리눅스 키보드, 콘솔 하우투

Andries Brouwer, aeb@cwi.nl

v2.8, 25 February 1998

노 한진, ffnhj@mail.hitel.net

이 문서에는 리눅스에서 사용하는 키보드와 콘솔에 대한 정보와 아스키 이외의 문자를 사용하는 방법에 대하여 설명되어 있다. 이것은 리눅스 2.0에 대하여 적용되는 내용이다.

목 차

제 1 절 유용한 프로그램

아래에 나열된 패키지는 키보드, 콘솔과 관계된 프로그램들이다.

kbd-0.95.tar.gz contains loadkeys, dumpkeys, showkey, setmetamode, setleds, setfont, showfont, mapscrn, kbd_mode, loadunimap, chvt, resizecons, deallocvt, getkeycodes, setkeycodes.

util-linux-2.6에는 setterm, kbdrate가 있다. (알다시피 util-linux-2.6에 있는 more는 이름이 충돌하는 관계로 코어 덤프한다. 옛 버전을 보존해 두거나 util-linux-2.5를 사용하든지, more.c 내의 'savetty'를 'my_savetty'로 바꾸면 된다..)

sh-utils-1.12에는 stty가 있다.

open-1.4.tgz에는 open이 있다. (이것은 openvt로 이름을 바꾸어야 한다). (dynamic-vc-1.1.tar.gz를 참조할 것.)

SVGATextMode-1.8.tar.gz에는 SVGATextMode가 있는데 현재는 이것이 resizecons를 대신한다.

X 배포본에는 xmodmap, xset, kbd_mode가 있다. (XFree86 1.3에서 생기는 일은 X386keybd(1)를 참조하고 X11R6의 XKEYBORAD extension에 대해서는 Xserver(1)를 볼 것.)

termcap-2.0.8.tar.gz에는 termcap이 있는데, 이것은 구형 터미널 사용을 위한 데이타베이스이다. ncurses-1.9.9e.tar.gz에는 termlib라는 데이타베이스가 있는데 이것은 옛날 쓰이던 termcap을 대신한다. (하지만 아직도 termcap을 쓰는 많은 프로그램이 있다.)

X 환경이 아닌 상태에서 키보드의 환경 설정과 키를 누를 때 발생하는 코드에 대해서는 loadkeys(1), setleds(1), setmetamode(1)를 참조할 것. X 환경에서는 xmodmap(1), xset(1)를 참조하라.

콘솔 폰트 로딩에 대해서는 setfont(8)를 참조하라. 리눅스에서 디폴트 폰트는 비디오 카드에 있는 하드웨어 폰트이고, 종종 액센트 기호나 Latin-1 기호가 없는 'Code Page 437'로 설정되어 있어서 iso01.f16 같은 폰트를 로딩하고 싶어하는 사람들이 많다.

X 환경이 아닌 곳에서 글자색이나 배경색, 스크린 정리, 글자 갱신율 등과 같은 속성에 대해서는 setterm(1), kbdrate(8)를 참조하면 된다. X 환경에서는 xset(1)를 참조하라. 여기에는 키 클릭과 벨소리 크기에 대한 정보도 있다.

/etc/termcap 화일에는 콘솔(또는 터미널 등)에 입출력을 하는 많은 프로그램들이 사용하는 이스케이프 시퀀스가 정의되어 있다. termcap(5)을 참조하라. /usr/lib/terminfo에 좀 더 최신의 버전이 있다. terminfo(5)를 권한다. Terminfo 파일들은 terminfo 컴파일러인 /usr/lib/terminfo/tic으로 컴파일된다, tic(1)을 참조하라. 그 내용은 infocmp 프로그램으로 검사할 수 있으니, infocmp(1)를 볼 것. 리눅스 콘솔 시퀀스는 console_codes(4)에 문서화되어 있다.

제 2 절 키보드 개요

사용자가 키보드를 누르면 키보드 콘트롤러는 커널 키보드 드라이버로 스캔코드를 보내게 된다. 코드값을 바꿀 수 있는 키보드도 있지만 보통은 각 키에 고정되어 할당된 스캔코드가 있다. X가 실행되어 있는 경우처럼 시스템이 *스캔모드*에 있을 때는 커널 키보드 드라이버가 하는 일이라곤 그냥 스캔코드를 전달받아 어플리케이션 프로그램에 전달해주는 것 뿐이다. 스캔모드가 아닌 경우에는 키가 눌리거나 풀리는 각각의 이벤트에 따라 스캔코드의 조합을 분석하여 키코드를 만든다. (하나의 키가 눌려도 최대 6개의 스캔코드가 발생할 수 있다.) 이렇게 만들어진 키코드가 (예를 들어 showkey와 같은 프로그램에서 사용되는) *키코드 모드*에서는 어플리케이션 프로그램에 전달된다. 키코드 모드도 아닌 경우에는 이 키코드를 keymap에서 찾아보고, 여기서 발견된 문자나 문자열을 어플리케이션에 전달하거나 해당되는 동작을 수행한다. (예를 들어보자. a 키를 눌렀다 떼면 키보드는 스캔코드 0x1e와 0x9e를 발생시킨다. 이것은 키코드 30과 158로 변환되어 아스키 코드 즉, latin-1 코드 'a'에 해당하는 0141로 전달된다. Delete키의 경우에는 스캔코드 0xe0 0x53 0xe0 0xd3가 발생하고 이것은 키코드 111과 239로 변환되어 4-심볼 시퀀스인 ESC [3 ~로 전달된다. 물론 이것은 US 키보드에 디폴트 키맵의 경우에 해당되는 이야기다. 하나의 동작에 대응하는 키 조합의 예로는 Ctrl-Alt-Del이 있다.)

스캔코드와 키코드간의 변환은 setkeycodes 유틸리티를 사용하면 된다. 아마도 이 기능이 필요한 사람은 극히 드물 것이다. 키코드를 문자, 문자열, 동작 등 키맵으로 변환하는 것은 loadkeys와 setmetamode 유틸리티를 사용하면 된다. 더 자세한 내용은 getkeycodes(8), setkeycodes(8), dumpkeys(1), loadkeys(1), setmetamode(1)를

참조할 것. dumpkeys의 화일 출력 형식과 loadkeys를 사용한 화일 읽기는 keytables(5)에 설명되어 있다.

위에서 '어플리케이션으로 보낸다'라고 말한 것의 정확한 의미는 터미널 드라이버로 보낸다는 뜻이다. 즉, 이 이후의 처리 과정은 시리얼 라인을 통해 오가는 텍스트와 똑같다. 이 처리 과정에 대한 세부 사항은 stty 프로그램으로 설정한다.

제 3 절 콘솔 개요

지금까지와는 반대의 이야기로, 사용자가 콘솔로 뭔가를 보내면, 그것은 우선 표준 tty 처리를 거쳐서 콘솔 드라이버로 넘겨진다. 콘솔 드라이버는 VT100을 흉내내어 (커서 이동이나 스크린 정리와 같은) VT100 이스케이프 시퀀스를 인식할 수 있도록 분석(parse)한다. 이스케이프 시퀀스에 해당하지 않는 문자들은 콘솔이 UTF-8 모드가 아닌한, 네개의 맵 테이블중 하나를 사용하여 제일 먼저 유니코드로 변환된다. 그 후 유니코드 값과 폰트 위치간의 대응관계가 적혀있는 테이블에서 찾아 얻어진 8- 또는 9-비트 폰트 인덱스를 비디오 메모리에 쓰게 되는데, 이 과정이 비디오 카드의 문자 ROM에 있는 문자의 모양을 표시하게 한다. setfont를 사용하면 사용자 입맛에 맞는 폰트를 문자 ROM에 로드할 수 있다. 또한 loadunimap으로는 해당하는 유니코드 맵을 로드하고, mapscrn으로는 사용자 맵 테이블을 로드할 수 있다. 자세한 이야기는 뒤에 나온다.

리눅스에는 동일한 스크린을 사용하는 여러개의 콘솔을 제공한다. (보통 *가상 콘솔* 또는 *가상 터미널*이라고 부른다. 약어로 VC 또는 VT.) 사용자는 이것을 독립적인 로그인 세션, 시스템 로그 상황을 보는 top같은 프로그램이 보내는 여러가지 출력을 보내는 등 독립적인 디바이스로 사용할 수 있다. 이러한 가상 콘솔을 사용하는 방법과 가상 콘솔간 이동에 대해서는 아래 '콘솔간 이동'을 참조하라.

제 4 절 터미널 재설정

화면에 이상한 문자들이 가득 차거나 키보드를 두드려도 이상한 선문자들이 출력된다면 어떻게 해야 하나?

대부분의 프로그램에서 ^L을 누르면 화면을 다시 그린다. 모뎀의 잡음이나 음성 메시지가 화면에 뿌려질 때 이 기능이 제법 유용하다. clear 명령은 화면을 깨끗이 청소한다.

reset 명령은 콘솔 드라이버를 재설정한다. 이 명령은 화면이 이상한 그림 문자로 가득 차거나 화면의 라인 수가 줄어들었을 때 아주 유용하다. 시스템에 이 명령어가 없거나 명령을 내려도 뭔가 다른 동작을 수행한다면 PATH가 걸려있는 디렉토리에 아래와 같은 내용으로 실행화일을 만들어 사용하면 된다.

```
#!/bin/sh
echo -e \\033c
```

동작 내용은 콘솔에 ESC c 문자를 보내는 것이다.

어째서 화면이 때때로 맛이 가서 25줄이 아니라 24줄짜리 또는 1줄짜리 화면으로 되는가? 이 문제의 주범은 원격으로 로그할 때 TERM=linux를 사용하지 않고 TERM=vt100(또는 이와 비슷한 24줄짜리 터미널)을 사용하기 때문이다. 이 문제가 /dev/tty2에서 발생했다면 다른 VT에서 아래와 같이 입력하면 된다.

```
% cat > /dev/tty2
^[c
^D
```

(이 내용은 cat에 4 심볼인 ESC, c, ENTER, Ctrl-D를 누르는 것을 의미한다.) 그 후에 /dev/tty2에서 (^L을 눌러) 화면을 정리하면 문제가 해결될 것이다. 물론 근본적인 해결법은 올바른 termcap이나 terminfo를 사용하는 것이다.

