cpuinfo 정보를 고쳐준다.

# vi /usr/src/linux/init/main.c
        $ dmesg 의 보고 밉스를 고쳐준다.
```

위 두 파일을 원하는 대로 고치면 된다. 약간의 C 프로그래밍만 할 수 있다면 쉽게 원하는대로 바꿀 수 있을것이다.

3.10 KERNEL32.DLL에서 잘못된 연산수행!?

위와 같은 질문을 하시는 분들이 가끔 계신데 MS-WINDOWS 커널을 어떻게 손볼 수 있겠는가? (검색엔진에서 kernel로 검색해서 커널 프로젝트 홈으로 들어왔다더군요.) 성의는 대단하나 단지 손상 안된 다른 KERNEL32.DLL을 카피하라고 할 수밖에...