Kondara MNU/Linux における UTF-8 対応の現状

Kondara Project
デジタルファクトリ株式会社 *
市村元信

2001 年 9 月 27 日

概 要

Kondara Project において、今後、Kondara MNU/Linux は UTF-8 に移行するという旨のアナウンスとしたのは 2000 年 5 月 10 日[1] の事であった。それから一年以上が過ぎようとしているが、現状、UTF-8 への対応はどの程度進んでいるのだろうか。本稿では Kondara MNU/Linux における UTF-8 対応の現状と今後について紹介する。

1 はじめに

Linux が普及しはじめ、世界に広く認知されるようになった。その結果、Linux に対する国際化対応を望む声が高まり、1999 年 9 月には LI18NUX (Linux Internatinalisation Initiative)[2] が設立された。また、国際化の流れがすすむ中で、今までのような直接的地域化から I18N のフレームワークに沿った形での地域化を行う必要性が増してきたといえる。そのような流れの中で、アメリカ、ヨーロッパ等では、Unicode、特に UTF-8 エンコーディングを用いて国際化を行う考えが主流になりつつあり、今後、UTF-8 のサポートは必須になっていく事が考えられる。このような状況を踏まえた上で、Kondara MNU/Linux では既存の Posix Locale の枠組の中で、UTF-8 ロカールをサポートする事にし、今後 UTF-8 ロカールを Kondara MNU/Linux の標準のロカールに移行するための作業を開始した。

本論文では、Kondara MNU/Linux の UTF-8 対応の過程と現在の状況、また今後の開発予定について述べる。

2 glibc

2.1 glibc

UTF-8 対応を開始する上で基盤となるものは libc である。Kondara MNU/Linux では、まず glibc を UTF-8 に対応させた。

2.2 locale-utf

UTF-8 ロカールを利用するためには UTF-8 用のロカールデータが必要となる。Kondara MNU/Linux では、UTF-8 のロカールデータは locale-utf パッケージで提供しており、これは glibc の提供する localedef によって作成される。locale-utf では、以下のような振る舞いを行うようになっている。

- **LC_CTYPE** と **LC_COLLATE** は全ロカールで一つのデータを共有している。
- **LC_CTYPE** には、**ideograph**(表意文字) という非標準の charclass を追加した。(現在の locale-utf では **ideograph** は提供されない)
- WIDTH は、Unicode Standard Annex #11[3] を元にして設定している。これを利用する事

*ソリューション開発部

[1] http://www.kondara.org/announcement/

[2] http://www.li18nux.org/

[3] http://www.unicode.org/unicode/report/tr11/

で、UTF-8 ロカールにおいても **wcwidth**、**wcswidth** といった関数が正常に動作する。なお AUX #11 で「Ambiguous」に分類されている文字については 1 カラムと判定する。(これは glibc 本体が WIDTH を提供するようになったため現在では用意されていない)

- **LC_COLLATE** は、Unicode Technical Standard #10[4] から持ってきたデータを利用しており、基本的には Normalization Form C な文字列に対しては、UTS #10 と同等の Collation が行われる。但し、UTS #10 で記述されている rearrangement は不可能であると思われるため、その部分はきちんと動作するかは保証できない。また、Hangle 文字列を Jamo まで分解した文字列と syllable と比較するためのデータは入っていない。フランス語は level 2 を Backward にしなければならないが、現状は Forward のままとなっている。

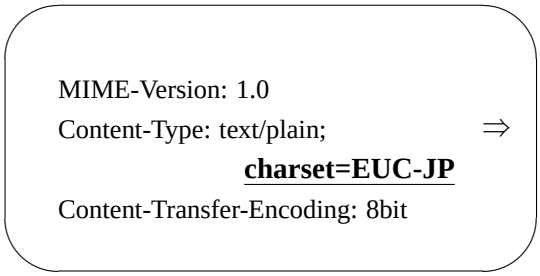

図 1: po ファイルへの charset の記述 (EUC-JP)

fgetwc	fgetws	fputwc
fputwc	fwide	fwprintf
fwscanf	getwc	getwchar
putwc	putwchar	swprintf
swscanf	ungetwc	vfwprintf
vswprintf	vwprintf	wcsftime
wprintf	wscanf	ngettext
dngettext	dcngettext	bind_textdomain_codeset

