Citrus iconv の実装

塩崎 拓也 Takuya SHIOZAKI *
<tshiozak@bsdclub.org>

2003 年 10 月 18 日

概要

本稿は，POSIX[1] において定義されている文字コード変換関数である iconv の Citrus Project[2] における実装について述べたものである．本稿では, Citrus iconv の実装についての詳細を述べ，本 iconv 実装の利点および欠点について議論する．

1 はじめに

iconv は Unix-like なオペレーティングシステムにおける標準的な文字コード変換 API である．今日では POSIX によってその仕様が規定されており，多くのアプリケーションが iconv を利用している．ところが，各 BSD には標準では iconv が存在しておらず，これを利用するためには例えば GNU libiconv[4] といった追加パッケージをインストールする必要があった．このことは，各 BSD が POSIX 準拠を謳う際の障壁になるだけではなく，さまざまなアプリケーションをインストールする際の手間の増加などの実用上の問題を生じさせる．

Citrus Project は各 BSD における国際化機能の実装を行うことを目的としており，iconv の実装もその目的の一つである．筆者により NetBSD に統合された iconv は，この成果である．

* Citrus Project[2], NetBSD[3] developer.

2 概念

Citrus iconv の実装について述べる前に，いくつかの概念について説明する．

2.1 コードセット (codeset)

この言葉は非常に漠然とした言葉だが，本稿では「ある具体的な文字の並び (文字列) をビットやオクテットなどの列で表現するための方法」と定義する．ここで，「ある具体的な」と注記しているのは，のちに述べる CES と区別するためである．コードセットの例としては，EUC-JP や ISO-2022-JP などが挙げられる．

2.2 CES と CCS

これらの用語は RFC2130[6] において定義され，Unicode TR17[7] において拡張されたものである．

2.2.1 符号化文字集合 (CCS)

コンピュータ上での文字集合は，各文字に一意なインデックス値を対応づけた文字の集合として表現される．このようなインデックス付きの文字集合のことを**符号化文字集合** *(Coded Character Set)* と呼ぶ．

たとえば, JIS X 0208 漢字集合は, 区点というインデックスづけが行われた CCS として表現することができる.

2.2.2 エンコーディングスキーム (CES)

一般的に, コードセットとは CCS を 1 つ以上含んだものである. 必ずしも一つの CCS のみを含んでいなければならないということはなく, たとえば, EUC-JP コードセットは US-ASCII, JIS X 0208, JIS X 0201 仮名, JIS X 0212 という 4 つの CCS を含んでいる. 各コードセットにおいて, 実際の文字列をどのようなバイト列で表現するかという方法を, **エンコーディングスキーム** *(Character Encoding Scheme, CES)* と呼ぶ. たとえば EUC-JP コードセットの文字列は EUC (Extended Unix Codeset) という CES で表現され, これは図 1 に示すようなバイト列により 4 つの図形文字集合を保持することができる. 同様に, Shift JIS コードセットの文字列は DBCS (Double Byte Character Set) Shift という CES で表現される[1].

G0 集合 $(0xxxxxxx)_2$
G1 集合 $(1xxxxxxx)_2$ ...
G2 集合 $(10001110)_2$ $(1xxxxxxx)_2$ ...
G3 集合 $(10001111)_2$ $(1xxxxxxx)_2$ ...

図 1: EUC エンコーディングスキーム

[1] [6] の記述は本稿で解説している CES を簡潔かつ明確に定義しているのに対し, [7] の記述は (おそらくは前者と同じものを定義しようとしているのだと推測されるのだが) 本稿における「コードセット」との区別が不明瞭である.

2.3 マルチバイト文字とワイド文字

CES の項で説明したようなバイト列で表現された文字列を処理する場合, バイト列表現のままでは文字の区切りなどがはっきりせず扱いにくいため, 一文字を 16bit や 32bit の整数値とした内部表現で扱うのが一般的である. たとえば ISO C locale の LC_CTYPE カテゴリでは, ある文字に対応する内部表現は一般的に wchar_t 型の整数値となるが, この場合には外部表現のことをマルチバイト文字列, 内部表現のことをワイド文字列と呼ぶ.

2.4 UCS 正規化と文字集合独立

ワイド文字列への変換の方法としては, *UCS* **正規化** *(UCS Normalization)* と**文字集合独立** *(Character Set Independent, CSI)* という二つの流儀が存在する.

UCS とは Unified Character Set の略であり, 広義には「あらゆる文字を含んだ大きな単一の文字集合」を意味し, 狭義には「Unicode[5] 文字集合」を意味する. UCS 正規化方式の国際化処理では, マルチバイト文字をコードセットに依存したワイド文字表現へと変換した後, これを UCS のインデックス値によるワイド文字表現へと「正規化」することによって処理の単純化を狙っている.

他方, CSI 方式の国際化処理では, マルチバイト文字からコードセットに依存したワイド文字表現に変換し, この形式のままで処理を行う. これにより, UCS という特定の CCS に依存しない柔軟な処理を狙っている.