왜 화면에 바이너리 화일을 (cat 등을 사용해서) 뿌리면 이상한 선문자들이 나타나는가? 그것은 이스케이프 시퀀스를 바꾸는 문자들이 몇가지 있는데, 우연히 바이너리에 그런 시퀀스가 포함되어 있었기 때문이다. ESC c는 모든 경우에 해결책이 되는 일반적인 재설정 시퀀스이다. 하지만 잘못된 것이 어떤 것인지 정확히 알고 있는 경우에는 다른 콘솔 특성까지 재설정하지 않고도 문제를 해결할 수 있다. 예를 들어 보자. 아래와 같이

```
% cat
^N
^D
```

입력하면 셸 프롬프트는 온통 선문자로 표시될 것이다. 이제 (뭔지 알 수 없는 상태에서라도)

```
% cat
^O
^D
```

라고 치면 모든 것이 원래대로 돌아올 것이다. (3개의 심볼 ˆN (또는 ˆO), ENTER, Ctrl-D가 cat을 사용하여 보내졌다.) 이 내부 사정을 이해하고 싶으면 아래 '콘솔 문자셋'을 읽어 보라.

이상한 폰트가 로드되어 디폴트 값으로 돌아가고 싶으면

```
% setfont
```

라고 하면 된다. (이것은 디폴트 장소에 디폴트 폰트가 저장되어 있을 경우에만 효력이 있다.) 이 디폴트 폰트에 유니코드 맵이 들어있지 않다면 (그래서 액샌트가 있는 문자에 잘못된 심볼을 준다면)

```
% loadunimap
```

라고 하면 된다. 예를 들어

```
% loadkeys de-latin1
```

라고 치면, German 키보드가 되는데, 이 키보드에서는 Enter 키의 왼쪽에 있는 키는 a-움라우트에 해당한다. a-움라우트는 CP437 코드 페이지에서 발생하고 비디오 카드에 내장된 CP437 폰트가 있다면 작동을 할 것이다. 다른 예로 만약

```
% setfont iso01.f16
```

라고 해서 ISO 8859-1 폰트를 로드하면, (폰트에 붙어있는 유니코드 맵이 없다면) setfont는 커널 유니코드 맵을 무효화시키고, 맵 없이 커널은 바로 폰트로 가게 되어 sio01.f16 폰트를 가진 ISO 8859-1 시스템에 올바르게 작동하게 된다. 하지만

```
% setfont
```

명령으로 이전 폰트로 돌아가면 a-움라우트 대신에 대문자 시그마가 나타난다. 이 폰트에는 유니코드 맵이 포함되어 있지 않아 모든 액센트 문자가 섞이기 때문이다.

```
% loadunimap
```

으로 디폴트 유니코드 맵을 로드하면 (이것은 디폴트 폰트로 적당한데) 모든 것이 다시 잘 될 것이다. 대개의 경우, loadunimap은 직접 호출되는 것이 아니라 setfont를 통하여 호출된다. 따라서 이전의 두 명령은 다음과 같이

```
% setfont -u def
```

로 해도 된다. Ethiopian 폰트와 lat1u*.psf 폰트에는 유니코드 코드 맵이 포함되어 있다. 하지만 대부분의 다른 폰트에는 이것이 포함되어 있지 않다.

구형 터미널에서는 탭과 관계된 출력은 약간의 시간 지연이 필요하다. 이 경우

```
% stty tab3
```

라고 하면 된다. (stty(1) 참조)

`resizecons`나 `SVGATextMode`를 사용하면 비디오 모드를 바꿀 수 있다. 이것은 보통 출력쪽을 설정한다. 입력쪽은 잘못될 가능성이 있는 경우가 많다. `X`나 `DOOM` 또는 raw 모드를 사용하는 여러 프로그램이 다운되면 키보드는 계속해서 raw 모드 (또는 준 raw 모드) 상태에 있게 된다. 이 경우 명령을 시스템에 내리기가 곤란해진다. (아래 'raw 모드에서 빠져나가는 법' 참조) 부적절한 키맵이 로드되었다면

```
% loadkeys -d
```

로 다시 디폴트 맵을 로드하면 된다. 하지만 아마도 '-'를 쳐 넣기가 어려울 것이다. 다른 방법으로

```
% loadkeys defkeymap
```

라고 입력하면 된다. 때때로 글자 배열이 뒤바뀌는 경우도 있다. 알파벳권 지역에서는 크게 네 종류의 자판배열이 있다 : QWERTY, QWERTZ, AZERTY, DVORAK. 앞의 세가지는 자판 배열의 첫 여섯 글자를 따서 붙인 이름이고 대충 말해서 각각 영어, 독일어, 프랑스어권 배열을 나타낸다고 보면 된다. QWERTY와 비교해보면, QWERTZ는 Y와 Z가 바뀌어있고, AZERTY는 Q와 A, W와 Z가 바뀌어 있고, M이 L 오른쪽(세미콜론자리)에 있다. DVORAK은 완전히 다른 자판 배열이다.

4.1 키보드 하드웨어 재설정

뭔가가 잘못되는 경우는 대개 리눅스가 알고 있는 것보다 더 낮은 레벨에서 작업을 할 경우이다. 적어도 두 개의 (키보드와 키보드 콘트롤러) 저수준이 존재하는데, 이 수준에서 사용자는 "keyboard disable" 명령을 키보드 하드웨어에 내릴 수 있다. 어떤 키보드는 세가지 종류의 스캔코드중 하나를 고를 수 있도록 프로그램된 것도 있다.

하지만 이것이 문제가 된 경우에 대해서는 아는 바 없다.

어떤 키보드에는 맵 재설정 기능이 내장된 것도 있다. Stormy Henderson (`stormy@Ghost.Net`)에 의하면 '재수없게도 당신의 키보드가 다시 프로그램되었다면 (Gateway AnyKey 키보드상에서) control-alt-suspend_macro를 누르면 정상 상태로 되돌릴 수 있다고 한다.

제 5 절 Delete와 Backspace

Delete와 Backspace 키가 제대로 동작하도록 하는 것이 그리 간단한 문제는 아니다. 특히 콘솔, `X`, `bash`, `emacs`, 원격 로그인 등 여러가지가 혼합된 환경에서는 더더욱 그러하다. 아마도 사용자가 원하는 바를 정확히 수행하도록 관계된 프로그램 모두에게 전달하려면 여러개의 환경설정화일을 편집해야 될지도 모른다. 한편으로는 키가 발생시키는 코드가 무엇인지도 (그리고 `kermit`이나 `emacs`등에서 이러한 코드가 어떤 규칙으로 다시 매핑되는지도) 문제가 되고, 다른 한편으로는 각 코드에 연결된 기능이 무엇인지도 문제가 된다.

사람들이 종종 '백스페이스 키가 작동하지 않는다.'고 불평하는데, 이것은 마치 이 키에 '이전의 문자를 지우는' 내장된 기능이 있는 것처럼 말하는 것이다. 하지만 불행히도 이 키가 하는 일이라곤 코드 하나를 발생하는 것뿐이며, 커널 tty 드라이버와 모든 어플리케이션 프로그램이 백스페이스 키가 실제로 '이전 문자를 지우는' 동작을 하도록 설정되어 있기를 고대하는 것이 전부이다.

모든 유닉스 프로그램은 '요리된' 모드에서 커널 tty 드라이버를 통해 tty 입력을 받고, 단순한 `stty` 명령이 삭제 문자를 결정한다. 하지만 bash나 `emacs`, `X` 같은 프로그램은 독자적으로 입력을 다루며 하나씩 올바로 동작하는지 확인해야 한다.

5.1 유닉스에서 직전 문자를 지우는데 사용할 문자를 지정하는 방법

```
% stty erase ^?
```

글자가 지워지긴 하지만 이상한 방법으로 지워진다면 사용자의 tty 설정에 뭔가 잘못된 것이 있는 것이다. echoprt가 설정되어 있다면 지워진 문자는 \와 /로 둘러싸여진다. echoe가 설정되어 있지 않으면 삭제 문자가 화면에 찍힐 것이다. (#과 같이 화면에 표시되는 문자인 경우라면 당연하다고 생각할 것이다.) 보통의 경우는 stty echoe -echoprt 상태를 원한다. stty sane이라고 하면 이런 상태가 되며, 그 이외에도 몇가지를 더 설정한다. stty -a라고 명령을 내리면 현재 설정 상태를 화면에 보여준다. 사용자가 올바른 getty를 사용하면 디폴트로 적절한 상태로 되어 있다.

bash나 emacs 등등 많은 프로그램이 독자적인 키바인딩을 가지고 있음에 유의하라. (~/.inputrc, ~/.emacs 등에 정의되어 있다.) 이런 프로그램들은 삭제 문자 설정등에는 영향을 받지 않는다.

표준 유닉스 tty 드라이버는 (화살표 같은) 키를 커서의 현재 위치를 이동하는 뜻으로 인식하지 않는다. 따라서 '현재 문자 삭제' 명령도 없다. 하지만 콘솔상의 bash에서 아래와 같이

```
set editing-mode emacs
"\e[3":delete-char
```

를 ~/.inputrc에 넣어주면 Delete 키를 인식하도록 할 수 있다.

5.1.1 'DEL과 BS가 잘 작동하던 Getty가 지금은 깨져있는 것인가?'

옛날에는 콘솔 드라이버가 DEL (\177)를 받으면 BS Space BS (\010\040\010)를 수행하도록 해두었다. 하지만 지금은 DEL은 무시된다. (이것이 당연한 것이 드라이버는 vt100을 흉내내기 때문이다.) DEL을 출력하지 않는 최신의 getty를 구해서 사용하라.

5.1.2 '첫번째 로그인 시도와 두번째 로그인 시도가 좀 다른것 같다. 이것이 정상인가?'

첫번째 로그인 시도는 getty와 이야기하는 것이지만, 두번째 시도는 login과 이야기하는 것이다. 서로 다른 프로그램이다.

5.2 리눅스에서 키를 누를 때 발생하는 코드를 지정하는 방법

콘솔에서, 정확히 말하자면 raw 모드가 아닌 경우에는

```
% loadkeys mykeys.map
```

이라고 하면 된다. X 환경에서는

```
% xmodmap mykeys.xmap
```

를 사용하면 된다. (XFree86-2.1 이래로) X는 처음 시작시에 X keymap을 초기화할 때, 리눅스 keymap 설정을 읽는다는 사실에 유의하라. 비록 두 시스템이 100% 호환되지는 않지만 이러한 사실은 많은 경우에 xmodmap이 보다 영향력이 있다는 것을 의미하기 때문이다.

예를 들어, 사용자가 백스페이스 키에 BackSpace (^H, 8진수 010)을 보내도록 하려 하고, 회색 Delete 키로는 DEL (8진수로 0177)을 보내려 한다면, /etc/rc.local (또는 사용자만의 부팅 설정화일에)

```
/usr/bin/loadkeys << EOF
keycode 14 = BackSpace
keycode 111 = Delete
EOF
```

라고 첨가하면 된다. 이 설정은 다른 설정을 바꾸지 않았다면 두개의 키 설정만 바꾼다는 사실에 유의할 필요가 있다. (다른 키맵에서도 키 설정을 바꾸고 싶다면, 어느 키맵을 바꿀지 키맵 라인을 지정해 주어야 한다.) 리

눅스 커널은 디폴트로 Ctrl-Backspace 키가 BackSpace를 발생하도록 되어 있다. 이것은 때때로 DEL 코드만 생기는 위급 상황에서 빠져나가는데 유용하다.