表 1: glibc 2.2 で新たに実装された関数

2.3 gettext

glibc 2.2 には gettext が提供する機能も取り込まれている。glibc 2.2 で提供されている gettext では、用意される po ファイルの Content-Type に正しく charset が設定されていれば、今現在利用している charset が po ファイルの記述のために利用されている charset と異なった場合にも、自動的にコード変換を行う機能が実装されている。この機能を正しく利用するためには、charset の指定には iconv の引き数として利用する事ができるものでなければならない。(図 1) この機能を用いる事で、po ファイルは各エンコーディング毎に用意する必要がなくなった。ただし、現実には、この機能が実装されたのは最近である事もあって、きちんと調整された po ファイルを提供していないアプリケーションも数多く存在しており、po ファイルの調整にはかなりの時間を費した。

[4]http://www.uni code.org/unicode/reports/tr10/

2.4 今後

現在では、glibc 2.2 の登場により、Linux においても国際化を意識したプログラム開発をするための下地が整ったといえる。glibc 2.1.3 の際には実装されていなかった表 1 のような関数が実装され、**LI18NUX 2000 Globalization Specification Version 1.0 with Amendment 2** [5] で定義されている関数も出揃った。また、最新の glibc 2.2.3 においては、UTF-8 用のロカールデータも glibc 本体に含まれるようになっている。以上の事をふまえると、

- カラム数の計算にはバイト数ではなく、glibc で提供される wcswidth,wcwidh 等の関数を利用する

- gettextize する場合には、適切な charset を設定する

[5]http://www.li18nux.org/li18nux2k/

といった点に注意してコーディングをする事によって、UTF-8 ロカールにも対応するアプリケーションを作成する事ができるようになった。一方で、既存のアプリケーションにはコラム数をバイト数とみなして記述されたものも数多く存在する。今後はそれらのアプリケーションの洗い出しと修正を進めていきたい。

3 X Window System

通常の Linux ユーザはほとんどの人が X Window System を利用している。そのため、UTF-8 対応を進めていくには、X Window System への対応も不可欠である。

3.1 XFree86

Kondara MNU/Linux において現在提供されている X Window System は XFree86 4.0.3 であるが Xlib に関しては、4.0.1 をベースにカスタマイズしたものを利用している。したがって Kondara MNU/Linux では Xutf8 関数は実装されていない。

Kondara MNU/Linux における XFree86 への UTF-8 サポートは、gtk+の次期バージョンから利用されるであろう Pango や Xlib-I18N のアプローチとは異なる。Pango や Xlib-I18N でのアプローチは既存のフォントセットを利用して UTF-8 のコードセットを描画するのに対し、Kondara MNU/Linux では、巨大な iso10646 フォントを利用する事で描画を行う。これらのフォントには電子書体オープンラボ[6]の成果も利用しており、既存のロカールとも共存できるよう、主要なフォントは**-alias-variable** で始まる別名を割りあてている。これによってロカールを切り替える度に設定ファイルを調整する必要がなくなった。

このようなアプローチを採用した理由は、表 2 にあるような双方のメリット、デメリットを実際に実装して比較した結果、単一の巨大なフォントをあらかじめ用意しておく方がパフォーマンスの点で効率がよいと判断したからである。XFree86 4.0 から導入された Bigfont 拡張を利用する事で、フォントを利用すると消費される約 800KB のメモリの消費を抑える事も可能になった。

	メリット	デメリット
Xlib-I18n	・非 UTF ロカールの時と同じフォントを利用できるため、ディスク容量が節約できる	・フォントの読み込みが遅い
	・CJK の間にロカールに従った優先順位を付ける事で、CJK で字形が微妙に違う文字に対応できる	・表示が遅い (Unicode マッピングを既存の legacy なマッピングに変換する必要がある)
		・CJK が混ざると文字のバランスが悪い
Kondara	・フォントの読み込みが速い	・CJK での字形の違いに対応できない
	・Unicode の全ての文字が表示できる。特にハングル文字	・ディスク容量が必要
	・CJK 間の文字のバランスを微調整できる	・メモリを消費する

表 2: Xlib-X18N での実装と Kondara MNU/Linux での実装

実際の開発時には、UTF-8 から Compound Text に変換する際、どこにマッピングするか、という問題が生じた。これはロカールで優先度を付ける事で回避している。

