viを使っての編集

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。

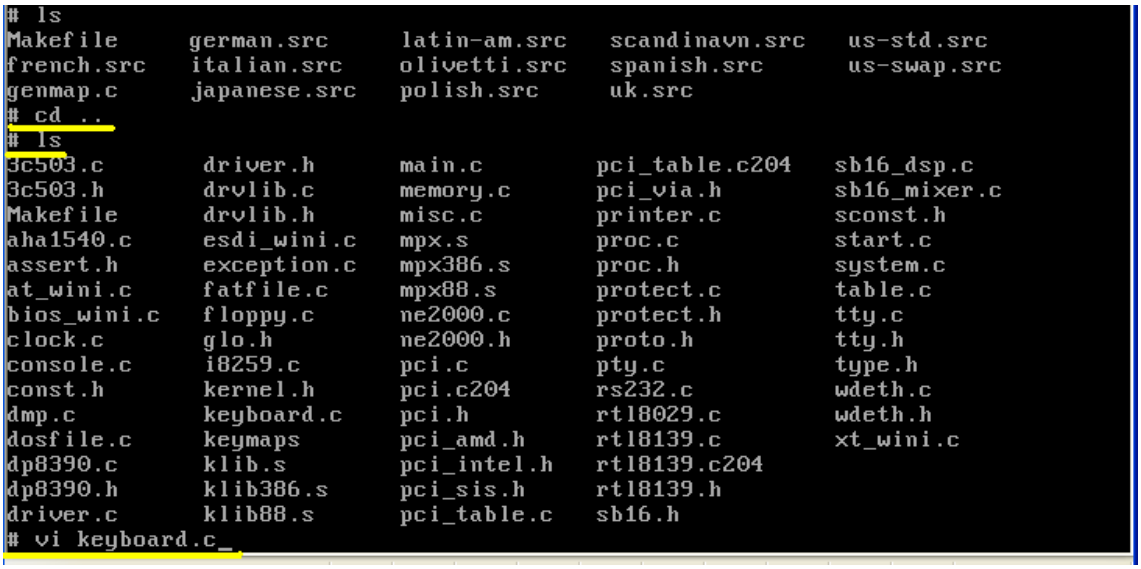
まず、root でログインします。root と入力してから[Enter]を押してください。
以下、コマンド入力する時には、最後に必ず[Enter]を押します。（いちいち書きません。）
次に、cd /usr/src/kernel/keymaps
で、ソースコードの、keymaps のディレクトリに移動します。