왼쪽 Alt 키는 종종 메타키라고 부른다. 디폴트로 왼쪽 Alt-X 키조합은 MetaX로 설정되어 있다. 그렇다면 MetaX는 어떤 키 시퀀스인가? 그것은 (각각의 tty에서) 메타 플래그에 의해서 결정된다. 이것은 `setmetamode` 명령으로 설정할 수 있다. ESC X를 치는 것과 0200과 OR연산을 한 X를 치는 두가지 방법이 존재한다.

5.2.1 '왜 디폴트로 백스페이스 키가 BackSpace를 발생하도록 하지 않았나?'

(i) 왜냐하면 VT100에는 Delete 키가 Enter 키 위에 있었기 때문이다.

(ii) 또한 리누스가 그렇게 정했기 때문이다.

5.3 X에서 Delete와 백스페이스를 바꾸는 방법

```
% xmodmap -e "keysym BackSpace = Delete" -e "keysym Delete = BackSpace"
```

백스페이스 키가 BackSpace를 발생하도록 하려면

```
% xmodmap -e "keycode 22 = BackSpace"
```

Delete 키가 Delete를 발생하도록 하려면

```
% xmodmap -e "keycode 107 = Delete"
```

(하지만 보통 이미 디폴트로 되어 있을 것이다.)

5.4 emacs에서 Delete나 Backspace에 대한 동작 지정하는 방법

사용자의 `.emacs` 화일에 다음 라인을 추가하면 된다.

```
(global-set-key "\?" 'help-command)
(global-set-key "\C-h" 'delete-backward-char)
```

물론 이런 방식으로 하면 다른 키에도 명령을 연결시킬 수 있다. 키 재정의 방식에는 여러가지 major 및 minor 방식이 있음을 유의할 필요가 있다. 예를 들어, incremental search 모드에서는 다음과 같은 코드가 있는 것을 발견할 수 있다.

```
(define-key map "\177" 'isearch-delete-char)
(define-key map "\C-h" 'isearch-mode-help)
```

이것을 보면 위의 두 명령을 global 키 설정으로 사용하는 것은 별로 좋은 방법이 아니라는 것을 알 수 있다. 많은 프로그램에서 ^H = help와 DEL = delete라는 가정하에 키를 사용한다. 물론 반드시 백스페이스 키가 DEL을 발생하도록 키를 설정해야만 하는 것은 아니다. 그러나 키 설정이 이렇게 되어 있지 않으면 emacs에서 사용할 수 있는 가장 낮은 레벨에서는 이 키들을 재설정하는 것이 가장 쉬운 방법이다.

5.5 emacs에서 Delete와 Backspace를 바꾸는 방법

사용자의 `.emacs` 화일에 다음 라인을 추가하면 된다.

```
(setq keyboard-translate-table (make-string 128 0))
(let ((i 0))
  (while (< i 128)
    (aset keyboard-translate-table i i)
    (setq i (1+ i))))
(aset keyboard-translate-table ?\b ?\^?)
(aset keyboard-translate-table ?\^? ?\b)
```

emacs의 최신 버전에서는 keyboard-translate 함수를 제공하기 때문에 간단하게 아래와 같이 해도 된다.

```
(keyboard-translate ?\C-h ?\C-?)
(keyboard-translate ?\C-? ?\C-h)
```

X 환경에서는 (콘솔에서 이 키가 어떤 코드를 만들던 간에) emacs가 Ctrl-h와 백스페이스 키를 구별할 수 있다. 그리고 emacs는 디폴트로 백스페이스 키를 DEL로 간주할 것이다. (물론 ^H에 연결된 도움말 기능이 아니라 문자를 지우는 일을 할 것이다.) 백스페이스와 Delete를 구분하여 사용하려면 다음과 같이 하면 된다.

```
(global-unset-key [backspace] )
(global-set-key [backspace] 'delete-backward-char)
(global-unset-key [delete] )
(global-set-key [delete] 'delete-char)
```

5.6 kermit에서 Delete와 Backspace를 바꾸는 방법

.kermrc 화일에 다음 라인을 추가하면 된다.

```
set key \127 \8
set key \8 \127
```

5.7 xterm에서 입맛에 맞는 tty 모드 설정하는 방법

보통 xterm은 이것을 호출한 프로그램의 tty 모드를 상속받는다. xdm 환경에서는 구형 유닉스 버전 6에서와 같이 디폴트로 설정된 지우기 및 죽이기 문자는 #와 @이다. 이 설정이 마음에 안들면 아래 내용을 /usr/lib/X11/app-defaults/XTerm나 $HOME/.Xresources에 넣으면 된다.

```
XTerm*ttymodes: erase ^? kill ^U intr ^C quit ^\ eof ^D \
                susp ^Z start ^Q stop ^S eol ^@
```

물론 $HOME/.xinitrc나 $HOME/.xsession에

```
xrdb $HOME/.Xresources
```

라는 구절이 있어야 작동한다.

5.8 xmosaic에서 Backspace 키가 DEL을 발생하도록 설정하는 방법

아래 내용을 $HOME/.Xresources에 넣으면 해결될 것이다.

```
*XmText.translations: #override\n\
   <Key>osfDelete: delete-previous-character()
*XmTextField.translations: #override\n\
   <Key>osfDelete: delete-previous-character()
```

하지만 넷스케이프 FAQ에는 다음과 같은 내용이 있다.

```
왜 텍스트 필드에서 백스페이스 키가 말을 듣지 않는가?
디폴트로 리눅스와 XFree86에는 백스페이스와 Delete 키가 잘못 설정되어
있다. (넷스 케이프 네비게이터를 포함한) 모든 모티프 프로그램에서
똑같이 잘못 작동될 것이다.

모티프 스펙에 보면 백스페이스는 이전 문자를 지우도록 되어 있고
Delete는 다음 문자를 지우도록 되어 있다. 리눅스와 XFree86은
백스페이스와 Delete 키가 모두 Delete를 발생하도록 설정되어 있다.

사용자는 xmodmap, xkeycaps, loadkeys중 하나를 사용하여 문제의
키가 Delete 대신에 BackSpace keysym을 발생하도록 할 수 있다.

다른 방법으로는 .motifbind 화일을 만들어 해결할 수 있다. man 페이지
VirtualBindings(3)을 참조하라.

주의: 이 문제를 해결하기위해 *XmText.translations나
*XmTextField.translations 리소스 화일을 사용하지 않는 것이 좋다.
아마도 이것을 건드리면 넷스케이프 네비게이터의 다른 텍스트 필드의
키 연결이 모두 망가질 것이다.
```

5.9 넷스케이프와 같이 모티프를 사용하는 프로그램의 더 좋은 해결책

Ted Kandell (ted@tcg.net)의 해결책은 다음과 같다.

사용자의 .profile 화일에 다음의 내용을 넣자.

```
stty erase ^H
```

bash 사용자라면 .inputrc에 다음 라인을 추가한다.

```
"\C-?": delete-char
"\C-h": backward-delete-char
```

.xinitrc 화일에는 아래의 내용을 추가한다.

```
xmodmap <<-EOF
keycode 22  =  BackSpace osfBackSpace
keycode 107 =  Delete
EOF

# 윈도우 관리자를 실행시킨다.
#(fvwm) 2>&1 | tee /dev/tty /dev/console

stty sane
stty erase ^H
loadmap <<-EOF
keycode 14  = BackSpace
keycode 111 = Delete
EOF
```

이렇게 하면 리눅스/XFree86 기반의 PC 101 또는 102 키보드에서는 반드시 잘 동작할 것이다.

넷스케이프와 같은 모티프 프로그램이 잘 동작하도록 하는데 중요한 부분은 osfBackSpace를 BackSpace와 함께 키코드 22에 넣는 점이다.

= 기호 양쪽에는 반드시 공백이 있어야 한다는 점도 유의할 것.

5.10 termcap와 terminfo는 뭐지?

사용자들이 백스페이스 문제에 부딪히면, 대개 터미널이 사용하는 termcap (또는 terminfo) 항목을 뒤적거린다. 물론 거기에 보면 kb (또는 kbs)라는 항목이 있고, 백스페이스 키가 만드는 코드에 대한 설명이 나온다. 하지만 그다지 많은 프로그램이 이것을 사용하는 것은 아니다. 따라서 만일 특정한 프로그램에서만 문제가 발생한다면 아마도 원인은 다른곳에 있을 가능성이 높다. 물론 termcap (terminfo) 항목을 고쳐서 문제를 해결하는 것도 좋은 생각이다. 뒤에 "TERM 변수" 부분을 참조하면 도움이 될 것이다.

제 6 절 콘솔 문자 세트

커널은 우선 사용자가 입력한 바이트가 어떤 모양의 심볼인지 알아보고, 현재 폰트에서 어디에 위치해 있는지 확인한다.

커널은 콘솔-스크린 심볼로 들어오는 바이트를 해석하는 5가지 방법을 알고있다. 유니코드 (UTF-8) 모드에서 UTF-8 모드는 직접 유니코드로 변환된다. 변환시에는 필요한 모든 심볼이 유니코드에 있다고 가정한다. 이 가정이 만족되지 않는 경우에는 코드 0xff**가 직접 폰트에 접근할 수 있도록 보존된다. 유니코드 모드가 아닌 경우에는 4가지 변환 테이블중 하나를 사용한다. 4가지 테이블은 a) Latin1 -> Unicode, b) VT100 그림문자 -> Unicode, c) PC -> Unicode, d) 사용자 정의 문자.

문자 세트에는 G0와 G1이 있고 이 중 하나가 현재 문자 세트로 사용된다. (초기값은 G0) ^N이라고 쳐 넣으면 G1이 현재 문자 세트가 되고 ^O라고 하면 G0가 현재 문자 세트로 된다.

이 변수 G0와 G1은 변환 테이블을 가리키며, 이것들은 사용자가 바꿀 수 있다. 초기값은 각각 테이블 a)와 b)를 가리키게 되어 있다. 시퀀스 ESC (B, ESC (U, ESC (K라고 하면 각각 G1이 변환 테이블 a), b), c), d)를 가리키게 된다.