[6] http://openlab.ring.gr.jp/efont/

3.2 X Input Method

XFree86本体をUTF-8に対応させる事によって、UTF-8 ロカールで X Window System を起動させた場合、フォントが調整されていれば、Unicodeで定義されている文字に関しは表示する事が可能になる。つまり en_US.UTF-8 ロカールの場合にも日本語を表示する事が可能になった。次に必要な事は UTF-8 ロカールでの入力である。

Kondara MNU/Linux で利用できる XIM (X Input Method) としては、

- xcin (Chinese)
- ami (Korean)
- skkinput (Japanese)
- kinput2 (Japanese)

等があげられるが、これらの XIM は Compound Textで通信を行うため、ロカール判別部分に UTF-8 用の調整をいれるだけで、UTF-8 ロカールで動作するクライアントと通信する事が可能になった。

3.3 XIM Switch

UTF-8 ロカールで XIM が動作するようになったが、現状のままでは XIM は 1 つしか起動する事ができない。したがって、日本語、韓国語、中国語を切り替えたい、といった要求に答える事ができない。この問題を解決するために、Kondara MNU/Linux では上記の XIM を動的に切り換えるためのツールとして ximswitch(図 2) を開発した。ximswitch は、

- XRegisterIMInstantiateCallback

等を利用する事で、XIM を (X を再起動する事なく) 動的に切り換える事を可能にする。今後 IIIMF を利用する事が主流になっていく可能性が高いが、レイアウトエンジンがそろってくるまではまだまだ XIM を利用する機会が多い。ximswitch のように XIM を動的に切り換える機構は非常に有効である。

図 2: ximswitch

3.4 IIIMF

Kondara MNU/Linux には

- IIIMECF
- iiimf
- iiimf-conv
- iiimf-newpy
- iiimf-xiiimp

といった IIIMF を利用するためのツールが取り込まれている。しかし、現状では Xlib-I18N の成果を完全には取り込んでいないため、IIIMF は

- emacs や xemacs から IIIMECF を経由して利用する
- httx を経由して利用する

という 2 つの選択肢が用意されているだけであり、X Window System と Native に通信する事はできない。

3.5 Terminal Emulator

UTF-8 を扱うためには UTF-8 を扱う事のできる Terminal Emulator が必須である。現在、UTF-8 に対応した Terminal Emulator としては XTerm と LI18NUX で作成されている CSI XTerm がある。Kondara MNU/Linux では XTerm のバージョン 148 に Robert Brandy 氏の作成しているパッチ[7] を適用して幅広文字等に対応させており、これに加えて、

- XIM の起動
- XRegisterIMInstantiateCallback と XNDestroyCallback プロパティを利用した動的な XIM 切り替え

を実現するための機構を用意している。また、この XTerm は XA_UTF8_STRING のみに対応していたが、これを XA_COMPOUND_TEXT にも対応させている。この XTerm は現在の所 UTF-8 にしか対応していないため、ユーザはロカール毎に利用する端末を切り替える必要がある。

3.6 Window Manager

Kondara MNU/Linux では主な Window Manager として

- GNOME (sawfish)
- KDE
- Enlightenment

が用意されている。

Window Manager を UTF-8 に対応させる際に最も気をつけなければならない事は、各 Window Manager のメニューファイルの違いである。最近の Window Manager は他の Window Manger 向けに用意されるメニュー等もパースする設計になっている事が多いが、これらのメニューが各 Window Manager によって異なる可能性がある。また、UTF-8 に対応する事によって従来は 1 つのロカールを利用していた言語が UTF-8 ロカールも利用できる事になるため、従来のように一つのコードセットの事を考慮するだけでよかった状況ではなくなりつつある。現に、KDE では提供される desktop ファイルはすべて UTF-8 エンコーディングで記述されているが、GNOME で提供される desktop ファイルは各言語毎に標準で利用されるエンコーディングで記述されている。これらの差異を吸収するためには、Window Manager が適宜コード変換を行う必要がある。このため Kondara MNU/Linux では

