

viを使つての編集

vi はコマンドラインエディタです。初めて使う学生が大半だと思います。

長所：

- ※ GUI が起動できない「トラブル時」でも起動ファイルの編集などができるため、システムをセットアップしたり、クラッシュしたシステムを復旧したりする際にも、活用できる。
- ※ キーボード以外を使用しないため、マウス操作のために指を離すことがなく、キー操作だけで「コピー&ペースト」などが可能なため、慣れてくるとマウスを使うキー入力よりもさらに高速なファイル編集が可能となる。

短所：

- ※ 頭の中で「編集集中の状態」をある程度把握していないと、うまく操作できない。

という長所、短所があります。

ここでは、キーボードを日本語対応する操作を前提に、vi の使用方法を説明します。

まず、root でログインします。root と入力してから **[Enter]** を押してください。
以下、コマンド入力する時には、最後に必ず **[Enter]** を押します。(いちいち書きません。)

次に、cd /usr/src/kernel/keymaps

で、ソースコードの、keymaps のディレクトリに移動します。

ここで、ls と入力して

```
Minix Release 2 Version 0.4
bochs-minix.local.net login: root
# cd /usr/src/kernel/keymaps
# ls
Makefile      german.src    latin-am.src  scandinavm.src  us-std.src
french.src     italian.src   olivetti.src  spanish.src     us-swap.src
genmap.c      japanese.src  polish.src    uk.src
# -
```

ディレクトリの中身を見てみます。ドイツ語、スカンジナビア語、フランス語など様々な言語に並んで、japanese.src があることに気づきます。

ここで、ひとつ上のディレクトリ (/usr/src/kernel) に移動します。

cd .. と入力します。

ls と入力すると、ファイル名がリスト表示されます。

```
# ls
Makefile      german.src    latin-am.src  scandinav.n.src  us-std.src
french.src    italian.src   olivetti.src  spanish.src       us-swap.src
genmap.c      japanese.src  polish.src    uk.src

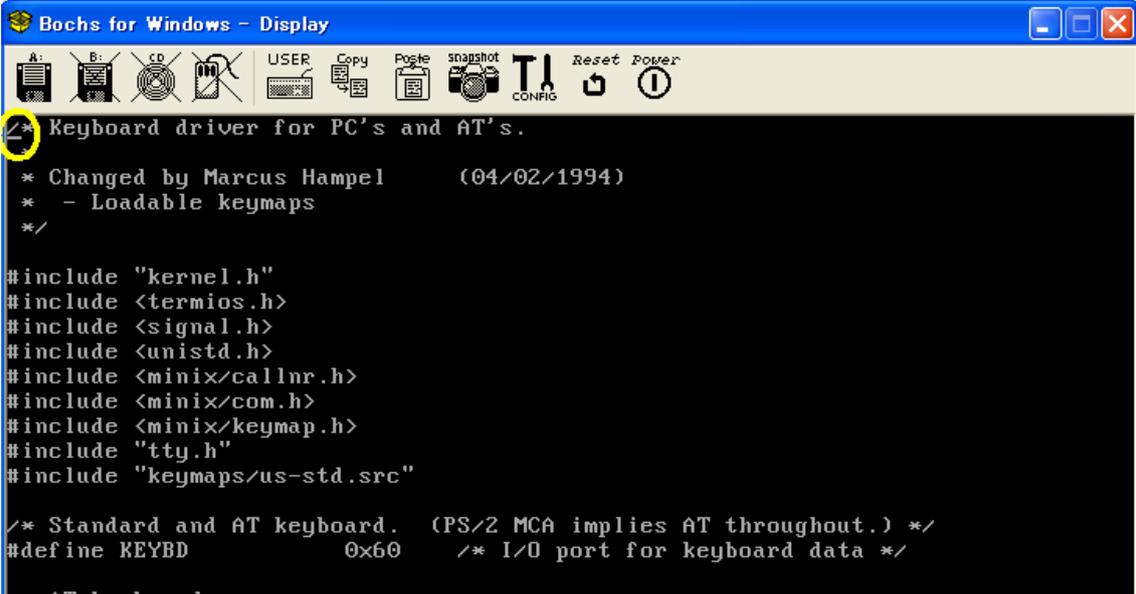
# cd ..
# ls
3c503.c        driver.h      main.c         pci_table.c204    sb16_dsp.c
3c503.h        drvlib.c      memory.c       pci_via.h          sb16_mixer.c
Makefile      drvlib.h      misc.c         printer.c          sconst.h
aha1540.c     esdi_wini.c   mpx.s          proc.c             start.c
assert.h      exception.c   mpx386.s       proc.h             system.c
at_wini.c     fatfile.c     mpx88.s        protect.c          table.c
bios_wini.c   floppy.c      ne2000.c       protect.h          tty.c
clock.c       glo.h         ne2000.h       proto.h            tty.h
console.c     i8259.c       pci.c          pty.c              type.h
const.h       kernel.h      pci.c204       rs232.c            wdeth.c
dmp.c         keyboard.c    pci.h          rtl8029.c          wdeth.h
dosfile.c     keymaps       pci_amd.h      rtl8139.c          xt_wini.c
dp8390.c      klib.s        pci_intel.h    rtl8139.c204
dp8390.h      klib386.s     pci_sis.h      rtl8139.h
driver.c      klib88.s      pci_table.c    sb16.h
# vi keyboard.c_
```

ここで、`keyboard.c` というソースファイルを編集します。

vi keyboard.c と入力します。

`vi` では、「文字入力モード」と、「移動・編集モード」を相互に切り替えて使いますが、画面上には「どちらのモード」になっているかが表示されません。つまり、頭の中でこの相互を切り替えることとなります。