시퀀스 ESC c는 터미널을 재설정하는데, 이 기능은 스크린이 먹통이 되었을 때 유용하다. 종종 추천되는 방식인 `echo ^V^O`는 단지 G0를 현재 문자 세트로 설정할 뿐이므로 G0가 테이블 a)를 가리키게 된다는 보장이 없다. 배포본 중에는 `echo ^[c`를 수행해주는 reset(1)이라는 프로그램이 있는 경우도 있다. 콘솔에 사용되는 termcap 항목이 올바르다면 (그리고 `:rs=\Ec:` 항목이 있다면), `setterm -reset`이라고 해도 같은 효과를 얻을 수 있다.

mapscrn(8)을 사용하면 사용자 정의 맵 테이블을 설정할 수 있다. 매핑을 하게 되면, 예를 들어 심볼 c를 화면에 찍으려면 심볼 `s = map[c]`를 비디오 메모리로 보내게 된다. `s`에 대응되는 비트맵이 문자 ROM에 있으며, setfont(8)을 사용하면 바꿀 수 있다.

제 7 절 콘솔간 이동

콘솔간 이동의 기본 키는 Alt-Fn 또는 Ctrl-Alt-Fn이다. `X` 환경이나 최근 버전의 `dosemu`에서는 Ctrl-Alt-Fn 만이 제대로 작동할 것이다. 많은 keymaps에서 Alt-오른화살 키와 Alt-왼화살 키로 할당된 콘솔간의 순환식 이동을 허용한다.

XFree86 1.3은 X 윈도우로 이동할 때, Alt 키가 눌려 있는 것을 인식하지 못한다. 따라서 다른 VT로 즉각 되돌아가지 못하고, Alt 키를 떼어야 반응을 한다. 다른 곳에서는 잘 작동된다. 커널은 항상 모든 키의 누름/뗌 상황을 추적한다. (물론 가능한 한 그렇다는 이야기다. 몇몇 키보드의 어떤 키들은 눌리거나(FOCUS 9000의 PFn 키) 뗄 때(많은 키보드의 Pause 키) 스캔코드를 보내지 않아 어쩔 수 없는 경우도 있다.)

XFree86 1.3은 처음 시작될 때, 문자 ROM에 로드된 폰트를 저장해두고 콘솔간 이동을 할 때, 그것을 꺼내어 사용한다. 따라서 VT에서 setfont를 사용하면 X로 이동했다가 되돌아오면 이전 값으로 복귀된다. X 환경에서 setfont를 사용하면 이상한 일들이 생긴다.

chvt 명령을 사용하면 프로그램 사용중에도 VT를 바꿀 수 있다.

7.1 가상 콘솔의 갯수 바꾸는 방법

이 질문은 아직도 가끔 나오지만 사실 디폴트로 설정된 갯수으로도 충분하다. 커널 1.1.54이후로 1부터 63 사이의 가상 콘솔이 존재한다. 새로운 가상 콘솔은 생성되면서 열린다. deallocvt라는 유틸리티를 사용하면 없앨 수 있다. (그러나 없앨려면 여기에 연결된 프로세스가 하나도 없고, selection이나 gpm 같은 프로그램이 이 콘솔에 있는 텍스트를 선택하고 있지 않아야 한다.)

이보다 이전 버전의 커널인 경우라면 include/linux/tty.h에 있는

```
#define NR_CONSOLES     8
```

를 바꾸어주고 커널을 컴파일하면 된다. (이 숫자를 63보다 크게 하지는 말 것.)

가상 콘솔이 아예 없는 경우라면 MAKEDEV 또는 mknod ttyN c 4 N (여기서 N은 tty 번호를 지정)로 tty 디바이스를 만들 수 있다. 예를 들어보면

```
for i in 9 10 11 12; do mknod /dev/tty$i c 4 $i; done
```

좀 더 좋은 예로는 (이 방법이 사용자와 허가권에 유의한다.),

```
for i in 9 10 11 12; do /dev/MAKEDEV tty$i; done
```

getty를 돌릴 VC가 새로 필요하다면 /etc/inittab에 라인을 추가하면 된다. (하지만 사용하고 있는 getty가 두개만 있는 것이 좋다. 그리고 추가로 더 필요해지면 동적으로 생성하면 된다. 사용자가 모든 콘솔을 사용하지 않는다면 이 방식이 보다 많은 메모리를 사용할 수 있다. /etc/inittab를 열어서 최초의 두개를 제외한 모든 getty를 주석으로 막아 두어라.)

콘솔을 동적으로 할당받을 때는 getty가 하나나 두개 정도만 돌도록 하는 것이 가장 쉽다. open -l -s bash라고 하면 더 많이 열 수 있다. (연결된 프로세스가 없는) 사용하지 않는 콘솔은 deallocvt(예전에는 disalloc)을 사용하여 없앨 수 있다. 그러나 작업중에 갑자기 콘솔이 더 필요해져서 당장 open 명령을 내릴 bash 프롬프트가 없다면 어떻게 하는가? 다행이도 현재 콘솔에서 무슨 일을 하건, 한 큐로 새로운 콘솔을 만드는 방법이 있다.

kbd-0.95.tar.gz에 있는 spawn_login을 설치하고 /etc/rc.local에 다음과 같이 넣어 두어라.

```
loadkeys << EOF
alt keycode 103 = Spawn_Console
EOF
spawn_login &
```

그 다음에 Alt-위화살 키를 누르면 login을 돌리고 있는 새로운 VC가 생성될 것이다. (물론 생성되어서 그곳으로 이동된다.) spawn_login & 대신에 spawn_console &를 설치했다면 아마도 새 콘솔에 bash가 돌고 있을 것이다. open-1.4.tgz와 dynamic-vc-1.1.tar.gz를 참고하라.

최신 버전의 init를 가지고 있다면 /etc/inittab의 kbrequest아래에서 Spawn_Console 키가 눌렸을 때 무슨 작동을 하는지 설정할 수 있다. inittab(5)를 참고할 것.

(이 동작은 완전히 다른 것으로 설정되어 있을 수 있다. Spawn_Console 키는 내가 사용하기 때문에 부르는 이름일 뿐이다. 다른 목적으로 사용한다면 덜 헷갈리도록 다른 이름을 붙이는 것이 좋을 것이다. 어떤 사람들은 /etc/inittab에

```
kb::kbrequest:/sbin/shutdown -h now
```

라고 넣어두고, keymap에는

```
control alt keycode 79 = KeyboardSignal
control alt keycode 107 = KeyboardSignal
```

를 넣어두고서 Ctrl-Alt-End를 시스템 셧다운 키로 사용하기도 한다.

루트 권한으로의 접속은 /etc/securetty에 등록되어 있는 터미널에서만 가능하다. /etc/ttys와 /etc/ttytype 화일에서 터미널 설정값을 읽는 프로그램이 있다. 이런 화일을 가진 사용자가 추가로 콘솔을 생성했다면 이 콘솔 항목을 이 화일에 추가하는 것도 괜찮은 생각이다.

제 8 절 Ctrl-Alt-Del와 그 밖의 다른 특별한 키 조합

8.1 Ctrl-Alt-Del (부팅)

사용자가 Ctrl-Alt-Del(또는 loadkeys에서 keysym Boot에 지정된 키는 무엇이든)을 누르면 머신은 (sync 동작 없이) 즉시 재부팅을 하거나, init가 SIGINT를 보낸다. 전자의 행동이 디폴트 값이다. 이것은 루트 권한에서 시스템 호출 reboot()를 사용하면 바꿀 수 있다. init가 SIGINT를 얻었을 때 동작하는 내용은 사용하는 init의 버전에 따라 달라진다. 종종 /etc/inittab의 pf 항목에 의해 정해진다. (이것은 사용자가 설정하기에 따라서는 임의의 프로그램을 실행시킬 수도 있음을 의미한다.) 현재 커널에서 Ctrl-AltGr-Del은 더이상 Boot에 할당되어 있지 않다.

8.2 다른 키 조합

```
Name            Default binding
--------------------------------
Show_Memory     Shift-Scrollock
Show_Registers  AltGr-ScrollLock
Show_State      Ctrl-ScrollLock
Console_n       Alt-Fn and Ctrl-Alt-Fn  (1 <= n <= 12)
Console_{n+12}  AltGr-Fn                (1 <= n <= 12)
Incr_Console    Alt-RightArrow
Decr_Console    Alt-LeftArrow
Last_Console    Alt[Gr]-PrintScreen
Scroll_Backward Shift-PageUp
Scroll_Forward  Shift-PageDown
Caps_On                                 (CapsLock은 토글이다. 이렇게 하면 키가 설정된다.)
Compose         Ctrl-.
```

8.3 X에서의 키조합

```
Ctrl-Alt-Fn     VT n으로 이동
Ctrl-Alt-KP+    다음 모드
Ctrl-Alt-KP-    이전 모드
Ctrl-Alt-Backspace      X 죽이기
```

어떤 마더보드에서는 Ctrl-Alt-KP-와 Ctrl-Alt-KP+가 터보 보튼을 누르는 것과 똑같은 동작을 한다. 두 키보드 모두 스캔코드 1d 38 4a ca b8 9d와 1d 38 4e ce b8 9d를 발생시킨다. 이것은 터보 (>= 25MHz)와 보통(8

또는 12 MHz) 모드를 전환한다. (종종 이 키 조합은 마더보드에서 점퍼가 설정되어 있을 때만 효과가 있는 경우도 있다.)

Perry F Nguyen (pfnguyen@netcom22.netcom.com)에 의하면, AMI BIOS는 BIOS 패스워드가 설정되어 있으면, Ctrl-Alt-Backspace를 누르면 CMOS/BIOS 패스워드가 입력될 때까지 키보드를 잠궈두고 LED를 켜둔다고 한다.

8.4 Dosemu에서의 키 조합

```
Ctrl-Alt-Fn     VT n으로 이동 (0.50 버전 이후부터; 이전에는 Alt-Fn)
Ctrl-Alt-PgDn   dosemu 종료 (RAW 모드일 경우)
(다른 키들은 dosemu 문서를 참조할 것.)
```

8.5 심볼 조합하는 방법

심볼 하나는 서너개의 키보드를 사용하여 만들 수 있다.