ここで、ls と入力して

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。

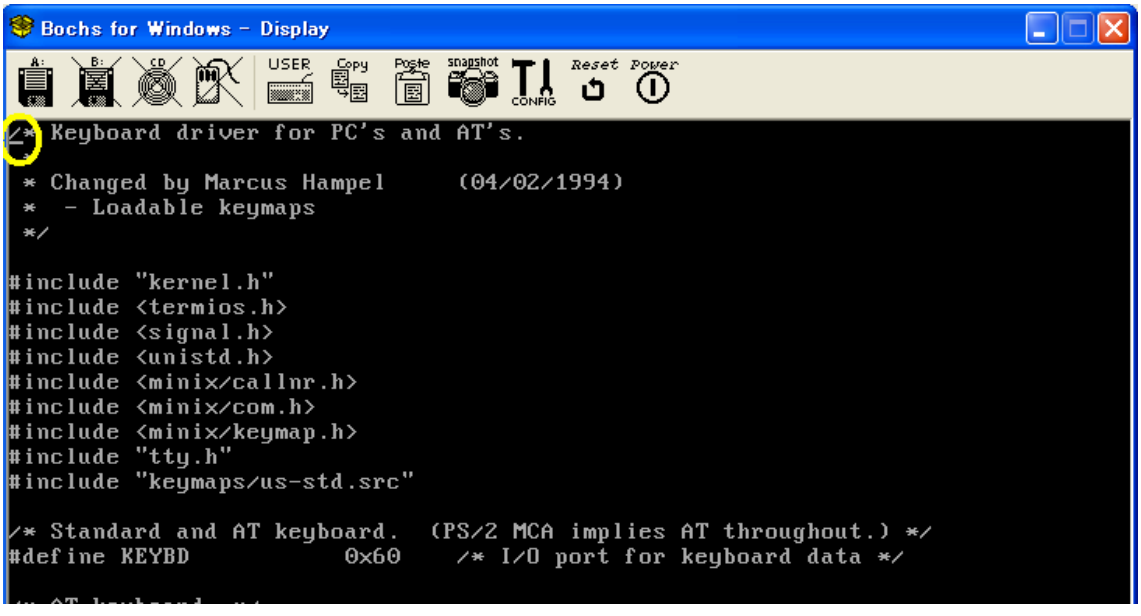


ここで、keyboard.c というソースファイルを編集します。

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になります。

vi

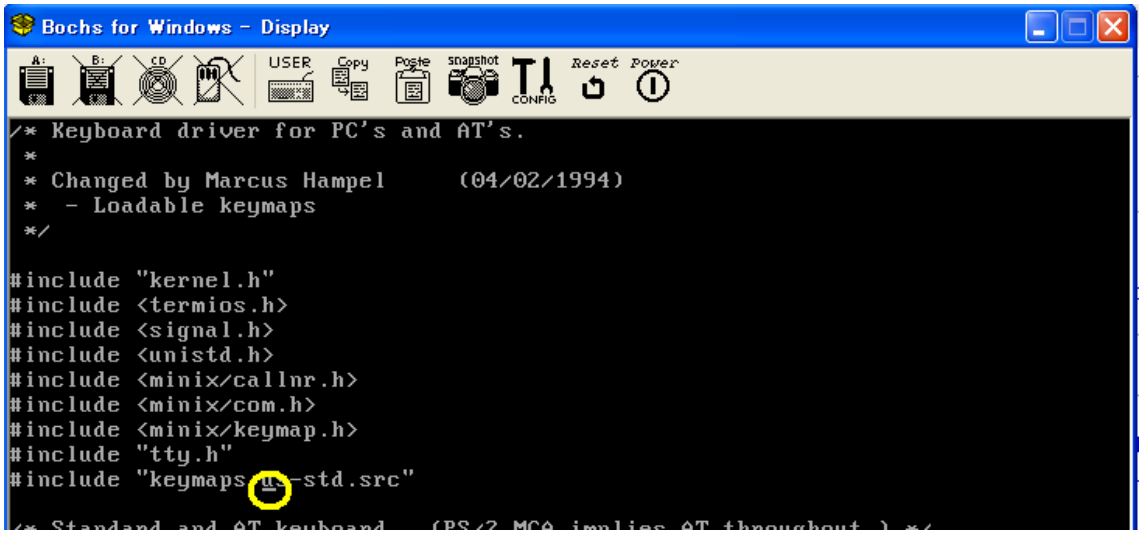


わかりにくいですが、左上にカーソル（__）が現れています。

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。

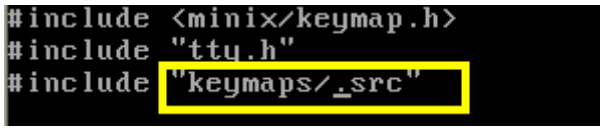
次に、ｌのキーを１８回押して右に移動します。
または、右向き矢印キーを１８回押します。
すでに気づいている学生がいると思いますが、18l と入力しても右に１８文字移動します。
さらに簡単な方法があります。ｗのキーを押すと「ワード」単位で右に移動しますから、ｗを５回押しても同じ位置に移動します。5w と入力しても同じです。

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。



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 の部分が削除されます。
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 の部分が消えます。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。