`vi`



```
Bochs for Windows - Display
A: B: CD USER Copy Paste Snapshot CONFIG Reset Power
/*
 * Keyboard driver for PC's and AT's.
 *
 * Changed by Marcus Hampel      (04/02/1994)
 * - Loadable keymaps
 */
#include "kernel.h"
#include <termios.h>
#include <signal.h>
#include <unistd.h>
#include <minix/callnr.h>
#include <minix/com.h>
#include <minix/keymap.h>
#include "tty.h"
#include "keymaps/us-std.src"

/* Standard and AT keyboard. (PS/2 MCA implies AT throughout.) */
#define KEYBD      0x60      /* I/O port for keyboard data */
```

わかりにくいですが、左上にカーソル (`_`) が現れています。

このカーソルを移動させるためには、j、k、l、hのキーを使うか、または、慣れていない場合には上下左右のカーソルキーを使用します。

jを14回押して、14行下に移動します。

または、14jと入力します。(入力した文字は画面に表示されません。) こうすると、一気に14行下にジャンプします。

下向き矢印を14回押しても構いません。

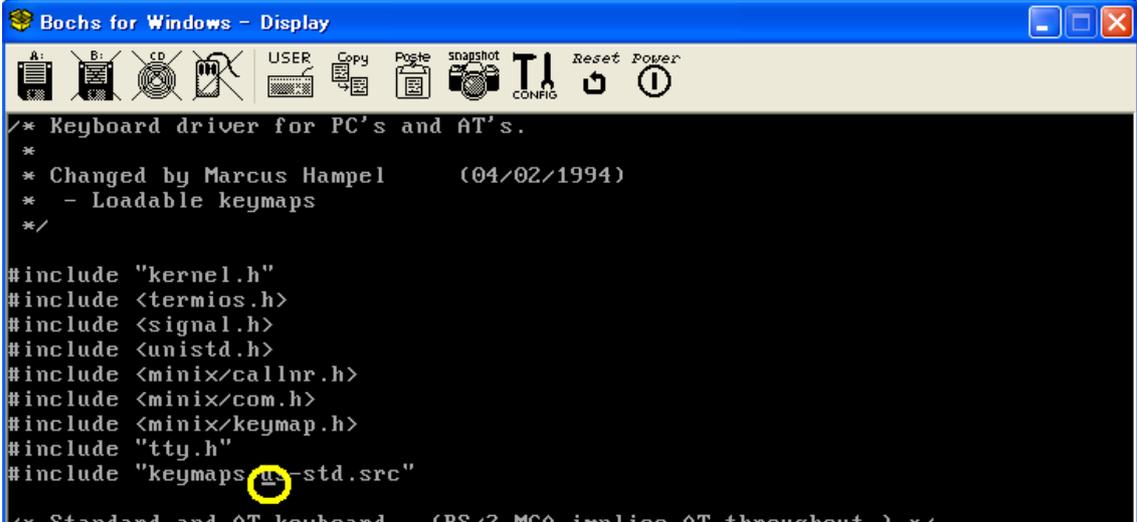
次に、lのキーを18回押して右に移動します。

または、右向き矢印キーを18回押します。

すでに気づいている学生がいると思いますが、18Lと入力しても右に18文字移動します。

さらに簡単な方法があります。wのキーを押すと「ワード」単位で右に移動しますから、wを5回押しても同じ位置に移動します。5wと入力しても同じです。

こうして移動すると、カーソルが以下の位置に移動しているはずですが。



```
Bochs for Windows - Display
A: B: CD USER Copy Paste Snapshot CONFIG Reset Power
/* Keyboard driver for PC's and AT's.
 *
 * Changed by Marcus Hampel (04/02/1994)
 * - Loadable keymaps
 */
#include "kernel.h"
#include <termios.h>
#include <signal.h>
#include <unistd.h>
#include <minix/callnr.h>
#include <minix/com.h>
#include <minix/keymap.h>
#include "tty.h"
#include "keymaps_us-std.src"
/* Standard and AT keyboard (PS/2, MCA implies AT throughout) */
```

ここで、us-stdと書かれている文字を削除します。カーソル位置の文字を1文字削除するキーは、xです。つまり、xを6回押すと、このus-stdの部分削除されます。

6xと入力しても、同じ結果になります。

別の方法もあります。dwと入力すると「ワード単位」での削除になりますから、ここでは、dwを3回繰り返す(dwdwdwと入力する)と、us-stdの部分削除されます。3dwでも同様に、3ワードが削除され、us-stdが削除されます。

うっかり消してしまった場合には、u (undo;アンドゥ)を押すと、一操作のみでしたら取り消すことができます。

```
#include <minix/keymap.h>
#include "tty.h"
#include "keymaps/_.src"
```

ここで画面は上のようになっているはずです。

今度は、この位置に japanese を入力します。

ここで、これまでの「移動・編集モード」から、「文字挿入モード」に切り替えます。

「文字挿入モード」に切り替えるには、i (insert)のキーを押します。「文字挿入モード」から抜け出すには、[ESC]のキーを押します。

一旦、iのキーを押して、モードを切り替えてから japanese と入力し、ESC で「挿入モード」から抜け出します。([ESC] キーは、画面左上にあります。)

ここでは、入力した内容を画面でも見ることができます。

```
#include <minix/callnr.h>
#include <minix/com.h>
#include <minix/keymap.h>
#include "tty.h"
#include "keymaps/japanese.src"

/* Standard and AT keyboard. (PS/2 MCA implies AT)
#define KEYBD 0x60 /* I/O port for keyboard
```