- 왼쪽 Alt를 누른채로, 키패드에 있는 숫자를 친 후, Alt 키를 놓으면 이 숫자에 해당하는 코드를 가진 심볼을 만들어 낸다. (유니코드 모드에서도 같은 구조이다. 하지만 이 경우 유니코드 심볼을 정의하는데 사용되는 4자리 16진수이어야 한다.)
- 구별 부호 다음에 심볼이 오면 그 구별부호가 붙은 심볼을 발생시킨다. 그 조합이 정의되어 있지 않으면, 각각의 키가 별도로 취급된다. (역자 주: 구별 부호란 액샌트 기호나 움라우트 같은 부호를 의미함.) 어느 키를 구별부호로 사용할 것인가는 사용자가 설정할 수 있다. 디폴트 값은 아무것도 설정되지 않은 상태이다. 5개(2.0.25 이후는 6개)의 구별 부호를 (loadkeys(1)을 사용하여) 정의할 수 있다. 그 5개(6개)는 dead_grave, dead_acute, dead_circumflex, dead_tilde, dead_diaeresis (dead_cedilla)이다. 정확히 말하자면, 어떤 구별부호가 붙는 지도 사용자가 설정할 수 있다. 이 때의 심볼은 Compose + 구별기호 + 심볼과 같다.
- Compose 다음에 두개의 심볼이 따라오면 조합된 심볼을 생성한다. 이 조합들도 사용자가 정의할 수 있다. 요즘에는 디폴트 값으로 68개의 조합이 정의되어 있다. "dumpkeys | grep compose"라고 하면 이 정의들을 볼 수 있다.
- (1.3.33 이래로) 'Sticky' 변경 키라는 것이 있는데, 예를 들면 SControl, C로 ^C를 입력할 수 있고, Scontrol, SAlt, Backspace로 Ctrl-Alt-Backspace를 입력할 수 있다.

적어도 이러한 조합 메카니즘에는 3가지가 있음을 유의하자.

1. loadkeys와 결합해서 사용하는 리눅스 키보드 드라이버 메카니즘.
2. X 메카니즘 - X386keybd(1), 최신 버전은 XFree86kbd(1)을 참조. X11R6하에서 /usr/X11R6/lib/X11/locale/iso8859-1/Compose를 참조.또한 Andrew D. Balsahttp://wauug.erols.com/\~{}balsa/linux/deadkeys/index.html 의 조언을 참조하면 도움이 될 것이다.
3. "iso-insert.el"를 로드하거나 'iso-accents-mode'를 호출하여 얻는 emacs 메카니즘.

X에서는 두 심볼의 순서는 관계없다. Compose-,-c와 Compose-c-는 둘 다 c-cedilla를 발생시킨다. 리눅스와 emacs에서는 단지 앞의 시퀀스만이 작동한다. X에서는 compose 조합의 목록이 고정되어 있지만 리눅스와 emacs는 변동 가능하다. 이 세 방법에서 디폴트 설정은 비슷하게 되어 있지만 세세히 보면 약간 다르다.

8.6 The SysRq key

커널을 CONFIG_MAGIC_SYSRQ를 설정해놓고 컴파일했다면 (이 기능은 리눅스 2.1.43 이후에 생겼다.) 현재 키보드 모드에 상관없이 특별한 시스템 함수가 연결되어 있는 단일 키가 존재한다. (이것은 `<linux/keyboard.h>`에 정의되어 있다.) PC 아키텍처에서 이 특수키는 당연히 Alt+SysRq 키이고 두 Alt 키중 어느것이라도 동작할 것이다. (하지만 CONFIG_MAGIC_SYSRQ가 설정되지 않은 상태라면 이 키의 디폴트 동작은 이전 콘솔로 돌아가는 것이다.)

이 키를 누른채로 다른 키를 누르면 해당되는 동작이 수행된다. 이 동작은 사용자가 누구건간에 수행된다. 더 자세한 것은 `/drivers/char/sysrq.c`를 참조하라. 이 기능은 커널 해커에게만 의미가 있으므로 이정도만 언급해도 충분할 것이다. 하지만 좀 더 해보자.

r키를 누르면 키보드 모드가 K_XLATE로 재설정된다. k키는 SAK와 콘솔 재설정이 수행된다. b키는 시스템이 즉시 재부팅된다. (사용자가 원치 않는 뭔가가 설정되는지 유의하라.) o키는 (머신이 이 기능을 지원하면) 전원이 꺼진다. s키는 응급 sync가 예약된다. u키는 응급 read-only remount가 예약된다. p, t, m 키는 여러가지 정보를 보여준다. (같은 정보를 RAlt,RCtrl,RShift+ScrollLock로도 볼 수 있다.) e, i, l키는 각각 모든 프로세스들이 SIG_TERM 또는 SIG_KILL을 받도록 한다. l키는 init 프로세스까지 죽인다. 숫자들은 로그 레벨을 설정한다. 그 외에 다음 것들은 간략한 정보를 보여준다. `SysRq: unRaw saK Boot Off Sync Unmount showPc showTasks showMem loglevel0-8 tErm kIll killalL`.

주의: 이것은 아주 위험한 작업이다! 또한 이것은 사실상 사용자의 keymap을 사용하지 않으므로 사실상 keymap의 상태나 전체 커널의 상태도 불확실하다는 것을 의미한다. 디보락 키보드 사용자라면 더더욱 문제가 된다. 다른 자판 배열을 사용하는 것이 안전할 것이다. 영어와 다른 배열인 프랑스어나 독일어 등의 키보드는 A,M,Q,W,Y,Z 같은 위험한 문자들은 이 동작에 사용하지 않는 것이 좋다.

제 9 절 raw 모드에서 탈출하는 방법

K_RAW 키보드 모드를 사용하는 프로그램이 K_XLATE로 키보드 모드를 복귀시키지 않고 종료되면 아무것도 할 수 없는 상태가 되어 버린다. 심지어는 Ctrl-Alt-Del도 먹지 않는다. 하지만 재수가 좋으면 재부팅 버튼을 누르지 않아도 되는 경우가 있다. (이것이 당연히 바람직한 것이, 리부팅으로 Hack 게임을 죽이면 이것을 하던 사람들이 화를 낼 것이 아닌가! 또한 화일 시스템이 손상을 입을 수도 있다.) 쉬운 방법으로는 다른 터미널이나 다른 시스템에서 접속하여 `kbd_mode -a`를 실행시키면 된다. 아래의 과정에서 X는 실행중이지 않고, 디스플레이는 텍스트 모드이며 사용자가 bash 프롬프트에 있으며 키보드는 US배열이고 종료 문자는 Ctrl-C임을 가정했다.

1 단계. X 기동. 2를 누른채로 F12를 누르면서 곧바로 =를 누른다. 이렇게 하면 X가 뜰 것이다. (설명: 임의의 키를 하나 눌렀을 때 만약 키코드 K가 발생한다면 그 키를 뗄 때는 K+128이라는 키코드가 발생하게 된다. 아마도 사용하고 있는 쉘은 이 때 발생하는 높은 값의 문자를 제대로 처리하지 못할 것이다. 따라서 키를 누르고 있으면 이렇게 높은 값의 키코드가 발생하는 것을 막을 수 있다. 하지만, 재빠르게 동작해야 한다. 왜냐하면 그렇지 않으면 키 반복이 시작되기 때문이다. 숫자 2는 이전 작업을 무조건 종료시키는 Ctrl-C를 내보내고, F12는 X를, =는 리턴을 발생시킨다.) 아마도 이렇게 띄운 X는 화면을 회색으로 만들었을 것이다. 왜냐하면 `.xinitrc`가 지정되지 않았기 때문이다. 하지만 Ctrl-Alt-Fn은 먹히기 때문에 사용자는 다른 VT로 이동할 수 있다. (물론 Ctrl-Alt-Backspace도 작동하므로 종료할 수 있지만 현 상태에서 종료는 원하는 바가 아니다.)

2 단계: 키보드 모드 변경 설정. (간단히 `sleep 5; kbd_mode -a`라고 하면 된다.)

3 단계: 다시 X 종료. Alt-Fx(종종 Alt-F7)로 X로 돌아가서 Ctrl-Alt-Backspace를 누르면 X가 종료된다. 5초가 지나면 키보드가 다시 말을 듣게 될 것이다.

이런 상황에 상시 대비하고 싶다면 `\215A\301` (3 심볼)을 `kbd_mode -a`에 대한 alias로 만들어 두자. 이제는 단지 = F7 = 이렇게 3 키만 누르면 키보드가 제정신으로 돌아오게 될 것이다.

제 10 절 키보드 LED

1. 각 tty당 키보드 플래그가 존재한다. 각각의 VC에는 고유의 NumLock, CapsLock, ScrollLock이 존재한다. 디폴트 값으로 이 키보드 플래그는 LED에 표시된다. 이것의 설정을 바꾸는 통상적인 방법은 해당되는 키를 누르는 것이다. (첨언: 어플리케이션 키모드에 있을 때 NumLock 키를 누르면 NumLock 상태를 바꾸지 않고 이스케이프 시퀀스를 발생시킨다. 어떠한 경우에라도 이 키를 먹도록 하고 싶다면 NumLock을 Bare_Num_Lock에 연결시켜두면 된다.

2. 각 tty당 디폴트 키보드 플래그가 존재한다. 재설정 신호가 들어오면 키보드를 초기화한다. 따라서 항상 NumLock이 켜지도록 할 수도 있다. '`setleds -D`라고 하면 된다.

3. led가 키보드 플래그를 반영하지 않고 뭔가 다른 것을 나타낼 수도 있다.

3A. 뭔가 다른 것중 하나는 커널에 있는 세개의 비트로서, 사용자가 어떤 하드웨어나 소프트웨어 상태를 감시하기 위해 사용될 수 있다. 이러한 기능을 원한다면 커널 소스를 편집하여 `register_leds()`를 호출하도록 해야 한다.

3B. 사용자 프로그램이 LED에 표시되도록 하는 것도 여기에 해당된다. 따라서 이러한 기능을 사용하면 쓸만한 프로그램을 만들 수 있다. 이렇게 하려면 KDSETLED ioctl을 사용해야 한다.

후자와 같은 사용은 tty별로 적용되는 것이 아니지만 전자와 후자간의 선택은 tty당 할 수 있다.

요약: 각각의 tty에는 플래그 `kbd->ledmode`가 있다. 이것이 LED_SHOW_FLAGS 값을 가지면 해당 tty에 키보드 플래그(NumLock 등)가 표시된다. 이것이 LED_SHOW_MEM 값을 가지면 선택된 3개의 메모리 주소가 표시된다. 이것이 LED_SHOW_IOCTL 값을 가지면 led는 KDSETLED ioctl가 할당한 가장 최신의 값을 보여준다.

하나만 더 추가하면, X는 ioctl을 사용하여 LED를 설정한다. 하지만 종료가 될 때 X를 띄웠던 VT를 재설정하지 못한다. 따라서 X를 사용하고 나면 디폴트 LED_SHOW_FLAGS 상태로 있지 않은 VT가 생길 수도 있다. 이런 경우 '`setleds -L`'을 그 VT에서 실행하면 문제가 해결된다. setleds(1)를 참조하라.

