国際化対応文字処理関数ライブラリの開発*

4S－1
林周志†
慶應義塾大学大学院政策・メディア研究科
e-mail: rin@mag.keio.ac.jp

重近範行‡
慶應義塾大学大学院政策・メディア研究科§
e-mail: nazo@mag.keio.ac.jp

1 はじめに

今日、ソフトウエアの国際化は、大きな課題の一つとなっている。しかし、さまざまな言語の文字を取り扱うことのできるソフトウエアの開発には、いくつかの障壁がある。そのひとつとして、世界中にはさまざまな文字コード表現が存在することをあげることができる。1文字が可変長の複数バイトで表現されるコード体系があり、そのなかにはJISコードのようにステートを持つものやISO10646/Unicode[1][2]のようなこれまでとはかなり異なったエンコーディング方法を用いているものも存在する。さらに文字の分類規則も言語によってかなり異なる。それでANSI Cでもこの様な多種多様な文字コードを扱うために、ロケールという考えが導入されたりしている。また、システムによっては複数バイト文字対応のC言語の関数ライブラリを用意しているものも存在する。しかしこれらもシステムによってかなり仕様に違いがあり、対応している文字コードも限定されており、十分なものとはいえない。

以上のような現状をふまえ、以下のような基本的要求を満たすC言語の文字処理関数ライブラリを開発した。

- ステートフルのものを含め、既存のほとんどの文字コードに対応する。
- 特定のシステムに依存しない。
- フリーソフトウエアとして公開する。

本稿では我々の開発したライブラリの設計と実装の概要について述べる。

2 ライブラリのアーキテクチャ

本ライブラリの基本構造は、ANSI Cのロケールモデルに基づくものである。図1に示すように、文字コードをファイルなどで用いられている外部コードとライブラリ内部の処理に用いる内部コード(ワイド文字コード)に分ける(以下では単に「文字コード」という場合は外部コードを指す)。ワイド文字コードは固定バイト長でステートレスの単一文字コードである。そしてワイド入出力関数(Wide I/O functions: 標準ライブラリのstdio関数に相当)の内部で外部コードと内部コードの変換を行ない、処理はワイド文字処理関数(Wide functions: 標準ライブラリのstring、ctype関数に相当)によって内部コードに対して行なう。内部コード↔外部コードの変換規則や文字分類規則などは、外部文字コードによって異なるが、これらはロケールデータファイルに記述し、これを読み込むことによって対応する。そのための関数が**setlocale**である。ロケールの指定は環境変数などによって行なう。

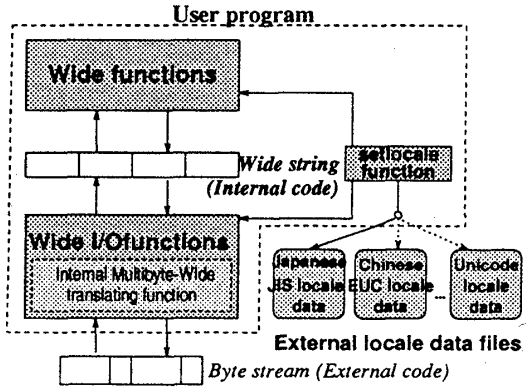

図1: ライブラリの構造

3 関数の仕様

用意する関数群はISO C Multibyte Support Extension(MSE)[3]の仕様に合わせたものとした。MSEは標準Cライブラリの文字(列)処理関数とほぼ1対1に対応するマルチバイト文字処理関数を規定している。またMSEは、ステートフルの文字コードの処理や文字分類の拡張方法も考慮されている点で本ライブラリ開発の目的に合致している。例えばprintfのMSE版として**wprintf**、strlenのMSE版として**wcslen**、拡張性のある文字分類関数として**wctype**、**iswctype**、文字変換関数として**wctrans**、**towctrans**などが規定されている。

*Development of the internationalized character and string function library
†Hiroshi Hayashi
‡Noriyuki Sigechika
§Graduate school of Media and Governance, Keio University

4 文字コード

対応すべき文字コードを大きく分類すると以下のようになる。

- ASCII、ISO8859-1···9
- ISO2022 ロックシフト準拠のもの (ISO2022-INT、JIS コードなど)
- ISO2022 シングルシフト準拠のもの (各国語 EUC)
- 独自規格 (シフト JIS、Big5 など)
- ISO10646 の変形 (UTF)

内部コードは、これらに含まれる文字をすべて含んでいなければならない。かつ上に述べたように 1 文字のバイト数が固定でステートレスであるべきである。これに適合するものとして ICODE[4] を参考にして、ISO10646 に日本語、中国語、韓国語の識別ビットを付加した 1 文字 4 バイトのコードを採用した。

5 外部コード↔内部コード変換

外部コードと内部コードの相互変換の方法としては、ビットシフトや演算による方法とマッピングテーブルを用いる方法が考えられる。このライブラリの場合、内部コードに ISO10646 の変形を用いたが、これは JIS コードなどとは文字をコードに割り当てる順番が全く異なる。そのため前者の演算などによる方法が用いることができない場合もある。そこで本ライブラリではマッピングテーブルによる方法を採用することにした。この方法により、柔軟なコード変換が可能になった。その反面、変換テーブルを関数内部に持たねばならないため実行時のプログラムのメモリ消費量が増大する欠点がある。

6 ロケールデータ

各文字コードごとの外部コード↔内部コード変換マッピングテーブル、文字分類テーブル (wctype、iswctype 関数で用いる)、文字変換テーブル (wctrans、towctrans 関数で用いる) は、ロケールデータファイルに記述する。それで新たな文字コードに対応しようという場合、基本的にはライブラリ自体の変更は必要ではなく、その文字コードにあったロケールデータを用意すればよい。ロケールデータファイルは標準的には/usr/local/lib/locale/ロケール名/LC_CTYPE というパス名で置かれる。テキスト形式で記述できるが、かなり大きなファイルになる場合があるので、実際に使用するファイルはユーティリティによって変換したバイナリ形式のファイルとした。プログラム中で **setlocale** 関数が呼ばれた時に、その引数もしくは環境変数 LANG(または LC_CTYPE、LC_ALL) に指定されたロケール名に対応するロケールデータをロードする。

7 結論

本ライブラリによって、文字コードの違いを意識することなく、C の標準ライブラリで英語用のプログラムを書く場合とそれほどかわらない労力でさまざまな言語の文字コードに対応できるプログラムを書くことが可能となった。しかし、処理スピードやメモリ消費量など改善すべき点も残っている。また、より効率的に処理するために内部コードの設計も見直す必要があると考えられる。今後は、ライブラリの改善とともに、このライブラリを用いたアプリケーションの作成、コンパイラの対応などを進める予定である。

8 謝辞

指導をして下さった村井純助教授、徳田英幸助教授に感謝する。また、必要な資料を提供して下さった齋藤信男教授に感謝する。そして実装についてさまざまなアドバイスを与えて下さった JUS の usrreq 研究会のメンバに感謝する。

参考文献

[1] Draft International Standard ISO/IEC DIS 10646-1.2,1992

[2] The Unicode Consortium, The Unicode Standard *Worldwide Character Encoding* Version1.0, Addison Wesley, 1991

[3] ISO/IEC JTC1/SC22, Proposed Draft Amendment 1 to ISO/IEC 9899:1990 Programming language C on: Normative Integrity Addendum, August 1993

[4] 太田昌孝、ＩＳＯ１０６４６と国際化された文字コードについて、 Proceeding of JWCC'93 in taiwan