- UTF-8 から現在のロカールで利用するエンコーディングに対する iconv を open
- 現在の言語で利用されるデフォルトのエンコーディング (ja なら eucJP) から現在のロカールで利用するエンコーディングに対する iconv を open
- コード変換要求があった場合、まず UTF-8 用に open した iconv を試し
- 失敗したら次の iconv を試す

というアプローチを採用した。しかし、このアプローチを行うためには、現在の言語で標準で利用されるエンコーディングを判別する必要がある。当初、Kondara MNU/Linux では、この判別のために特別な環境変数として **GETTEXT_ICONV** を利用した。この環境変数は

- EUCJP:UTF-8

といった形になっており、標準で利用されているエンコーディングと現在利用しているエンコーディングを対にしたものである。これを利用して、まずはパースしようとしているメニューが UTF-8 エンコーディングであるかどうかをチェックし、違うならば **GETTEXT_ICONV** で定義されているエンコーディングと見なす、という手法である。ただし、この実装の場合、ただしく動作するためには **GETTEXT_ICONV** 環境変数が正しく設定されている必要がある。

[7] http://www.suse.org.uk/f̃obert/xterm/

現在はこの環境変数は利用せず、localeが**言語_地域**と設定されている場合に利用されるエンコーディングを標準で利用されているエンコーディングとして設定するような実装となっている。

現在の言語で利用されるデフォルトのエンコーディングを取得するためのコードの一部を以下に記す。

```
static const char *
get_legacy_codeset (void)
{
   char *current_locale, *alt_locale, *p;
   const char *r;

   r = getenv ("E_LEGACY_CODESET");
   if (r) return strdup (r);

   current_locale = strdup (setlocale (LC_CTYPE,NULL));
   alt_locale = strdup (current_locale);
   // TODO: what about locale modifiers?
   p = strchr (alt_locale,'.');
   if (p) {
      *p = '\0';
      if (setlocale (LC_CTYPE, alt_locale) != NULL) {
         r = strdup (nl_langinfo (_NL_CTYPE_CODESET_NAME));
      }
      setlocale (LC_CTYPE, current_locale);
   }
   free (current_locale);
   free (alt_locale);
   if (r) return r;
   return strdup (nl_langinfo (_NL_CTYPE_CODESET_NAME));
}
```

このカスタマイズにより、UTF-8 対応とともに、Window Manager 毎のメニューファイルのエンコードの違いに対応させた。

3.7 sdr,gdm,wdm,gnomecc

Kondara MNU/Linux では、国際化を進めていった結果、多数の Window Manager、Input Me thod、ロカールがサポートされるようになった。特に、UTF-8 ロカールへ対応した結果、Legacy Codeset を含め、既に 190 以上ものロカールが選択できる状況にある。このように多種多様なセッティングができるようになると、容易にこれらを設定できる機構が必要である。この機構を実装したのが sdr (Simple Display Reconfigure) である。(図 3)。現在では、sdr の他に

- wdm (WINGs Display Manager)
- gdm (The GNOME Display Manager)
- gnomecc (The Gnome Control Center)

もこの機構に対応させた。

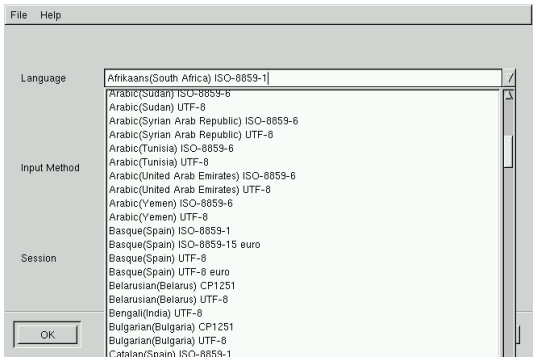

図 3: sdr

3.8 今後

現在では、X Window System における UTF-8 対応は実用的なレベルにまで達している。しかし、現状では

- Kondara MNU/Linux における実装
- li18nux における実装
- XFree86 本家における実装

という 3 つの異なるアプローチの実装がある状況であり、あまり好ましい状況にあるとはいえない。その中でも特に意見の相違が顕著である部分は、UTF-8 そのものの捉え方の違いにある。Kondara MNU/Linux における実装と li18nux における実装は、その点では意見は一致しており、UTF-8 を数あるロカールの内の 1 つと捉えている。一方で、XFree86 本家のアプローチとしては UTF-8 を万能薬として扱う傾向になりつつあり、これは、Xutf8 関数群にみてとれる (前述した通り、Kondara MNU/Linux の実装には Xutf8 関数群は含まれていない)。今の所 Kondara MNU/Linux で提供される XFree86 パッケージは、オプションを変更してリビルドする事によって Xlib-I18N 相当になるように調整されているが、今後は徐々に統合していく形を念頭に起きながら調整をしていく必要がある。