제 11 절 TERM 변수

많은 프로그램들이 TERM 변수와 `/etc/termcap` 또는 `/usr/lib/terminfo/*` 데이타베이스를 사용하여 문자열을 어디로 보낼 지, 커서를 어디로 움직일 지 등등을 결정한다. 또한 사용자 백스페이스 키, 기능키 등등이 보낸 문자열을 결정하는데 사용하기도 한다. 이 값은 최초에는 커널이 (해당 콘솔에 대해) 설정한다. 보통, 이 변수는 `/etc/ttytype`를 사용하여, 또는 `/etc/inittab`에 지정된 인수로 재설정된다. 때때로 `/etc/profile`에서 설정되기도 한다.

구형 시스템에서는 `TERM=console` 또는 `TERM=con80x25`를 사용한다. (ncurses 1.8.6을 사용하는) 신형 시스템은 좀 더 자세히 기술되어 있는 `TERM=linux` 또는 `TERM=linux-80x25`를 사용한다. 그러나 `setterm`의 옛 버전은 `TERM=con*`에 대해 제대로 작동하지 않는다. `TERM=linux`를 사용하라.

커널 1.3.2 이래로 콘솔에 대한 커널의 디폴트는 `TERM=linux`로 되어 있다.

리눅스에 대한 항목이 없는 termcap을 사용하고 있다면 콘솔에 대한 항목에 아래와 같이

```
console|con80x25|linux:\
```

리눅스에 대한 항목을 넣고, `/usr/lib/terminfo/l/linux`를 `/usr/lib/terminfo/c/console`에 복사하거나 심볼릭 링크를 해두면 된다.

11.1 Terminfo

리눅스 콘솔에 대한 terminfo 항목은 ncurses 1.8.6 이후로 `kich1=\E[2~` 항목을 빼두었다. 하지만 일부 프로그램에서 필요한 경우가 있다. 이것을 사용하려면 화일을 편집하고 tic을 실행시키면 된다.

제 12 절 ASCII가 아닌 문자세트에서 프로그램 사용하는 법

불행했던 과거에는 이것이 상당히 문제거리였다. 각각의 프로그램들이 개별적으로 모든 비트가 제대로 남아 있는지 확인해야 했다. 물론 현재도 모든 것이 쉬운 것은 아니다. 하지만 최근에 많은 gnu 유틸리티에서 LC_CTYPE=iso_8859_1 또는 LC_CTYPE=iso-8859-1에 대해서 어떻게 대응해야 하는지 알고 있다. 이 방법을 먼저 시도해 보고 안되면 아래에 소개한 힌트를 참고해 보라. libc의 최신 버전에서 setlocale()은 locale 화일(즉, /usr/lib/locale)을 설치해야만 제대로 작동한다는 점에 유의해야 한다.

우선, 8번째 비트가 커널 입력 프로세스에서 살아 남도록 stty cs8 -istrip -parenb가 설정되었는지 확인한다.

A. emacs의 경우에는 개별적인 사항은 그 버전에 상당히 의존한다. 아래 정보는 버전 19.34에 대한 것이다. 사용자의 $HOME/.emacs에 아래와 같은 내용을 넣는다.

```
(set-input-mode nil nil 1)
(standard-display-european t)
(require 'iso-syntax)
```

첫번째 줄 (정확히는 마지막의 1)은 emacs가 입력되는 문자들의 8번째 비트를 없애지 않도록 지정한다. 두번째 줄은 emacs가 비ASCII 문자를 8진수 이스케이프로 표시하지 않도록 한다. 세번째 줄은 syntactic 특성을 지정하고 Latin-1 문자세트를 변환 테이블로 적용하도록 한다. 사용자가 환경변수로 LC_CTYPE=ISO-8859-1를 설정했다면 뒤의 두 줄은 중복되는 것이다. (여기에 사용되는 변수는 LC_ALL 또는 LANG일 수도 있다. 값은 뒤에 붙는 숫자가 '88591' 또는 '8859-1' '8859_1' 등 어느것도 된다.)

여기까지 되어 있으면 일단 OK. 비ASCII ISO 8859-1 심볼을 표시하지 못하는 터미널에서는

```
(load-library "iso-ascii")
```

라고 명령을 내리면 액샌트 기호의 문자가 표시될 것이다. 사용자의 keymap이 비ASCII 문자를 만들기 쉽지 않다면

```
(load-library "iso-transl")
```

라고 명령을 내리면 2문자 시퀀스 Ctrl-X 8을 하나의 조합문자로 만든다. 따라서 4문자 시퀀스 Ctrl-X 8 , c는 c-cedilla를 만든다. 무지 불편한 방법이다.

```
(iso-accents-mode)
```

명령은 ISO-8859-1 액센트 모드를 토글할 것이다. 이 모드에서는 ', ', ", ^, ~, / 이렇게 6개의 키가 그 다음 나오는 심볼을 변경하는 데드키이다. 특수한 조합: ~c는 cedilla가 붙은 c, ~d는 Icelandic eth, ~t는 Icelandic thorn, "s는 German sharp s, /a는 ring 붙은 a, /e는 a-e ligature, ~<와 ~>는 guillemots, ~!는 뒤집힌 감탄부호, ~?는 뒤집힌 물음표, "는 뾰죽한 액센트이다. 액센트의 기본 대응은 위와 같다. 변수 iso-languages는 (언어 이름, 액센트 대응) 쌍의 목록이다. 디폴트와 다른 대응을 하려면

```
(iso-accents-customize LANGUAGE)
```

를 사용하면 된다. 여기서 LANGUAGE는 "portuguese", "irish", "french", "latin-2", "latin-1" 중 하나이다.

리눅스의 디폴트 조합 문자는 Ctrl-이기 때문에 어디서나 이것을 사용하는 것이 편리할 것이다. 다음과 같이 한번 해보자.

```
(load-library "iso-insert.el")
(define-key global-map [?\C-.] 8859-1-map)
```

emacs -nw를 사용하는 사람이라면 두번째 줄이 말을 듣지 않을 것이다. 이 경우, .Xresources에 다음과 같은 라인을 넣어두면 된다.

```
XTerm*VT100.Translations:       #override\n\
      Ctrl <KeyPress> . : string("\0308")
```

B. less에서는 환경변수에 LESSCHARSET=latin1을 넣어두면 된다. man 출력에 \255 또는 <AD>가 보이는 경우에도 효과가 있을 것이다. less의 어떤 버전에서는 Latin-1 출력에 허가가 거부되는 경우에 소프트 하이픈(octal 0255, hex 0xAD)을 이런 식으로 만든다.

C. ls에서는 옵션 -N을 주면 된다. (아마도 alias를 만들어 두어도 좋을 것이다.)

D. bash (버전 1.13.*)에서는 (Danish HOWTO에 따르면) 아래의 내용을 $HOME/.inputrc에 넣어두면 된다고 한다.

```
set meta-flag on
set convert-meta off

set output-meta on
```

E. tcsh에서는

```
setenv LANG     US_en
setenv LC_CTYPE iso_8859_1
```

를 사용하면 된다. 시스템에 nls가 설치되어 있으면 그에 해당되는 루틴이 사용된다. 그렇지 않은 경우에는 tcsh는 LANG과 LC_CTYPE에 주어진 값에 관계없이 iso_8859_1을 가정한다. tcsh(1)의 NATIVE LANGUAGE SYSTEM 단락을 참조하면 도움이 될 것이다. (Danish HOWTO에는 setenv LC_CTYPE ISO-8859-1; stty pass8라고 나와 있다.)

F. flex에서 생성하는 파서가 8비트 입력을 처리할 수 있게 하려면 -8 옵션을 주면 된다. (너무 당연하다.)

G. elm에서는 displaycharset을 ISO-8859-1로 설정하면 된다. (Danish HOWTO: LANG=C와 LC_CTYPE=ISO-8859-1)

H. (lynx와 같이) curses를 사용하는 프로그램의 경우에는 David Silbey가 다음과 같은 이야기를 했다. 보통의 curses 패키지는 최상위 비트를 비디오 모드 반전용으로 사용한다. (/usr/include/curses.h에 정의되어 있는 _STANDOUT 플래그를 볼 것.) 하지만 ncurses는 8-비트를 사용하지 않는 것 같으며 iso-latin-8859-1을 올바로 보여준다.

I. (man과 같이) groff를 사용하는 프로그램의 경우, -Tascii 대신에 -Tlatin1을 사용하면 된다. man 프로그램의 구 버전에서는 col도 사용했기 때문에 다음번 항목도 같이 적용해야 한다.

J. col의 경우, 1) setlocale(LC_CTYPE,"");를 할 수 있도록 수정했는지 확인하고, 2) LC_CTYPE=ISO-8859-1를 환경변수로 넣어두어라.

K. rlogin의 경우, -8 옵션을 사용하면 된다.

L. joe의 경우, L. For joe, sunsite.unc.edu:/pub/Linux/apps/editors/joe-1.0.8-linux.tar.gz는 환경설정 화일을 편집하면 된다. 어떤 사람들은 /usr/lib/joerc 첫번째 칼럼에 -asis 옵션을 넣으라고 한다.

M. LaTeX의 경우, \documentstyle[isolatin]{article}. LaTeX2e: \documentclass{article}\usepackage{isolatin}, 여기서 isolatin.sty는 ftp://ftp.vlsivie.tuwien.ac.at/pub/8bit에서 구할 수 있다.

ISO-8859-1 주제에 대한 여러가지 좋은 논의와 8비트 문자를 다루는 방법이 grasp.insa-lyon.fr:/pub/faq/fr/accents에 (프랑스어로) 실려 있다. (영어로 된) 다른 좋은 내용은 ftp.vlsivie.tuwien.ac.at:/pub/8bit/FAQ-ISO-8859-1에서 볼 수 있으며, 미러 사이트는 rtfm.mit.edu:pub/usenet-by-group/comp.answers/character-sets/iso-8859-1-faq 이다.

8비트 문자를 제대로 다루지 못하는 프로그램을 손수 고쳐보고 싶다면 한가지 유념할 점이 있다. 만약 signed char 형 변수를 사용한다면 문자는 음수가 되는 경우가 생겨서 이것을 배열 인덱스로 취급하는 방법은 듣지 않는다. 몇몇 프로그램의 경우 경우를 잘 가려서 (unsigned char) 캐스트로 고칠 수 있다.

제 13 절 XFree86-2.1이 keymap을 초기화할 때 하는 일이 정확이 무엇일까?

버전 2.1 이래로 XFree86은 가능한한 리눅스 keymap으로부터 사용할 keymap을 초기화한다. 하지만 각 키당 16개의 항목을 가지고 있고, (하나당 Shift, AltGr, Ctrl, Alt의 조합으로 변경할 수 있으므로) 현재는 각 키당 256개의 항목을 가지고 있다. 반면에 X는 각 키당 4개의 항목만이 있다. (각 키당 Shift, Mod의 조합) 따라서 키 정보의 일부는 필연적으로 잃을 수 밖에 없다.