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。

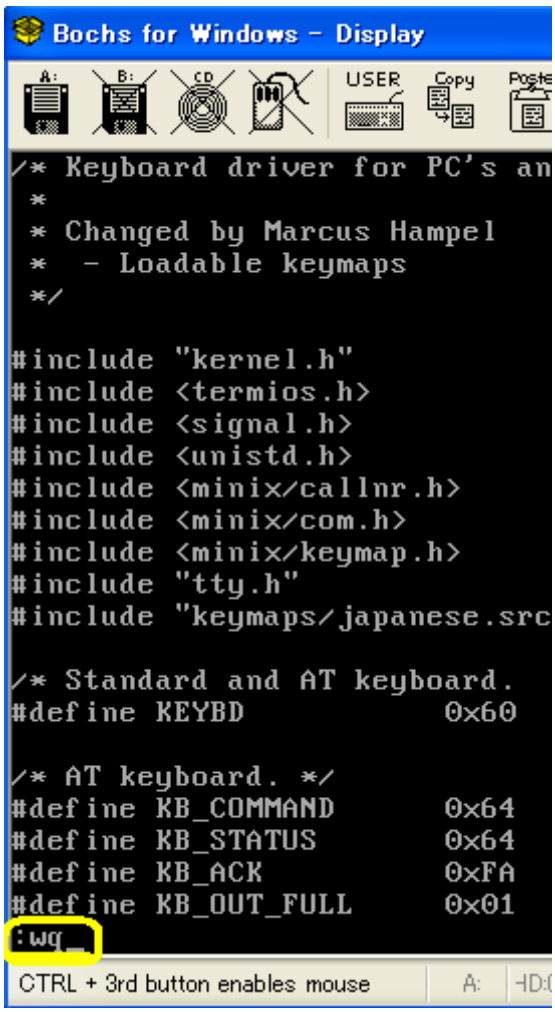
一旦、ｉのキーを押して、モードを切り替えてから japanese と入力し、ESC で「挿入モード」から抜け出します。（[ESC]キーは、画面左上にあります。）
ここでは、入力した内容を画面でも見ることができます。

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。

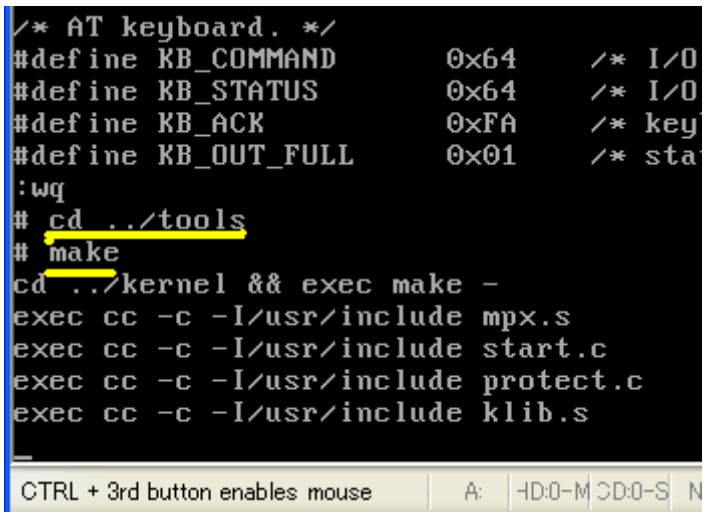


画面では、上のようになります。「：」を受け付けると、「コマンド入力行」は画面左下に表示されます。そこで、画面で確認しながらｗｑなどを入力します。

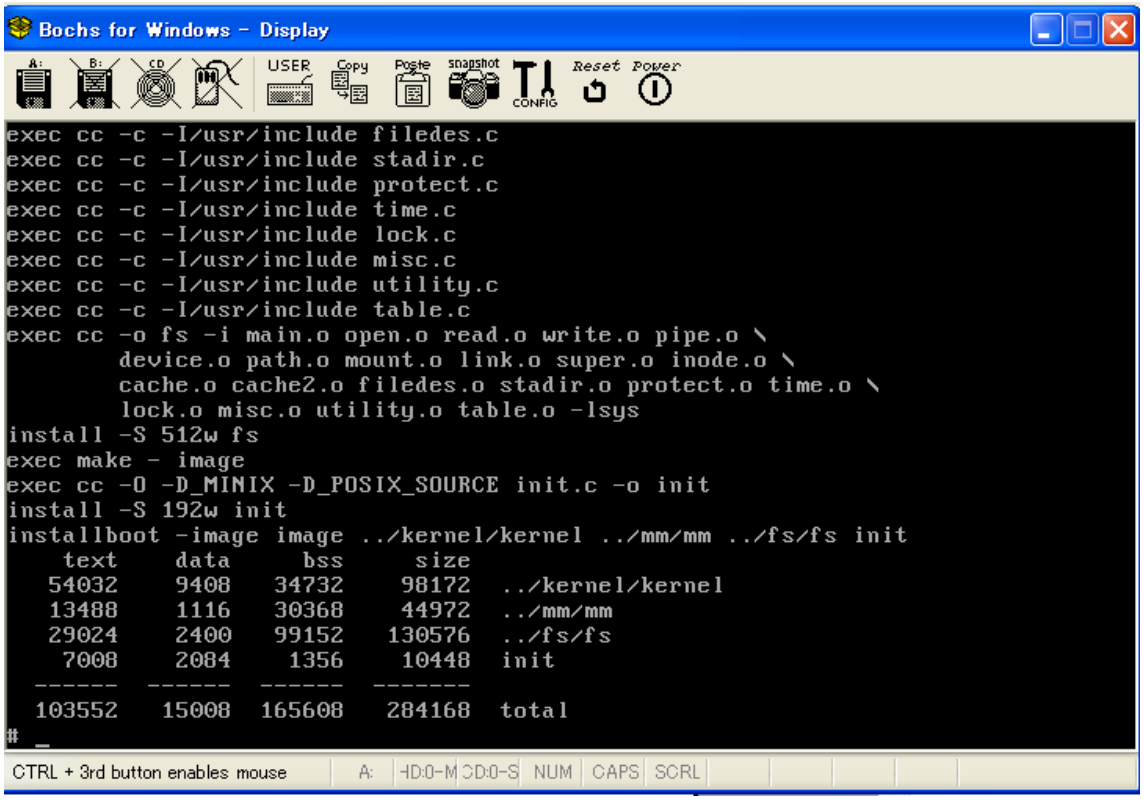
無事終了すると、「#」（MINIX の root モードでのプロンプト）が表示されます。