4 Library

4.1 libjconv

libjconv は当初は、日本語のエンコーディングを変換するためのライブラリとして開発した。しかし、glibc に対する UTF-8 サポートの追加や、iconv 関数が充実してきた事もあって、現在ではより発展して、UTF-8 を含む様々なエンコーディング変換を行えるように拡張された。libjconv は名前からも想像できるように、当初から iconv へのラッパーライブラリであり、文字コード変換機能をより容易に組み込む事を可能にするものである。libjconv の設定ファイルは図 4.1 に示すように

- **ロカール: 利用されるエンコーディング候補**

という書式になっており、ja_JP.UTF-8 というロカールを利用しているとすると、

```
ja_JP.UTF-8: UTF-8 ISO-2022-JP ⇒
ISO-2022-JP-2 EUC-JP SJIS UTF-8
```

図 4: libjconv の設定ファイル

という設定になっている。

libjconv のアプローチは基本的に try & error である。設定ファイルに記述されているエンコーディング候補に従い、順番に iconv を試し、失敗したら次のエンコーディングでの iconv を試みる。もちろん、プログラマの意図に応じて、異なるアプローチを利用する事も可能である。

libjconv は特にネットワークアプリケーションのように、様々なコードセットを判別する必要があるアプリケーションで特に威力を発揮する。Kondara MNU/Linux では、メーラである Sylpheed や IRC クライアントの XChat、XMMS など文字コード変換を伴う多数のアプリケーションで libjconv を利用するように調整している。

4.2 gtk+

gtk+は現在では GUI アプリケーションを作成するために良く利用されるライブラリの一つである。Kondara MNU/Linux では、GtkText での Word Wrap 機能等を UTF-8 エンコーディングに対応させている。また、UTF-8 エンコーディング用の gtkrc ファイルもあらかじめ用意している。

4.3 readline

前述したように、UTF-8 でエンコーディングされた文字列は、既存のマルチバイト対応パッチでは対応できなくなる場合が生じる。
通常、マルチバイトに対応させるためのパッチは

- マルチバイト文字であるかどうか判別するコード
- マルチバイト文字の処理を施すコード

といった構成を持つ事が多い。この際、今までは 1 バイトで表現される文字は 1 カラム、2 バイトで表示される文字は 2 カラム分の幅を利用する事が多かったため、バイト数 == カラム数として処理を行う事が多かったといえる。しかし、UTF-8 エンコーディングを利用する場合、日本語であれば、通常表現に 3 バイトを要する。しかし、表示に際して必要な幅は 2 カラム分である。つまり UTF-8 でエンコードする場合

```
バイト数 == カラム数
```

という前提を利用する事はできなくなる。実際には UTF-8 だけではなく既存の ja_JP.euc JP を利用している場合でも、半角カナや jisx0212 等の文字はバイト数とカラム数が異なる。この問題を解決するためには、正しく動作するようになった **wcwidth** や **wcswidth** を利用して文字幅を計算する必要がある。readline ライブラリのような端末操作の基本となるライブラリでは特に注意する必要がある。Kondara MNU/Linux では、文字幅の計算には **wcwidth** を利用するようにカスタマイズした readline ライブラリを提供している。

4.4 今後

ライブラリはアプリケーション開発の基盤となるものであり、ライブラリを整備しておく事でユーザが苦労する事なしに UTF-8 対応にさせる事ができる。現在でも gtk+を UTF-8 対応にさせた事により、普通の gtk+アプリケーションは UTF-8 ロカールでも動作するようになっている。また libjconv を開発した事で文字コードの変換を伴うアプリケーションの UTF-8 対応も非常に容易になった。このように UTF-8 対応をより促進させるために今後もライブラリの整備により力をいれていきたい。