無事入力されると、上記のようになっているはずです。

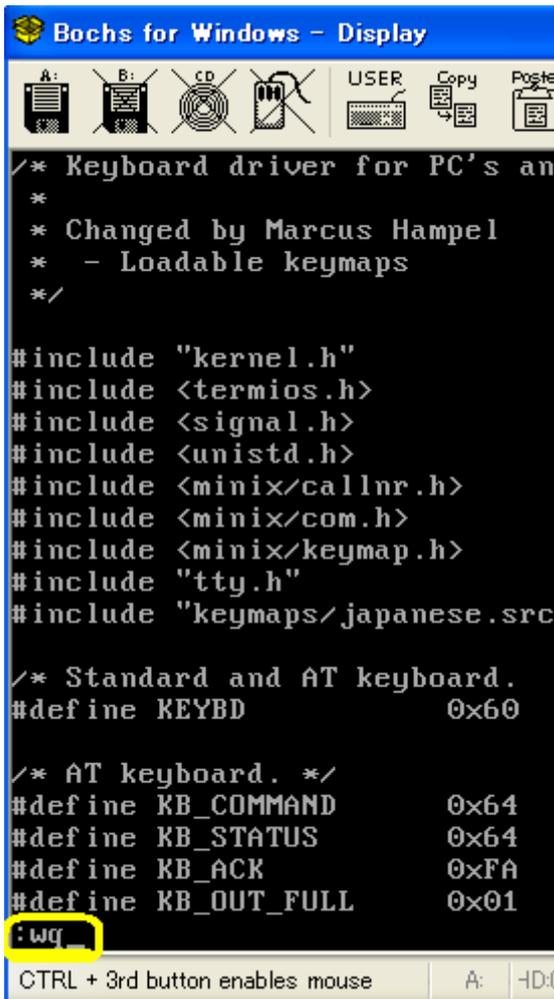
ここで、今度は「編集した内容」を保存します。

vi エディタを「コマンド入力」モードに切り替えます。

コマンド入力に切り替えるためには、「:」のキーを押すのですが、現時点ではまだキーボードが日本語対応になっていません。つまり、「:」のキーは本来の日本語の位置とは別の場所にあります。

us-std キーボードの「:」が日本語キーボードのどの位置に対応するかと言えば、「+」のキーです。MINIX にログインするときに、=の入力で「^」のキーを押しましたが、「:」を入力するためには、「+」のキーを押さなければなりません。つまり、シフトを押しながら「;」のキー (Lのキーの右隣です) を押すと、画面に「:」が表示されます。

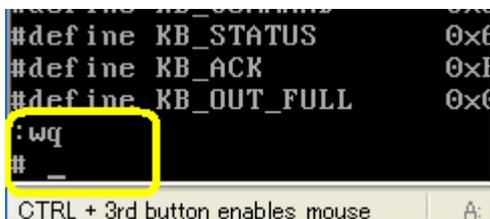
次に、ファイルの「上書き保存」のコマンド (w) 入力し、(一旦ここで、ENTER して保存してから、改めて「:」を押しても構いませんが、) 引き続き「vi の終了」コマンド (q) を押します。



```
Bochs for Windows - Display
A: B: CD USER Copy Poste
/* Keyboard driver for PC's an
*
* Changed by Marcus Hampel
* - Loadable keymaps
*/
#include "kernel.h"
#include <termios.h>
#include <signal.h>
#include <unistd.h>
#include <minix/callnr.h>
#include <minix/com.h>
#include <minix/keymap.h>
#include "tty.h"
#include "keymaps/japanese.src
/* Standard and AT keyboard.
#define KEYBD 0x60
/* AT keyboard. */
#define KB_COMMAND 0x64
#define KB_STATUS 0x64
#define KB_ACK 0xFA
#define KB_OUT_FULL 0x01
:wq
CTRL + 3rd button enables mouse A: -D(
```

画面では、上のようになります。「:」を受け付けると、「コマンド入力行」は画面左下に表示されます。そこで、画面で確認しながらwqなどを入力します。

無事終了すると、「#」（MINIXのrootモードでのプロンプト）が表示されます。



```
#define KB_STATUS 0x6
#define KB_ACK 0xF
#define KB_OUT_FULL 0x0
:wq
# _
CTRL + 3rd button enables mouse A:
```