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。



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。）

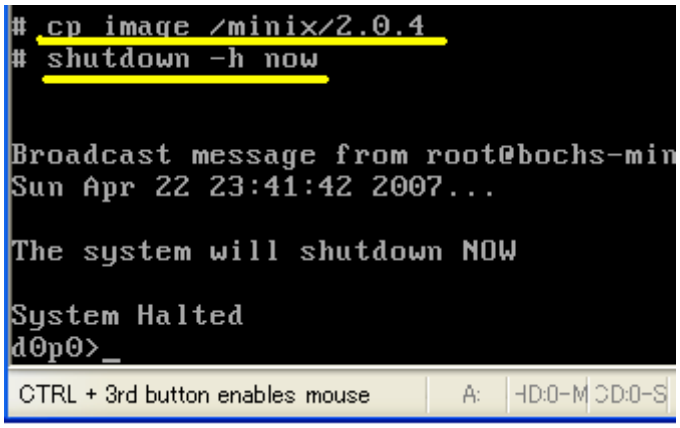


コンパイルが完了すると、上のような画面になっているはずです。

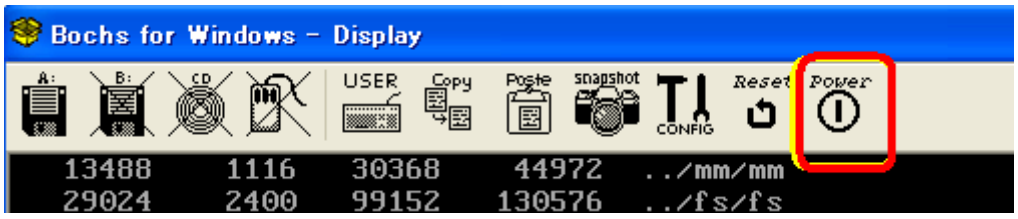
ここで、cp image /minix/2.0.4 と入力します。
コマンドの意味は、今コンパイルしたばかりの「OS のバイナリーイメージ（image）のファイルを、/minix というディレクトリの下の 2.0.4 というファイルにコピーする、」です。
ここで、2.0.4 というファイルはすでに存在しています。が、MINIX の OS では、「上書き」についての警告は一切ありません。