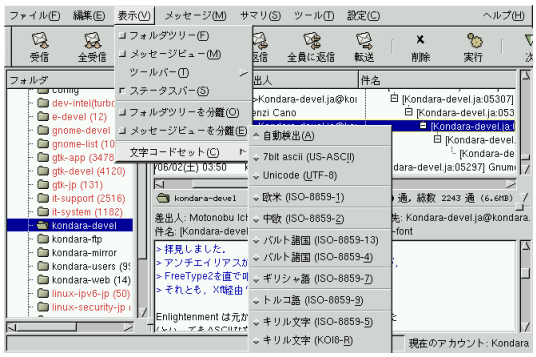

図 5: Sylpheed

5 Internet Tools

5.1 Sylpheed

Sylpheed[8] は山本博之氏によって作成されたメーラである。メーラはインターネットを利用する上で最も頻繁に利用するアプリケーションの一つであり、その中でも Sylpheed は最近急激に人気を増しているメーラである。Kondara Project では、Sylpheed に対して libjconv を適用するためのパッチを作成し、取り込んでもらう事ができた。このため、Sylpheed は libjconv がインストールされていれば UTF-8 をはじめ多くのエンコードに対応する事ができる。libjconv 対応時にはデフォルトの設定で、言語ごとに最も広く利用されているコードセットが使用される。また、メニューバーから送信コードセットを変更する事も可能である。(図 5)

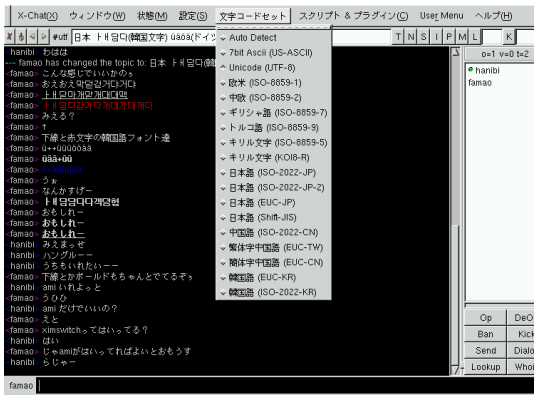

図 6: XChat

5.2 XChat

Kondara MNU/Linux に搭載されている XChat は libjconv を適用するパッチをあてている。このため、各種ロカールに応じて UTF-8 を含めた様々な文字を扱う事ができる。(図 6)

[8] sylpheed.good-day.net/

5.3 Mozilla

インターネットブラウザとしては Mozilla が UTF-8 に対応している。Kondara MNU/Linux では動的な XIM の切り換えにはやや難があるものの、Japanese,Korean,Chinese をテキストエリアに入力する事が可能となっている。

5.4 w3mmee

w3mmee (WWW wo Miru Tool Multiple character Encoding Extension) [9] は SUTO Kiyokazu 氏によってメンテナンスされており w3m [10] をベースに多国語への対応を行っている。w3mmee は多国語に対応したページャであり、テキストベースのブラウザとしても利用できる。

5.5 今後

現在では、Internet Tools として必要と思われるアプリケーションは UTF-8 対応のツールを見つけられるようになってきた。また RFC2277 等でもわかるようにこれからネットワークアプリケーションも UTF-8 に対応したものが増えていく事が予想できる。しかし、実際に UTF-8 を利用しているか、というとまだまだ利用している人は少ない。特に Internet Tools の場合、各地域において直接的地域化する、という傾向が強く、UTF-8 対応の実装が行われても、すぐに UTF-8 を利用しよう、という話にはなりそうもない。徐々に広めていくしか方法がないように思われる。今後は UTF-8 に対応するアプリケーションを徐々に増やしていく、といった方向で開発をすすめていく必要がある。また、実際に IRC において UTF-8 チャンネルを開いて体験した事だが、Posix Locale の枠組みの中においては文字コードの自動判別が不可能な場合も存在し、韓国の人が EUC-KR な文字コードを利用していたために、こちらから EUC-KR を送信できても、相手からのメッセージをロカールに依存した自動判別にしておくと EUC-JP として解釈されてしまう、といった事態も生じる。これらについても良いアプローチを探していく必要がある。