さて、ここでコンパイルです。

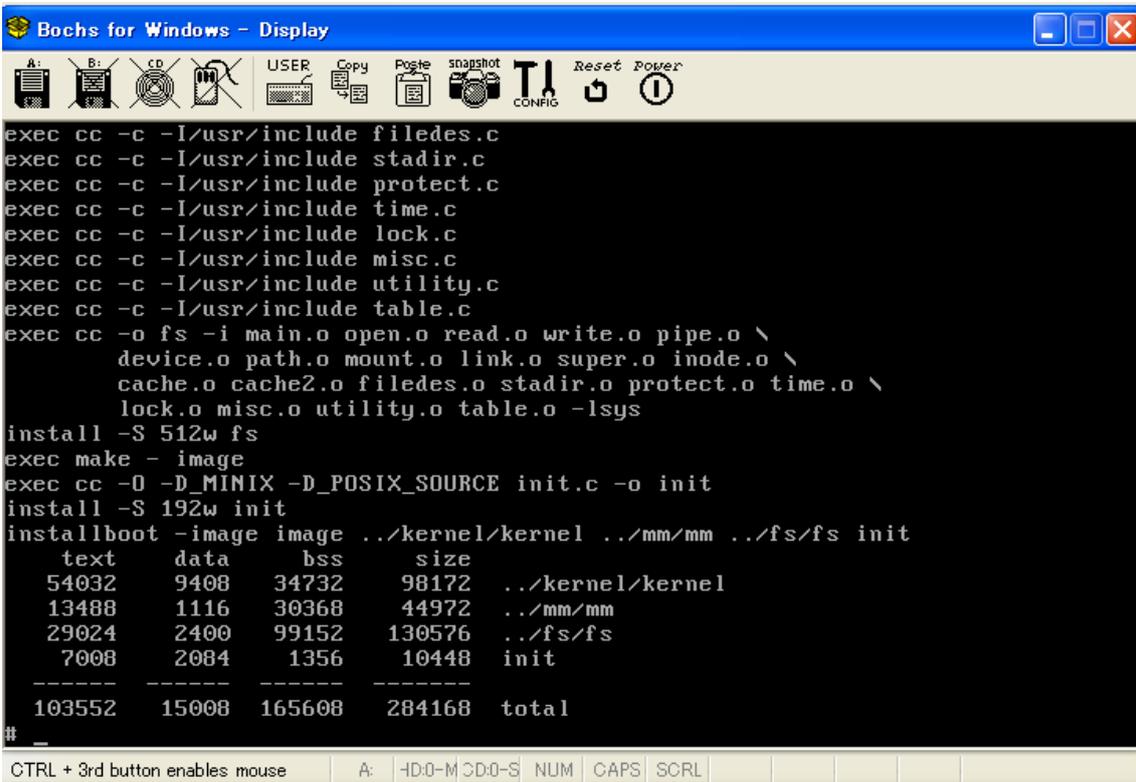
`cd ../tools` と入力して、ひとつ上のフォルダに移動し、`make` と入力します。そうすると、コンパイルが開始されます。

```
/* AT keyboard. */
#define KB_COMMAND      0x64      /* I/O
#define KB_STATUS      0x64      /* I/O
#define KB_ACK         0xFA      /* key
#define KB_OUT_FULL    0x01      /* sta

:wq
# cd ../tools
# make
cd ../kernel && exec make -
exec cc -c -I/usr/include mpx.s
exec cc -c -I/usr/include start.c
exec cc -c -I/usr/include protect.c
exec cc -c -I/usr/include klib.s
```

CTRL + 3rd button enables mouse A: HD0-M CD0-S N

コンパイルが開始されると、上記のように「コンパイルの途中経過」が表示されます。これが、結構時間がかかりますので、授業開始の際に、us-std の状態のままで一旦「コンパイル」を実行するように指示を出すかも知れません。授業中の指示に従ってください。（聞いていない場合には、20分くらいコンパイルが終わるのをじっと待つこともありますから、聞いていなかった自分が悪いと思ってあきらめて下さい。）



```
exec cc -c -I/usr/include filedes.c
exec cc -c -I/usr/include stadir.c
exec cc -c -I/usr/include protect.c
exec cc -c -I/usr/include time.c
exec cc -c -I/usr/include lock.c
exec cc -c -I/usr/include misc.c
exec cc -c -I/usr/include utility.c
exec cc -c -I/usr/include table.c
exec cc -o fs -i main.o open.o read.o write.o pipe.o \
device.o path.o mount.o link.o super.o inode.o \
cache.o cache2.o filedes.o stadir.o protect.o time.o \
lock.o misc.o utility.o table.o -lsys
install -S 512w fs
exec make - image
exec cc -O -D_MINIX -D_POSIX_SOURCE init.c -o init
install -S 192w init
installboot -image image ../kernel/kernel ../mm/mm ../fs/fs init
text      data      bss      size
54032    9408    34732    98172    ../kernel/kernel
13488     1116   30368    44972    ../mm/mm
29024     2400   99152   130576    ../fs/fs
7008      2084    1356    10448    init
-----
103552    15008   165608   284168    total
#
```

CTRL + 3rd button enables mouse A: HD0-M CD0-S NUM CAPS SCRL

コンパイルが完了すると、上のような画面になっているはずです。

ここで、cp image /minix/2.0.4 と入力します。

コマンドの意味は、今コンパイルしたばかりの「OS のバイナリーイメージ (image) のファイルを、/minix というディレクトリの下の 2.0.4 というファイルにコピーする、」です。ここで、2.0.4 というファイルはすでに存在しています。が、MINIX の OS では、「上書き」についての警告は一切ありません。

一旦、システムを抜け出します。

shutdown -h now

と入力します。

```
# cp image /minix/2.0.4
# shutdown -h now

Broadcast message from root@bochs-min
Sun Apr 22 23:41:42 2007...

The system will shutdown NOW

System Halted
d0p0>_
```

CTRL + 3rd button enables mouse A: -D:0-M:0D:0-S

(この状態で、POWER のボタンを押して、仮想コンピュータの電源を切ると、次にブートした際に、「ふだんと違うメッセージ」は表示されません。)



ここで、boot と入力します。

```
The system will shutdown NO

System Halted
d0p0>boot_
```

これで起動したら、root でログインした後で、日本語キーボードのままの文字が表示されていることを確認してください。

```
Multiuser startup in progress.
Starting daemons: update cron.

Minix Release 2 Version 0.4

bochs-minix.local.net login: root
# --;+***:_
```

この、キーボードの配置変換の切り替えが完了したら、第3回の課題はOKです。

おまけ「ビープ音の停止について」

viの編集などで、パソコンによっては大音量でビープ音が鳴ったことと思います。カーソルが移動できない状態で左向きのキーを押したり、コマンドエラーなどがある度に、ビープ音が鳴動して、操作が一時停止していたと思います。