一旦、システムを抜け出します。

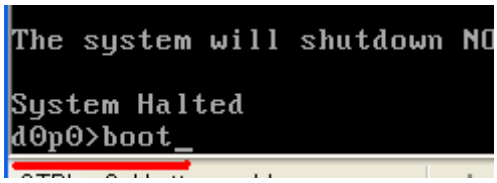
shutdown -h now
と入力します。



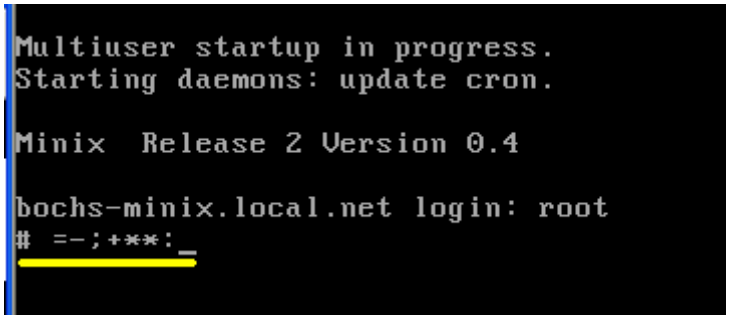
（この状態で、POWER のボタンを押して、仮想コンピュータの電源を切ると、次にブートした際に、「ふだんと違うメッセージ」は表示されません。）



ここで、boot と入力します。



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。



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」

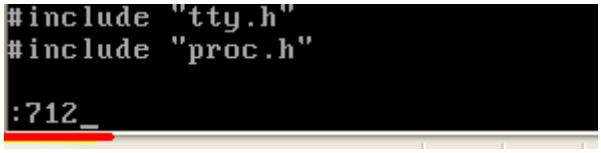
ｖｉの編集などで、パソコンによっては大音量でビープ音が鳴ったことと思います。カーソルが移動できない状態で左向きのキーを押したり、コマンドエラーなどがある度に、ビープ音が鳴動して、操作が一時停止していたと思います。