[9] http://pub.ks-and-ks.ne.jp/prog/w3mmee/

[10] http://ei5nazha.yz.yamagata-u.ac.jp/ aito/w3m/

6 Shell,Utilities

Shell や Utility は Linux を利用するためには欠かせないツールである。基本的に、OS との対話は Shell を通じて行われる。つまり、Shell や、よく利用するツールが UTF-8 を対応してなければ、それは UTF-8 に対応しているとはいい難い。

6.1 bash

Kondara MNU/Linux では bash の version 2 をカスタマイズしたものを採用しており、bash を利用する事で、UTF-8 を始めとした各種ロカールを利用する事ができる。この bash は、file-system-safe な文字列をファイル名や、引き数に利用する事ができ、グロブする事もある程度可能なように設計されている。

6.2 sed

Kondara MNU/Linux の sed では、正規表現のコンパイルには glibc の提供する regex が利用される。また y コマンドを利用した際には、マルチバイト文字は一旦ワイド文字へと変換され、UCS-2 の範囲で計算が行なわれるようになっている。

6.3 fileutils

Kondara MNU/Linux の fileutils に含まれる ls は wchar_t を利用するようにカスタマイズされている。そのため、UTF-8 エンコーディングで記述されたファイルを始め、file-system-safe なエンコーディングを扱う事が可能である。(図 7)

図 7: File-System-Safe な文字を扱う

6.4 lv

lv (a Powerful Multilingual File Viewer) [11] は Narita Tomio 氏によって作成されたページャであり、UTF-8 を始めとして多種のエンコーディングに対応している。XTerm を利用している時には、lv -Ou8 とする事で UTF-8 で記述されているテキストを閲覧する事が可能である。また、MANPAGER に'lv -Ou8' を設定しておく事で、ja_JP.UTF-8 ロカールを利用している時には XTerm で日本語の man を閲覧する事が可能になる。

6.5 今後

現在の所、Kondara MNU/Linux で提供している shell の中で、UTF-8 に対応しているシェルは bash のみである。shell は最も良く使うツールであり、tcsh など bash 以外の shell を利用している人が UTF-8 に対応している、というだけで移行するとは考えにくい。今後はより多くの shell を UTF-8 に対応させていかなくてはならない。また、コンソール系の Utility は、UTF-8 に対応するためにちょっとした修正がいるものが非常にたくさん存在している。通常に日本語を表示させるだけでもなんとなく列がズレてしまっていたり、といった事はよく目にする光景である。こういったツールを UTF-8 に対応させていく事も必要になってくる。

[11] http://www.ff.iij4u.or.jp/ nrt/

7 Editor

UTF-8 ロカールにおいてテキスト操作をするためには、UTF-8 ロカールにも対応したエディタが必要である。ここでは、UTF-8 に対応しているエディタを紹介する。

7.1 MGEdit

MGEdit は Kondara Project の 樋口証氏によって作成されたエディタで、iconv を駆使して多様なエンコーディングに対応している (図 8)。

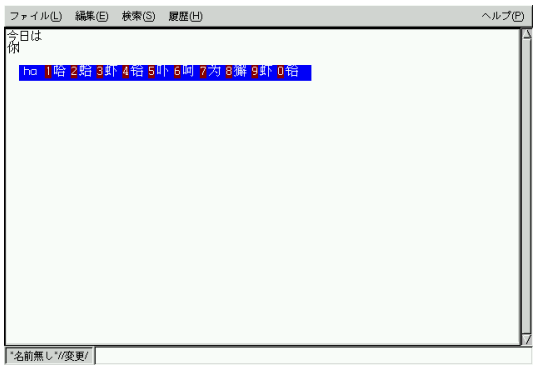

図 8: MGEdit

7.2 Vim

vim は現在 6.0 のアルファ版が UTF-8 に対応している。Kondara MNU/Linux では、この 6 .0 アルファ版を利用しており、UTF-8 ロカールで起動した場合には、fileencodings ディレクティブを利用して、コードの自動判別を行うように設定されている。Kondara MNU/Linux では、日本語を利用する場合には fileencodings ディレクティブは

iso-2022-jp,iso-2022-jp-2,euc-jp,sjis,utf-8

のように設定される。このアプローチは libjconv のものと同様である。(図 9)