제일 처음 X는 Xconfig 화일을 읽는다. 여기에는 LeftAlt, RightAlt, RightCtl, ScrollLock 키가 각각 Meta, ModeShift, Compose, ModeLock 또는 ScrollLock으로 정의되어 있을 것이다. X386keybd(1) 또는 XFree86kbd(1)를 참조하라.

오른쪽 Ctl키가 ModeShift 또는 ModeLock으로 정의되어 있지 않으면 Mod는 보통 왼쪽 Alt키가 사용된다. 오른쪽 Ctl키가 이렇게 정의되어 있으면 오른쪽 Ctl키가 사용된다. 오른쪽 Alt키가 이렇게 정의되었다면 이 경우는 오른쪽 Alt키가 사용되는 것이다. 이런 방식으로 한 키에 대하여 리눅스에서 사용하는 16개의 키 의미에서 XFree86의 4개의 의미를 결정한다. 지금의 리눅스는 두개의 Ctl키(좌,우)간, 두개의 Shift키(좌,우)간의 차이는 구별하지 않는 것이 디폴트이다. 하지만 X는 이것을 구별한다.

커널 keymap이 읽혀지고 보통 이에 해당하는 명확한 X 바인딩이 만들어진다. "action 키"에 해당하는 Show_Memory, Show_State, Show_Registers, Last_Console, Console_n, Scroll_Backward, Scroll_Forward, Caps_On, Boot에 대한 바인딩은 구별하지 않는 바인딩으로 취급되어서 무시되고, (ShiftLock을 제외한) lock들과 "ASCII-x" 키들이 만들어진다.

다음, Xconfig 화일에 정의되어 있는 것들이 사용된다. (따라서, Xconfig의 Compose에 대한 정의가 리눅스 keymap에 있는 값들을 무시하고 사용될 것이다.)

기능키에 연결되어 있는 문자열은 어떻게 되는가? 아무일도 안일어난다. X에는 그러한 개념이 아예 없다. (하지만 xterm에서 기능키에 문자열을 정의하는 것은 가능하다. 그러나 윈도우 관리자가 이 키를 먼저 가로챈다는 점에 유의하라.)

나는 아직 Alt키가 눌렸을 때, xterm이 X keymap을 사용하는지 확신하지 못하고 있다. 그냥 리소스 eightBitInput을 살펴보고 문자의 최상위 비트가 설정되어 있는지에 따라 추가적인 Escape 문자를 발생시키는 것 같다. (마치 콘솔에서 setmetamode(1)가 그러하듯이.)

제 14 절 잘 안쓰이는 키와 키보드

두개의 키 PrintScrn/SysRq와 Pause/Break는 각각 두개의 키코드를 가지고 있다는 점에서 좀 특별하다. 전자는 Alt키가 눌려있을 때는 키코드 84를 가지며, 그렇지 않은 경우는 99를 갖는다. 후자는 Ctrl키가 눌려있을 때는 키코드가 101이며 그렇지 않은 경우는 키코드가 119이다. (따라서, Alt 키코드 99 또는 Ctrl 키코드 119에 함수들을 연결시키는 짓은 무의미하다.)

이상한 키가 붙어있는 키보드를 사용하는 경우, 리눅스에서는 그 키에 대해서는 아무런 키코드로 발생시키지 않는다. (어쩌면 "알지 못하는 키코드" 같은 메세지를 보낼 지도 모른다.) 커널 버전 1.1.63 이후의 사용자라면 setkeycodes(1)을 사용하여 커널에게 이 키에 대한 정보를 말해 줄 수 있다. 하지만 X에서는 여전히 사용할 수 없을 것이다. 일단 setkeycodes로부터 키코드를 받고 나면 loadkeys를 사용하여 어떤 기능과 연결시킬 수 있다.

제 15 절 loadkeys와 xmodmap의 사용예

키보드상의 Caps Lock과 Ctrl 키를 바꾸려면 아래와 같이 하면 된다. (keymaps 0-15를 사용한다고 가정했다. `dumpkeys | head -1`로 확인해볼 수 있다.)

```
% loadkeys
keymaps 0-15
keycode 58 = Control
keycode 29 = Caps_Lock
%
```

X 환경에서만 이것들을 바꾸려면 다음과 같이 하면 된다.

```
% xmodmap .xmodmaprc
```

여기서 `.xmodmaprc`에는 아래와 같은 내용이 들어있어야 한다.

```
remove Lock = Caps_Lock
remove Control = Control_L
keysym Control_L = Caps_Lock
keysym Caps_Lock = Control_L
add Lock = Caps_Lock
add Control = Control_L
```

도데체 여기에 있는 키 번호란 것이 무엇일까? Backspace는 리눅스에서는 14이고 X에서는 22라니? 뭐 별거 아니고 임의로 붙였다고 보면 된다. 리눅스에서 사용되는 키 번호를 보고 싶으면 showkey(1)를 사용하면 되고, X에서는 비슷한 것으로 xev(1)를 사용하면 된다. 종종 X에서 사용하는 번호는 리눅스에서 사용하는 번호에 8을 더하면 된다.

사람들이 종종 바꾸기 좋아하는 것이 기능키이다. F12를 누르면 "emacs "라는 문자열이 생기도록 하려면 아래와 같이 하면 된다.

```
% loadkeys
keycode 88 = F12
string F12 = "emacs "
%
```

좀 더 직접적인 방법으로는 다음과 같은 것이 있다. (i) showkey(1)를 사용하여 바꾸고자 하는 키의 키코드를 알아낸다. (ii) 현재 keymap을 저장하고 복사본을 만들어 다음과 같이 편집한다.

```
% dumpkeys > my_keymap
% cp my_keymap trial_keymap
% emacs trial_keymap
% loadkeys trial_keymap
%
```

테이블 형식은 `dumpkeys`의 출력을 보면 충분히 추측할 수 있고 keytables(5)에 잘 문서화되어 있다. 새로운 keymap 기능을 원한다면 `/etc/rc.local`에 아래의 내용을 넣어 호출하여 부팅시에 자동적으로 실행되게 하면 된다.

```
loadkeys my_new_keymap
```

변경키를 바꾸게 되면 혼란스러워짐을 유의하라. 대개 초심자들이 잘 빠지는 함정은 숙련자가 되어야만 빠져나오는 방법을 알게 된다.

keymaps에 대한 디폴트 디렉토리는 /usr/lib/kbd/keytables이다. keymaps에 대한 디폴트 확장자는 .map이다. 따라서 loadkeys uk라고 하면 아마도 /usr/lib/kbd/keytables/uk.map를 로드하게 될 것이다.

(내가 쓰는 머신에서는) /dev/console은 /dev/tty0에 심볼릭 링크되어 있다. 그리고 커널은 /dev/tty0를 현재 VT와 같은 것으로 간주한다. XFree86 1.3은 /dev/tty0의 소유자를 바꾸지만 종료한 후에 이것을 재설정하지는 않는다. 따라서 loadkeys 또는 dumpkeys는 말을 안들을 수도 있다. 왜냐하면 다른 사용자가 /dev/tty0소유하고 있을 수 있기 때문이다. 이런 경우 X를 먼저 실행시켜 보라. 콘솔에서가 아니면 (그리고 수퍼유저가 아니면) 키보드 매핑은 바꿀 수 없음을 유의하자.

15.1 '오직 한 손가락으로만 타이핑할 수 없을 때'

"Shift, Ctrl, Alt 키를 토글 형식으로 만들 수 있을까?"

할 수 있다. 아래와 같이 명령을 내리자.

```
% loadkeys
keymaps 0-15
keycode 29 = Control_Lock
keycode 42 = Shift_Lock
keycode 56 = Alt_Lock
%
```

이렇게 하고 나면 왼쪽 Control, Shift, Alt 키가 토글 형식으로 반응할 것이다. 여기에 사용되는 번호는 showkey를 사용하면 볼 수 있다. (보통 29, 97, 42, 54, 56, 100이 각각 왼쪽, 오른쪽 control, shift alt 키에 해당한다.) 또한 각각의 기능은 Control_Lock, Shift_Lock, Alt_Lock, ALtGr_Lock이다.

"'sticky' 변경키는 가능한가?"

커널 버전 1.3.33 이후에는 'sticky' 변경키를 인식한다. 'sticky' 변경키는 다음에 눌려진 키에 작용하는 것을 말한다. 예를 들어, 대문자 'A'를 치기 위해 3심볼 시퀀스 Shift_Lock a Shift_Lock을 쳐야 하는 곳에서 이 키를 쓰면 2심볼 시퀀스 SShift_Lock a만으로 해결된다. kbd 패키지 0.93보다 낮은 버전에는 아직 이러한 sticky 변경에 대한 코드가 포함되어 있지 않다. 따라서 이 키의 16진수 코드를 호출해야만 사용할 수 있다.

```
% loadkeys
keymaps 0-15
keycode 54 = 0x0c00
keycode 97 = 0x0c02
keycode 100 = 0x0c03
%
```

이렇게 하면 오른쪽 Shift, Ctrl, Alt 키가 해당 왼쪽 키들의 sticky 형식으로 반응할 것이다. >0.93부터는 아래와 같은 명령으로 같은 효과를 낼 수 있다.

```
% loadkeys
keymaps 0-15
keycode 54 = SShift
keycode 97 = SCtrl
keycode 100 = SAlt
%
```

이렇게 하면 Ctrl-Alt-Del을 한 손으로 입력할 수 있게 된다.

이 예에서 keymaps 줄은 사용자가 사용하고 있는 모든 keymaps을 적어야 한다. 사용하고 있는 keymaps을 확인하려면 아래와 같이 명령을 내리면 된다.

```
% dumpkeys | head -1
```

제 16 절 비디오 모드 바꾸기

내가 아는 한 해상도를 바꾸는데는 6가지 방법이 있다.

1. 컴파일 할 때: /usr/src/linux/Makefile에서 아래와 같은 라인을 바꾼다.

```
SVGA_MODE=      -DSVGA_MODE=NORMAL_VGA
```

1A. 컴파일 후, rdev -v를 사용하라. 상당히 위험한 해킹법이지만 분명히 잘 된다.

2. 부팅시: lilo 환경설정 화일에 vga=ask라는 줄을 추가하면 lilo는 부팅시 사용자가 원하는 비디오 모드를 물어볼 것이다. 여러번 해봐서 가장 맘에 드는 것을 알아냈으면 vga=맘에드는번호 로 바꿔 넣어라.