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。

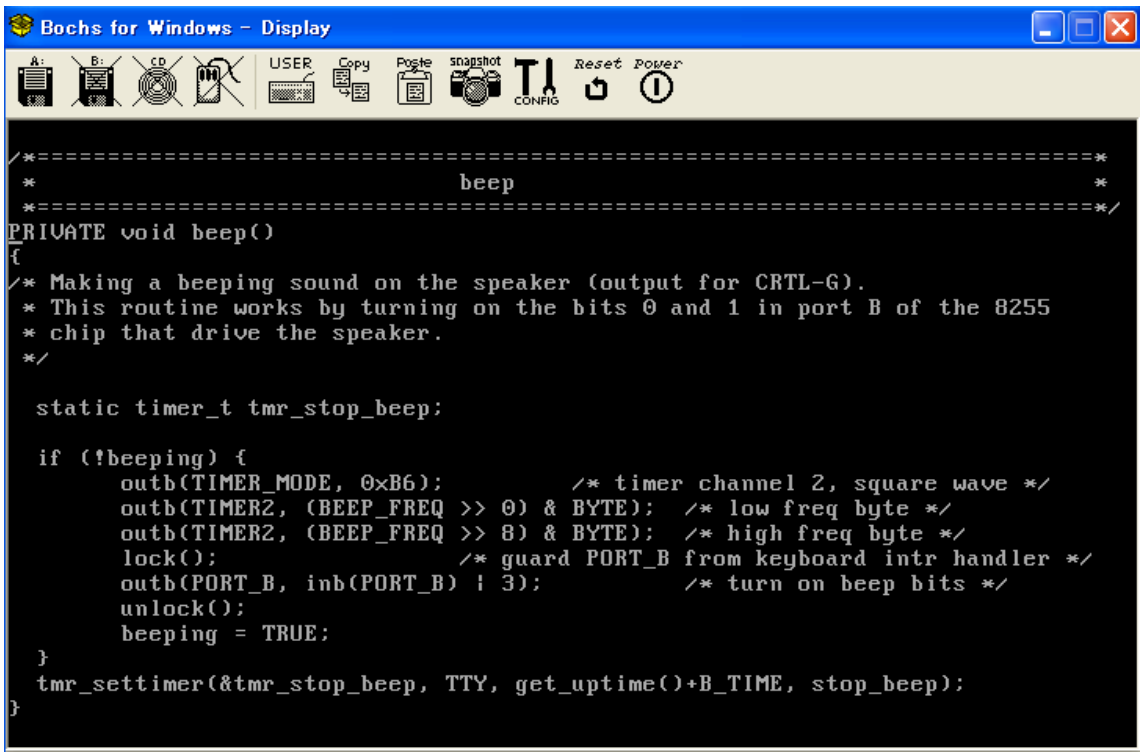
`:712`
と入力して、７１２行目にジャンプする。



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いましょう。

```
 *===========================================
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。

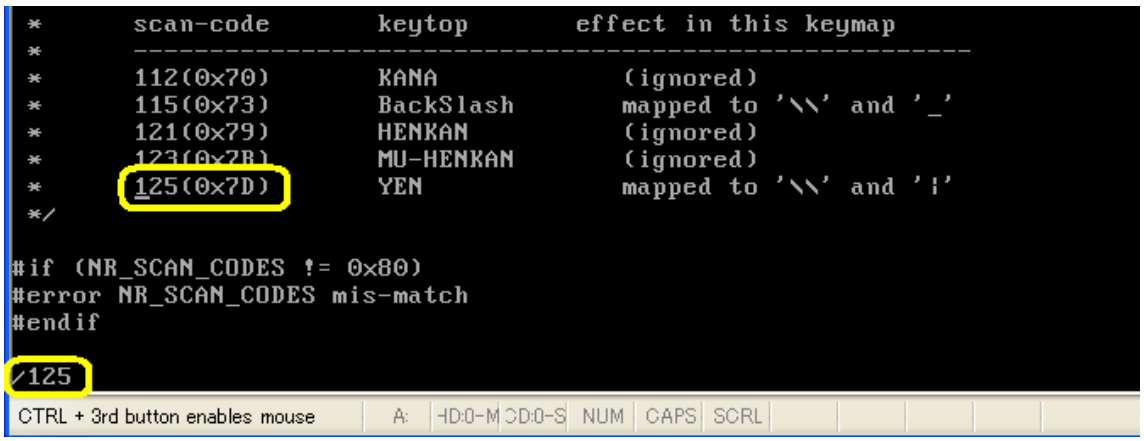
__（アンダースコア）と‘｜’（縦棒）のキー割付

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。

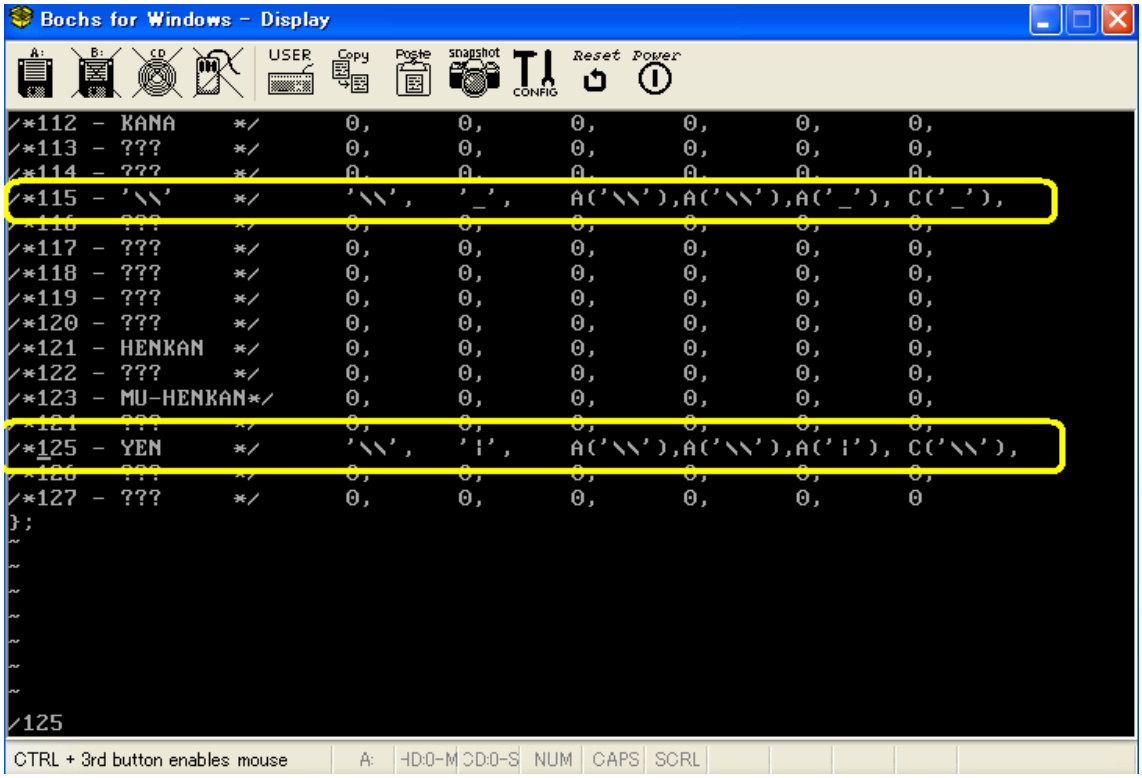
cd /usr/src/kernel/keymaps でディレクトリを移動し、
vi japanese.src で、日本語キー割り当てのファイルを開く。

/125 と入力して、ファイル中の 125 という文字列を検索する。



最初はまず、プログラム先頭部のコメントの１２５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。



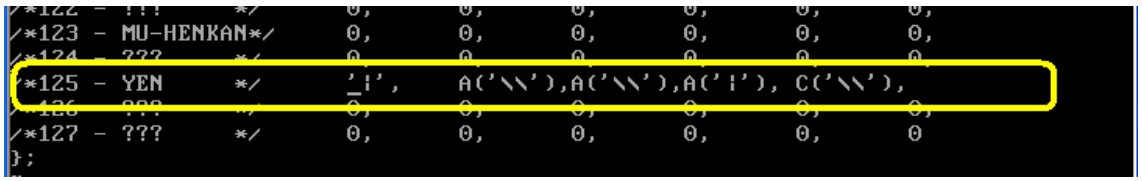