これが「うっとうしい」場合には、以下のような方法でbeepを停止できます。

cd /usr/src/kernel

で、カーネルのソースコードのページにジャンプする。

vi console.c

で、コンソール端末の処理プログラムファイルを開く。

:712

と入力して、712行目にジャンプする。

```
#include "tty.h"
#include "proc.h"

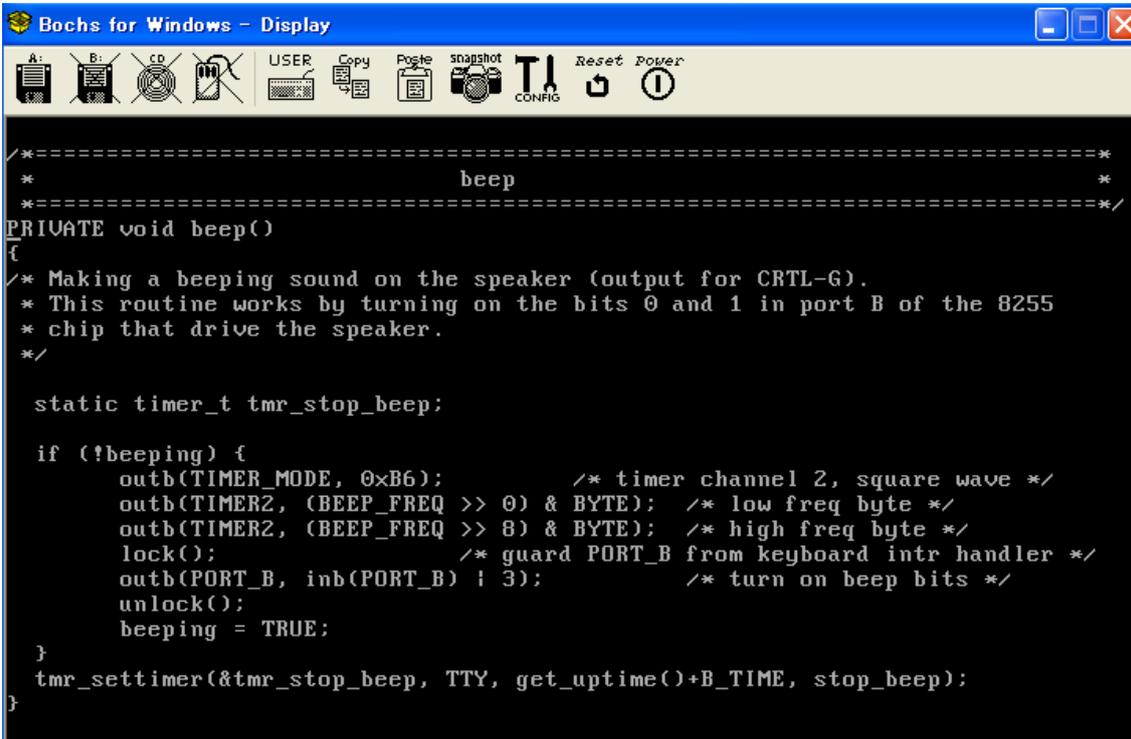
:712_
```

以下のような、beepの処理関数（javaで言うメソッド）が表示されます。

ここでは、outbで「音を出す」というコマンドを直接ハードウェアコントローラに送っています。このコマンドの中身を調べるためには、周辺LSIチップのコマンド一覧表を参照しなければなりません。

つまり、この次元の「プログラム」は、コンピュータのハードウェアをダイレクトに管理しているために、Windowsなどの端末で「音を出さない」設定をいくら行っても、まったく効果がないのです。Windowsなどをまったく介在させずに、直接システムの「ビープ音

源」に音を出させています。

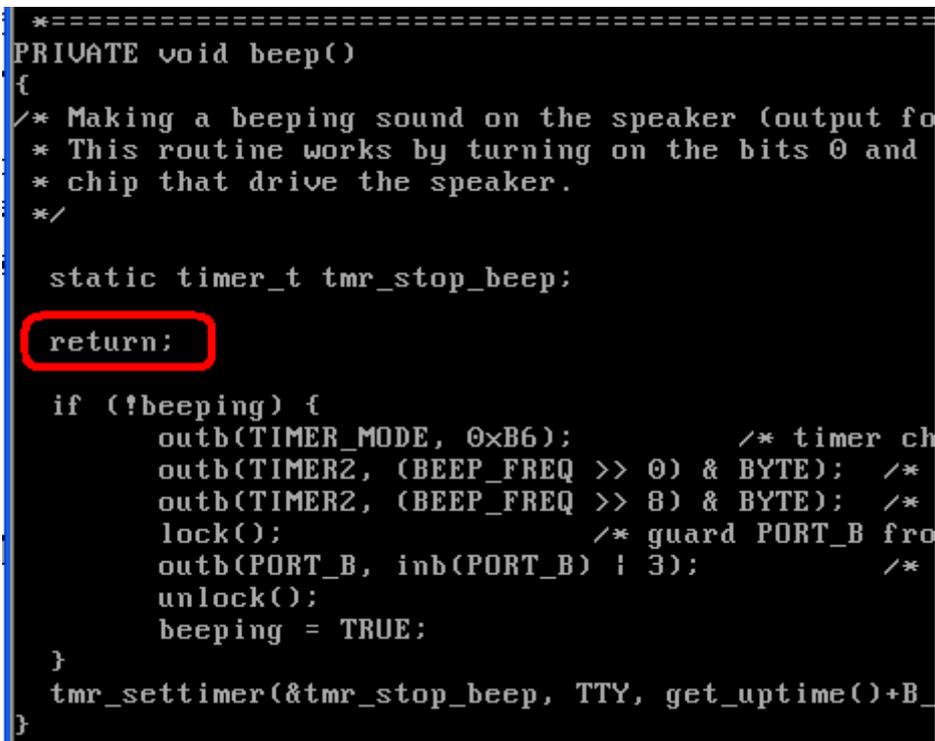


```
/*=====
 *
 *=====*/
beep
/*=====*/
PRIVATE void beep()
{
/* Making a beeping sound on the speaker (output for CRTL-G).
 * This routine works by turning on the bits 0 and 1 in port B of the 8255
 * chip that drive the speaker.
 */

static timer_t tmr_stop_beep;

if (!beeping) {
outb(TIMER_MODE, 0xB6); /* timer channel 2, square wave */
outb(TIMER2, (BEEP_FREQ >> 0) & BYTE); /* low freq byte */
outb(TIMER2, (BEEP_FREQ >> 8) & BYTE); /* high freq byte */
lock(); /* guard PORT_B from keyboard intr handler */
outb(PORT_B, inb(PORT_B) | 3); /* turn on beep bits */
unlock();
beeping = TRUE;
}
tmr_settimer(&tmr_stop_beep, TTY, get_uptime()+B_TIME, stop_beep);
}
```