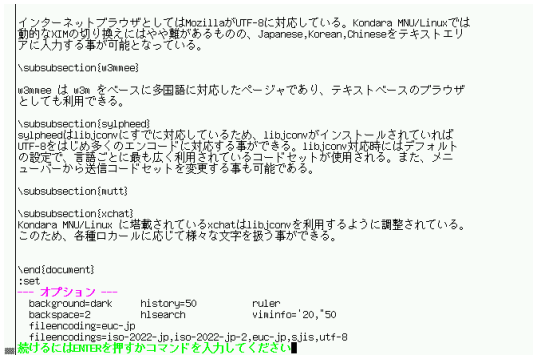

図 9: Vim

7.3 Others

上述の Editor の他にも gtk+や qt、Xlib を利用した Editor はコード変換はしてくれないものの、ほとんどのものが UTF-8 に対応している。また、emacs、xemacs も Mule-UCS パッケージを利用する事で、UTF-8 を利用する事ができる。

7.4 DataBase

7.4.1 PostgreSQL

PostgreSQL は UTF-8(ただし UCS-2 の範囲) 対応している。Kondara MNU/Linux においては configure 時に –enable-multibyte=UNICODE を利用している。そのため、標準でデータベースを作成した場合に利用されるエンコーディングは UTF-8 となる。 また 4.3 節に記述したようにカスタマイズされた readline を利用してコンパイルされる事によって SQL フロントエンドである psql も UTF-8 に対応している。pgaccess も UTF-8 の文字列を表示する事が可能である。

7.5 Others

今まで説明してきたアプリケーション以外にも様々なアプリケーションが UTF-8 に対応、もしくは対応を予定している。スクリプト言語等のテキスト処理を得意とするアプリケーション等は、ほぼ UTF-8 に対応する、といってよいだろう。本論文では筆者が調査していないために書く事ができなかったが、UTF-8 に対応したライブラリ、アプリケーションはまだまだたくさんある。また、これからも数多くのアプリケーションが UTF-8 に対応していくだろう。

8 課題

Kondara MNU/Linux は早期から国際化を目指して作成されているディストリビューションであり、UTF-8 への対応もはやくから行われてきた。また、Linux の普及と共に、IBM、Sun、LI18NUX を始めとして多くの企業、団体が Linux の国際化にたくさんの寄与をしてくれるようになった。これらの作業がほぼ同時に行われてきた事から、現在、Kondara MNU/Linux で独自に開発をすすめてきた部分も多い。今後、どのような形であれ、国際化への作業結果を積極的に取り込み、また逆に積極的に取り込んでもらうための作業が必要になってくる。

また、UTF-8 に対応するためにはまだまだしなくてはいけない事が山積みである事もまた事実である。ざっとあげても

- 印刷環境の整備
- フォントの整備
- コンパイラ等の対応
- Input Method の充実
- BiDi への対応
- 各種 daemon の対応

といった事がすぐに思い浮ぶ。これらの課題についても徐々に対応していく必要がある。

9 おわりに

UTF-8 へ移行するというアナウンスをした際には各方面から賛否両論があったがとにかくやって

みなければはじまらないという思いで開発をしてきた。

現在では Kondara MNU/Linux における UTF-8 対応は、ほぼ日常的に UTF-8 ロカールで生活していても特に不自由を感じないまでになってきている。テキスト編集、メールの送受信、IRC での会話、Web の閲覧といったものから、シェルの操作、正規表現、データベースといったものまで今までとほぼ同じ感覚で動作する事ができる。また、XIM を動的に切り替える事で日本語以外の言語の入力も可能である。

このように Kondara MNU/Linux における UTF-8 への対応はゆっくりとではあるが、着実に歩みを進めており、UTF-8 対応をするための下地はほぼ整ったと考えてよい。あとはより多くのアプリケーションを対応させていく、といった作業を地道にこなしていけば、近い将来の内に UTF-8 ロカールを Kondara MNU/Linux の標準のロカールとして採用する事ができるようになるだろう。

10 謝辞

無知な私にいろいろ知識を教えてくれ、Kondara MNU/Linux の UTF-8 対応を積極的に行ってくれた樋口証氏に感謝します。そして、我儘な私をまだ許してくれる Kondara Project の皆様、あきらめずに Kondara MNU/Linux を利用してくださっている人達に感謝します。そして、これから開発に参加しよう、UTF-8 ロカールを試してみよう、と思った方にも。