[w]（ワードジャンプ）を４回押して、‘＼＼’,の部分まで、カーソルを移動させる。

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。

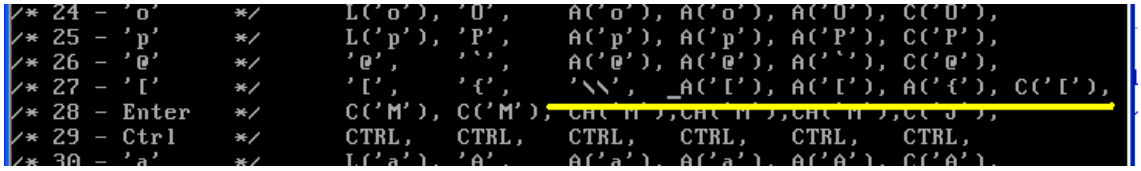
削除してから元に戻す、とは、何のためにこんな無駄な操作を行ったかと言えば、「削除」した際に「削除」した部分が「コピーバッファー」に取り込まれるため、次から「貼り付け」が可能になるためです。

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。

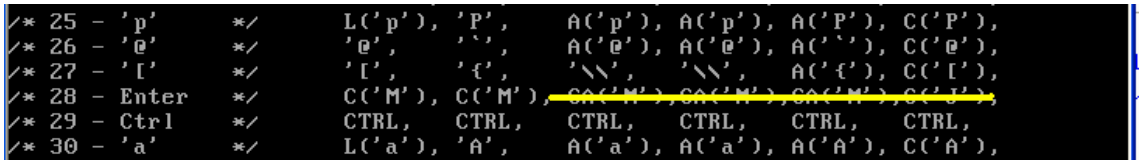


次に、／２７と入力して２７の文字列を検索させ、([n]（次）も使いながら、）２７のコードにジャンプします。

ここの、A('['], 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ーに取り込まれた‘＼＼’の文字列がここに挿入されます。

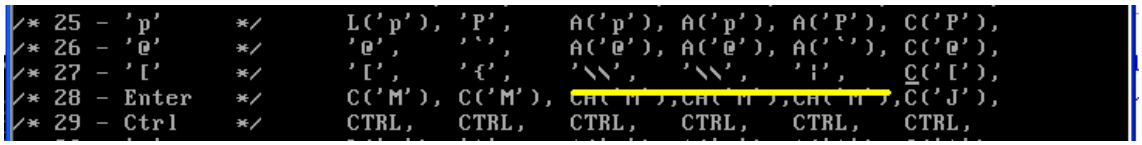


この状態の画面の一部です。左右両側の[Alt]キーに対応させるために、もう一度この[p]のキーを押す操作を行い、次に、A['['], A['['],を[x]のキーで削除して、以下のような表示にします。

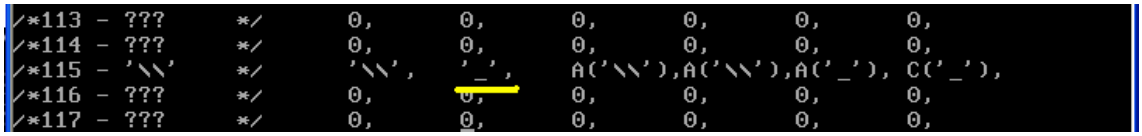