BEEP_FREQ を変更して、音の出る時間を短くする、という「テクニック」もありますが、ここでは大胆に、ビープそのものを停止してしましましょう。



```
/*=====
PRIVATE void beep()
{
/* Making a beeping sound on the speaker (output fo
 * This routine works by turning on the bits 0 and
 * chip that drive the speaker.
 */

static timer_t tmr_stop_beep;

return;

if (!beeping) {
outb(TIMER_MODE, 0xB6); /* timer ch
outb(TIMER2, (BEEP_FREQ >> 0) & BYTE); /*
outb(TIMER2, (BEEP_FREQ >> 8) & BYTE); /*
lock(); /* guard PORT_B fro
outb(PORT_B, inb(PORT_B) | 3); /*
unlock();
beeping = TRUE;
}
tmr_settimer(&tmr_stop_beep, TTY, get_uptime()+B_
}
```

if のプログラム本体の記述が始まる前に、return;を入れてしまって、何もしないように

してしまうのです。

まずは、これで、MINIX システムが「静かに」なります。

(特に、これをやっておかないと、電車の中で大ひんしゅくを買うことになります。)

最後に cp image /minix/2.0.4 と入力するのは、キーボードの切り替えと同様です。

__(アンダースコア)と'|'(縦棒)のキー割付

大学の PC では、上記の編集だけで__ (アンダースコア) や、'|' 縦棒、および\ (バックスラッシュ) が使えるようになっているようですが、私のノート (ばかりではなく、大半のノートパソコン) では、うまく縦棒やバックスラッシュなどの重要な文字が表示できません。

かなり強引ですが、これらの割付を別のキーに行います。

いわゆる「パッチ」(つぎあて) をあてる方法ですが、vi の使用方法の練習だと思ってこの方法で行ってください。

cd /usr/src/kernel/keymaps でディレクトリを移動し、

vi japanese.src で、日本語キー割り当てのファイルを開く。

/125 と入力して、ファイル中の 125 という文字列を検索する。



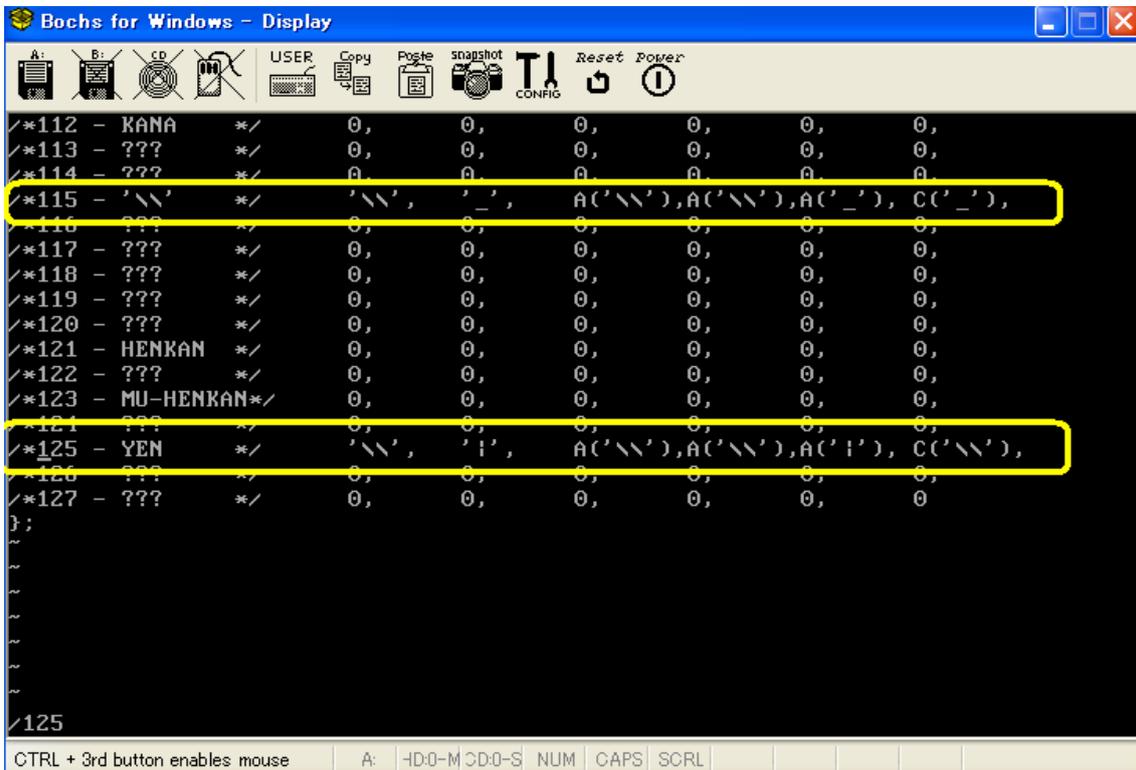
```
*      scan-code      keytop      effect in this keymap
*      -----
*      112(0x70)      KANA       (ignored)
*      115(0x73)      BackSlash  mapped to '\ ' and ' _ '
*      121(0x79)      HENKAN     (ignored)
*      123(0x7B)      MU-HENKAN  (ignored)
*      125(0x7D)      YEN       mapped to '\ ' and ' ! '
*/

#if (NR_SCAN_CODES != 0x80)
#error NR_SCAN_CODES mis-match
#endif

/125
```