3. 실행중에: A. resizecons 명령을 사용한다. (이것은 VT_RESIZE ioctl에 대한 아주 원시적인 wrapper이다.) B. SVGATextMode 명령을 사용한다. (이것이 VT_RESIZE ioctl에 대한 좀 덜 원시적 wrapper이다.)

4. "콘솔에서"가 아닐 때: dosemu를 사용중이거나 svgalib를 사용하는 프로그램을 실행중일 때 등등, 이런 경우 콘솔 드라이버가 인식하지 못하면서 하드웨어 비디오 모드를 바꿀 수 있다. 때때로 이 방법은 resizecons 또는 SVGATextMode 설정을 할 때 유용한 경우가 있다. dosemu를 실행하고 적당한 비디오 모드에서 돌아가는 DOS 프로그램을 실행시킨다. 그 다음 (다른 VT에서 명령을 내려서) 모든 비디오 하드웨어 레지스터의 내용을 덤프한다. 여기에 나온 내용으로 resizecons이나 SVGATextMode에서 요구하는 사항에 이용한다. 때때로 비디오 모드가 불안정한 상태에 놓이게 되는 경우가 있다. dosemu 기동시에 비디오 모드를 설정하는 것이 BIOS에 의존하기 때문인데, 이럴 경우 (kill -9로) dosemu를 죽이면 간단히 원래 상태로 돌아온다.

16.1 resizecons를 사용하는 방법

svgalib를 얻어서 restoretexmode 프로그램을 컴파일한다. (lilo 환경설정 화일에 vga=ask를 사용하여) 모든 가능한 비디오 모드로 부팅해보고, 비디오 하드웨어 레지스터 내용을 CxR(여기서 C는 컬럼수, R은 열수)라는 이름의 화일에 저장한다. 예를 들어 80x25, 132x44 이런 식의 이름으로 저장한다. /usr/lib/kbd/videomodes에 이 화일을 넣어둔다. 이렇게 하면 resizecons 132x44라는 명령으로 비디오 모드가 바뀔 것이다. (이것에 대하여 알 필요가 있는 모든 프로세스에 SIGWINCH를 보내고 필요하다면 다른 폰트도 로드한다.)

현재, resizecons는 바뀌기 전과 바뀐 후의 콘솔에 대한 메모리가 모두 충분한 경우에만 성공적으로 작동한다.

제 17 절 키보드 갱신율 바꾸기

기동시에 리눅스 커널은 갱신율을 최대값으로 설정한다. 대부분의 키보드에서 이 값은 문제가 없다. 하지만 어떤 키보드들은 아무리 짧게 눌러서 같은 글자가 서너개 이상 찍히는 경우가 있다. kbdrate(8)을 사용하여 갱신율을 바꾸든지, 이 방법이 효과가 없으면 /usr/src/linux/[arch/i386/]boot/setup.S에서 아래 부분을 없애거나 편집한다.

```
! set the keyboard repeat rate to the max

    mov     ax,#0x0305
    xor     bx,bx           ! clear bx
    int     0x16
```

제 18 절 스크롤

화면을 스크롤하는 방법은 두가지다. 첫째, '하드 스크롤'이라고 하는 것인데, 비디오 메모리에 텍스트를 남겨두고, 보는 기준점을 바꾸는 방법이다. 이 방식은 아주 빠르다. 둘째, '소프트 스크롤'이라고 하는 것인데, 화면에

표시된 텍스트를 전부 위 또는 아래로 이동시키는 방법이다. 이것은 물론 느리다. 커널 콘솔 드라이버는 비디오 메모리의 꼭대기에서 텍스트를 쓰기 시작하여 바닥에 도달할 때까지 계속한다. 바닥에 도달하면 이 내용을 위에 복사한 후에 같은 일을 계속한다. 항상 하드 스크롤을 사용하여 화면에 보여준다. 디폴트 keymap을 사용하는 경우, Shift-PageUp (회색 PageUp)을 사용하면 비디오 메모리의 꼭대기로 스크롤할 수 있고, Shift-PageDown (회색 PageDown)을 사용하면 아래쪽으로 스크롤할 수 있다. 따라서, 스크롤할 수 있는 양은 사용자가 할당받게 되는 비디오 메모리의 양으로 제한된다. 더 많은 양을 스크롤하려면 텍스트 버퍼를 사용하는 프로그램을 사용해야 한다. `less`나 `screen` 등이 이런 프로그램인데 디스크상에 버퍼를 만들어 두어 이것을 이용하여 이전에 했던 것을 다시 볼 수 있다. (xterm에서 가능한 스크롤 양을 설정하려면 `.Xresources` 안에 `XTerm*saveLines: 2500`과 같은 형식으로 추가하면 된다.)

가상 콘솔을 바꾸면, 이전 VT의 화면 내용은 커널 메모리로 복사되고 새로운 VT의 화면 내용이 커널 메모리에서 비디오 메모리로 복사된다. 비디오 메모리의 모든 정보를 복사하는 것이 아니라 단지 보이는 스크린만을 복사하기 때문에 콘솔을 전환하면 스크롤 정보를 잃게 된다.

때때로, 하드 스크롤이 부적당한 경우가 있다. 예를 들어 하드웨어가 화면 표시 기준점을 바꾸지 못하는 경우에 그렇다. 대표적인 예가 Braille 머신인데, 이 머신은 항상 Braille의 비디오 메모리의 꼭대기만 사용한다. 커널 부트타임 옵션으로 `no-scroll`이 있는데 여기서 콘솔 드라이버가 하드 스크롤을 사용하지 않도록 할 수 있다. bootparam(7)을 참고하면 도움이 될 것이다.

제 19 절 스크린 세이브

`setterm -blank` *nn*이라고 명령을 내리면 콘솔 드라이버는 *nn* 분동안 사용하지 않으면 화면을 비운다. (*nn* = 0라고 하면 스크린 세이브 기능이 꺼진다. 구버전 커널에서는 이것이 다음 키보드 인터럽트 후에 효과가 나타나는 경우도 있었다.)

xset(1)의 `s` 옵션을 사용하면 X의 스크린세이브 파라미터를 설정할 수 있다. `xset s off`sms 스크린 세이버를 끄고, `xset s 10`는 10분 후에 화면을 검게 만든다.

비디오 하드웨어 파워 세이빙 모드는 `/usr/src/linux/drivers/char/vesa_blank.c`의 시작 부분에 있는 `setvesablank` 프로그램을 사용하면 켜거나 끌 수 있다.

제 20 절 스크린 덤프

현재 디렉토리에 `screen.dump`라는 화일로 `/dev/tty`*N*의 화면 내용을 덤프하려면 `setterm -dump` *N* 라고 하면 된다. setterm(1)을 참조하라.

`/dev/tty`*N* 스크린의 현재 내용은 `/dev/vcs`*N* 디바이스를 사용하면 액세스할 수 있다. (여기서 'vcs'는 가상 콘솔 스크린의 약자이다.) 이것을 사용하면 콘솔 스크린의 오른쪽 위에 현재 시간을 표시하는 시계 프로그램을 실행시킬 수 있다. (`kbd-0.95.tar.gv`에 있는 `vcstime` 프로그램을 참조하라.) 단지 내용만을 덤프하려면, `cat /dev/vcs`*N*이라고 해도 된다. 이러한 디바이스 화일`/dev/vcs`*N*에는 newline도 색과 같은 속성도 없다. 좀 더 나은 프로그램으로는 `/dev/vcsa`*N*가 있다. (vcsa: 'virtual console screen with attributes') 이 프로그램은 행과 열의 수, 커서의 위치를 담은 헤더로 실행시킨다. vcs(4)를 참조하라.

제 21 절 VT100의 특성 - 어플리케이션 키 모드

: 때때로 커서 키나 키패드 키가 이상한 코드를 만든다.

터미널이 어플리케이션 커서 키 모드에 있으면 커서 키는 Esc 0 x를 발생시키고 그렇지 않은 경우에는 Esc [x를 발생시킨다. (여기서 x는 A,B,C,D중 하나다.) 어떤 프로그램은 터미널을 어플리케이션 커서 키 모드로 두는데, 이 때 사용자가 `kill -9`로 죽이거나 프로그램이 비정상 종료를 하면 모드가 되돌아 오지 않게 된다. 이 경우

```
% echo -e '\033c'
```

라고 하면 현재 VC의 특성을 모두 재설정할 것이다. 단지 커서 어플리케이션 키 모드를 바꾸려면 다음과 같이 해도 된다. (설정)

```
% echo -e '\033[?1h'
```

(해제)

```
% echo -e '\033[?1l'
```

터미널이 어플리케이션 키패드 키 모드에 있을 때는 키패드 키는 y를 누른 경우 Esc O y를 발생시키고 이 모드에 있지 않으면 Esc [y ~를 발생시킨다. 어플리케이션 키패드 키 모드를 설정하려면

```
% echo -e '\033='
```

해제하려면

```
% echo -e '\033>'
```

라고 하면 된다.

제 22 절 하드웨어 비호환성

몇몇 사람들이 플로피 디스크가 작동중일 때는 입력한 문자들이 사라지는 것을 알아냈다. Uni-486WB 마더보드에 이런 문제가 있는 것 같다. (확인하기 위하여 다음의 간단한 양식으로 제게 메일을 보내 주세요.(aeb@cwi.nl) [yes, I have the same problem], deny [no, nothing wrong with my Uni-486WB], modify [My Xyzzy machine has the same problem].)

Tjalling Tjalkens (tjalling@ei.ele.tue.nl)도 "AMD 486DX2-66 CPU를 탑재한 GMB-486 UNP Vesa 마더보드"에서 플로피를 사용하는 동안 키입력의 일부가 사라지는 현상, 플로피 테이프 streamer (Conner C 250 MQ)를 사용하는 동안에는 상당한 양의 키입력이 사라지는 현상을 보고했다.

산발적인 lockup을 경험한 사람들도 있다. - 주로 하드 디스크나 다른 I/O를 사용중인 경우이다.

Ulf Tietz (ulf@rio70.bln.sni.de): '나의 마더보드가 너무 빠르게 조정되어 있어서 비슷한 문제가 있었다. 그래서 모든 시간설정 (CLK, 대기 상태 등등)을 재설정하여 좀 더 사용하기 편한 값으로 바꾸었더니 문제가 해결되었다.'

Bill Hogan (bhogan@crl.com): 'AMI BIOS를 사용한다면, Gate A20 에뮬레이션 파라미터를 "(그 옵션이 있다면) chipset"으로 설정해야 할 지도 모른다. 이 값을 다른 것("fast", "both", "disabled")으로 설정하면 종종 키보드 lockup에 걸리곤 했다.'

제 23 절 저작권(Copyright)

Additions and corrections are welcome. Andries Brouwer - aeb@cwi.nl