再び、/125 でファイル下部にジャンプして、今度は、'ｉ', の部分で[dw][u]を実行して、コピーバッファーに取り込み、/27 で画面上部に戻ってから、今度は A('{'},の部分を置き換えます。

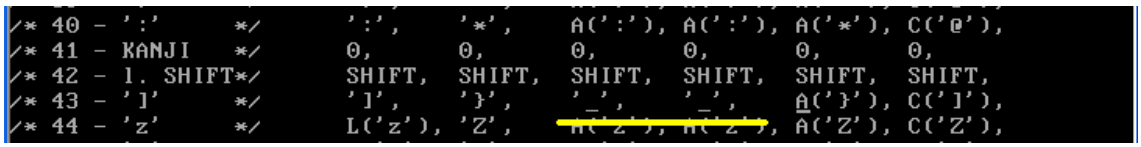
［ｐ］を押して貼り付けを行ってから、A('{'},を削除します。２７の行は、最終的に以下のようになります。



同様の操作を繰り返して/115 の行から、‘__’をコピーして、

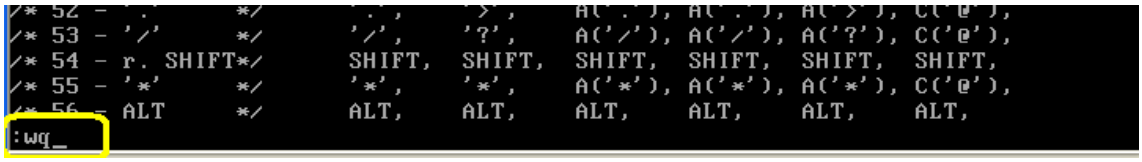


/43 で、４３行目に貼り付けて、元々の記述（A(')')，（A(')')，を削除して以下



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。



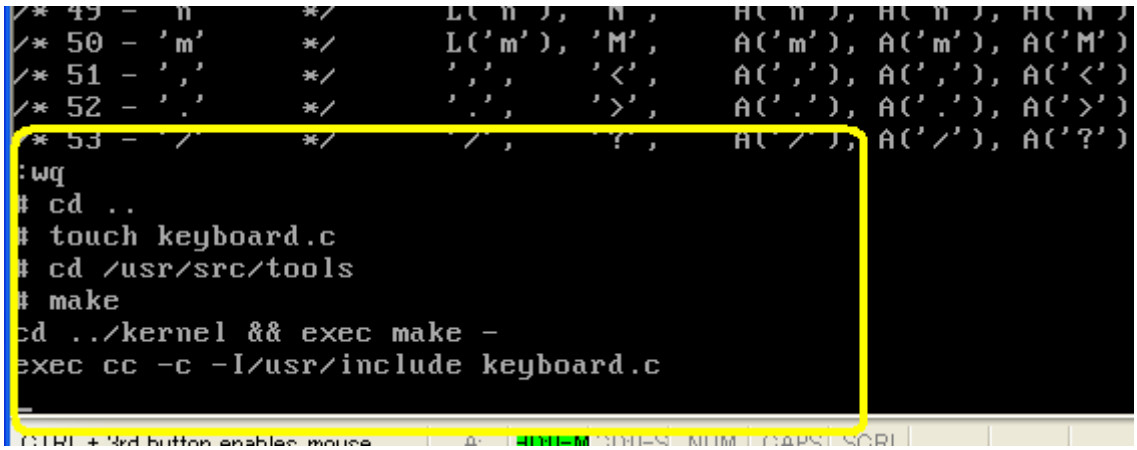
この状態で、japanese.src は更新されていますが、このファイルを使用している keyboard.c がまだ古いために、ひとつ上のディレクトリにある keyboard.c に「タッチ」して、keyboard.c を新しくします。

```
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。



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、コピーバッファー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。