最初はまず、プログラム先頭部のコメントの 1 2 5 にジャンプする。次を検索するために「n」(next) のキーを押す。

直接的には、[] も、[|] も、[\] も、画面からは入力できないので、すでにファイルの中に記載されているところからコピーを持ってくることにします。



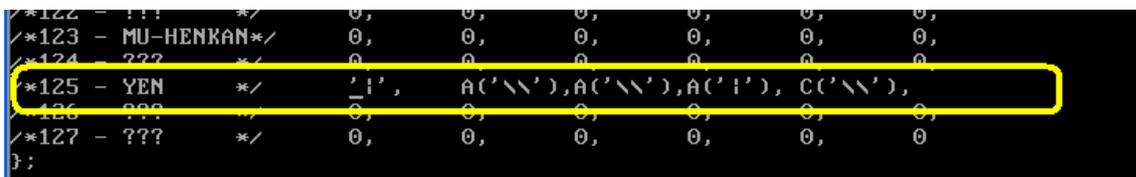
[w] (ワードジャンプ) を4回押して、'\\\'', の部分まで、カーソルを移動させる。

次に、[dw] (ワード削除) を行って、この部分を削除し、次に[u] (undo; 取り消し) して、削除した部分を元に戻す。

(dw と u) は、必ずワンセットで行ってください。元に戻らなくなります。

削除してから元に戻す、とは、何のためにこんな無駄な操作を行ったかと言えば、「削除」した際に「削除」した部分が「コピーバッファ」に取り込まれるため、次から「貼り付け」が可能になるためです。

一時削除しただけだと、以下のようになっています。このままにしないように、気をつけて下さい。



次に、/ 27と入力して27の文字列を検索させ、([n] (次) も使いながら、) 27のコードにジャンプします。

```

/* 26 - 'e' */ L('e'), 'E', A('e'), A('e'), A('E'), C('E'),
/* 27 - 'l' */ L('l'), 'L', A('l'), A('l'), A('L'), C('L'),
/* 28 - Enter */ C('M'), C('M'), CA('M'), CA('M'), CA('M'), C('J'),

```

この、A('L')、の部分、を、'\\"'に置き換えます。[A]の左側にカーソルを移動させて、[p]のキー（ペースト）を押します。そうすると、上記の「削除+undo」を行った際にコピーバッファに取り込まれた'\\"'の文字列がここに挿入されます。

```

/* 24 - 'o' */ L('o'), 'O', A('o'), A('o'), A('O'), C('O'),
/* 25 - 'p' */ L('p'), 'P', A('p'), A('p'), A('P'), C('P'),
/* 26 - 'e' */ L('e'), 'E', A('e'), A('e'), A('E'), C('E'),
/* 27 - 'l' */ L('l'), 'L', A('l'), A('l'), A('L'), C('L'),
/* 28 - Enter */ C('M'), C('M'), CA('M'), CA('M'), CA('M'), C('J'),
/* 29 - Ctrl */ CTRL, CTRL, CTRL, CTRL, CTRL, CTRL,
/* 30 - 'a' */ L('a'), 'A', A('a'), A('a'), A('A'), C('A'),

```

この状態の画面の一部です。左右両側の[Alt]キーに対応させるために、もう一度この[p]のキーを押す操作を行い、次に、A('L')、A('L')、を[x]のキーで削除して、以下のような表示にします。

```

/* 25 - 'p' */ L('p'), 'P', A('p'), A('p'), A('P'), C('P'),
/* 26 - 'e' */ L('e'), 'E', A('e'), A('e'), A('E'), C('E'),
/* 27 - 'l' */ L('l'), 'L', A('l'), A('l'), A('L'), C('L'),
/* 28 - Enter */ C('M'), C('M'), CA('M'), CA('M'), CA('M'), C('J'),
/* 29 - Ctrl */ CTRL, CTRL, CTRL, CTRL, CTRL, CTRL,
/* 30 - 'a' */ L('a'), 'A', A('a'), A('a'), A('A'), C('A'),

```

再び、/125でファイル下部にジャンプして、今度は、'l'の部分で[dw][u]を実行して、コピーバッファに取り込み、/27で画面上部に戻ってから、今度はA('l')、の部分、を置き換えます。

[p]を押して貼り付けを行ってから、A('l')、を削除します。27の行は、最終的に以下ようになります。

```

/* 25 - 'p' */ L('p'), 'P', A('p'), A('p'), A('P'), C('P'),
/* 26 - 'e' */ L('e'), 'E', A('e'), A('e'), A('E'), C('E'),
/* 27 - 'l' */ L('l'), 'L', A('l'), A('l'), A('L'), C('L'),
/* 28 - Enter */ C('M'), C('M'), CA('M'), CA('M'), CA('M'), C('J'),
/* 29 - Ctrl */ CTRL, CTRL, CTRL, CTRL, CTRL, CTRL,

```

同様の操作を繰り返して/115の行から、'_'をコピーして、

```

/*113 - ??? */ 0, 0, 0, 0, 0, 0,
/*114 - ??? */ 0, 0, 0, 0, 0, 0,
/*115 - '\\"' */ A('\\"'), A('\\"'), A('_'), C('_'),
/*116 - ??? */ 0, 0, 0, 0, 0, 0,
/*117 - ??? */ 0, 0, 0, 0, 0, 0,

```

/43で、43行目に貼り付けて、元々の記述(A(''))、(A(''))、を削除して以下

```

/* 40 - ';' */ L(';'), 'S', A(';'), A(';'), A('S'), C('E'),
/* 41 - KANJI */ 0, 0, 0, 0, 0, 0,
/* 42 - 1. SHIFT */ SHIFT, SHIFT, SHIFT, SHIFT, SHIFT, SHIFT,
/* 43 - 'l' */ L('l'), 'L', A('l'), A('l'), A('L'), C('I'),
/* 44 - 'z' */ L('z'), 'Z', A('z'), A('z'), A('Z'), C('Z'),

```

のように修正します。

この状態で、ファイルを保存 ([:]のキーでコマンドモードに切り替えて、w (write,) q (quit) を押し、ファイルを保存・終了) します。

```
/* 52 - ' ' */          ' ',          AC( ' '), AC( ' '), AC( ' '), CC( '@' ),
/* 53 - ' /' */        ' /',          A( ' /' ), A( ' /' ), A( ' ?' ), C( ' @' ),
/* 54 - ' r, SHIFT */  ' r,          SHIFT,  SHIFT,  SHIFT,  SHIFT,  SHIFT,  SHIFT,
/* 55 - ' *' */        ' *',          A( ' *' ), A( ' *' ), A( ' *' ), C( ' @' ),
/* 56 - ' ALT */       ' ALT,         ALT,    ALT,    ALT,    ALT,    ALT,    ALT,
:wq_
```

この状態で、japanese.src は更新されていますが、このファイルを使用している keyboard.c がまだ古いために、ひとつ上のディレクトリにある keyboard.c に「タッチ」して、keyboard.c を新しくします。

cd ..

touch keyboard.c

ここで、

cd /usr/src/tools で、ディレクトリを移動し、

make でコンパイルを行います。

```
/* 49 - ' n' */        ' n',          LC( ' n' ), ' n',          AC( ' n' ), AC( ' n' ), AC( ' n' )
/* 50 - ' m' */        ' m',          L( ' m' ), ' M',          A( ' m' ), A( ' m' ), A( ' M' )
/* 51 - ' , ' */       ' ,',          ' ,',          ' <',          A( ' , ' ), A( ' , ' ), A( ' <' )
/* 52 - ' . ' */       ' .',          ' .',          ' >',          A( ' . ' ), A( ' . ' ), A( ' >' )
/* 53 - ' /' */        ' /',          ' /',          ' ?',          AC( ' /' ), A( ' /' ), A( ' ?' )
:wq
# cd ..
# touch keyboard.c
# cd /usr/src/tools
# make
cd ../kernel && exec make -
exec cc -c -I/usr/include keyboard.c
```

keyboard.c が touch によって強制的に更新されて、コンパイルが実行されます。

最後に cp image /minix/2.0.4 と入力するのは、キーボードの切り替えと同様です。

この操作がうまくいくと、

Alt + [で \ が、

Alt +] で _ が、

Shift + Alt + [で | が表示されます。

これで、ノートパソコンの日本語キーボードでも、MINIX にすべてのキー入力を行うことができます。

vi使用での「復帰方法」について

上記の、「デリート+UNDO」で、コピーバッファに文字列を取り込むというのは、高速に操作するための技のひとつですが、慣れるまでは「予想外」の動作をすることになります。

そうした場合には、「ESC」を何度か押して「挿入モード」から抜け出し、あるいは、コマンドを中断させて、

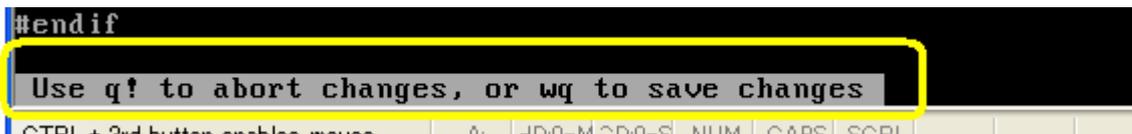
:q!

と入力します。

単に

:q

でも中断できますが、「ファイルが保存されていない」という理由で、以下のような



メッセージが表示されます。強制的に終了するには、q!と入力すればよく、こうすれば、誤ってファイルの上にごちゃごちゃと書き込んで壊してしまっても、復帰することができます。

もう一つの確実な方法は、viで編集操作を行う前に、

cp japanese.src japanese.src.org

と入力して、これから編集するファイルを、.org (original 元々のファイル) にコピーしておくことです。

これならば、万一誤編集操作でjapanese.srcというファイルを壊してしまっても

cp japanese.src.org japanese.src

として、壊したファイルに、元々のファイルを上書きして元に戻すことが可能です。

さらに大技のバックアップとしては、Windowsの上でminix.imgを別ファイルにコピーしておくことです。