Citrus の国際化フレームワークでは iconv サポートよりも先に LC_CTYPE のサポートを行っているため，マルチバイト文字とワイド文字との間で相互変換する部分がモジュール化されている．後で述べるが Citrus の国際化フレームワークは CSI 方式で作られているため，たとえ同じ符号化文字集合の同じ文字を表わす場合でも，一般にワイド文字の表現はコードセットに依存して異なる．たとえば，「あ」という JIS X 0208 の文字に対応するワイド文字の値は，UCS 正規化実装の場合には外部のコードセットにかかわらず常に同じ値を取るのに対し，CSI 実装である Citrus では外部のコードセットが EUC-JP と Shift JIS の場合でそのワイド文字の値が異なる (表 1)[2].

[2] この表において，例として UTF-8 などの Unicode 系のコードセットを取り上げていないのは意図的なものである．CSI の観点からは，JIS X 0208 集合における「あ」という文字と Unicode 集合における「あ」という文字は，別の CCS に属する文字なのでそれぞれ論理的に関連性がないものとして扱われる．この観点で言えば，この「フレームワーク上はもともと関連性のない文字」に関連性を与えることが iconv の仕事であると言える．一方，UCS 正規化方式では，これらの文字が「等しい」ことをフレームワークが保証していることになる．

3 iconv とは

iconv は，ある文字コードで記述された文字列を，それと (なるべく) 等価な別の文字コードの文字列へと変換するための API である．そのインターフェースの定義を図 2 に示す.

3.1 iconv の処理

iconv の処理は基本的に，

1. 入力バッファから一文字読み込み
2. 表変換などで出力側の文字コードへと変換し
3. それを出力バッファへと出力する
4. 1 から 3 を各文字について繰り返す

という順序で行われる．前述したように，文字列をマルチバイト表現のままで扱おうとするとその後の処理が難しくので，読み込まれた文字は何らかのワイド文字表現へと変換されることになる．そして iconv の場合にも，ワイド文字表現へと変換されたあとの文字の取り扱い方法としては，大きく分けて UCS 正規化と文字集合独立の二つが存在する．

コードセット	マルチバイト文字表現	ワイド文字表現
EUC-JP	0xA4 0xA2	U+3042
Shift JIS	0x82 0xA0	U+3042

UCS 正規化における表現

コードセット	マルチバイト文字表現	ワイド文字表現
EUC-JP	0xA4 0xA2	0xA4A2
Shift JIS	0x82 0xA0	0x82A0

Citrus における表現

表 1: 文字「あ」のワイド文字表現

```
iconv_t iconv_open(const char *dstname, const char *srcname);
```
コンバータを開き，そのハンドルを返す．

- dstname　変換先文字列の文字コード名
- srcname　変換元文字列の文字コード名

```
iconv_t iconv(iconv_t cd, char **src, size_t *srclen, char *dst,size_t *dstlen);
```
文字列を変換する．

- cd　コンバータのハンドル
- src　変換元バッファへのポインタ変数へのポインタ
- srclen　変換元バッファのサイズを保持する変数へのポインタ
- dst　変換先バッファへのポインタ変数へのポインタ
- dstlen　変換先バッファのサイズを保持する変数へのポインタ

```
iconv_t iconv_close(iconv_t cd);
```
コンバータを閉じる．

- cd　コンバータのハンドル

図 2: iconv API

3.1.1　UCS 正規化による iconv 実装

UCS 正規化方式の iconv 実装では，前述したように入力バッファから読み込んだ文字を入力コードセット依存のワイド文字表現へと変換し，これをさらに UCS のインデックス値へと正規化する．その後，この UCS のインデックス値を今度は出力側の文字コードにおけるワイド文字表現へと変換し，これをさらにマルチバイト表現へと変換して出力バッファへ書き込むこととなる (図 3)．

iconv の実装に UCS 正規化を利用することによるメリットは，変換のバリエーションの削減である．iconv がサポートする文字コードが n 個存在する場合，これらを入力および出力に取る変換のすべての組み合わせの数は n^2 となる．これを単純に実装しようとすると，いうまでもなく n^2 通りの変換処理を作成する必要がある．UCS 正規化方式では常に UCS を経由するため，必要な変換は「入力文字コード→UCS」の n 通りと「UCS →出力文字コード」の n 通りの計 $2n$ 通りの変換を用意すれば済むことになる．

一方で UCS に存在しない文字をうまく扱えないということと，あらゆる場面で一

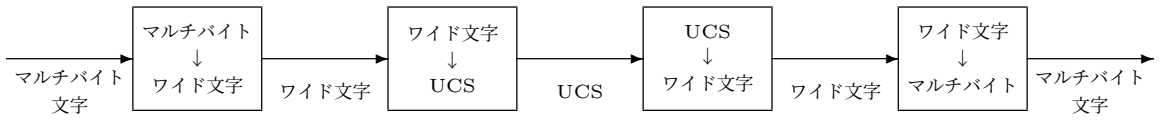

図 3: UCS 正規化実装における iconv の処理

旦 UCS を経由するための変換が必要になるというデメリットが存在する．

UCS 正規化を用いた実装の代表例が GNU libiconv である．

3.1.2 CSI による iconv 実装

一方，CSI による実装では，特定の文字集合に対する依存を排除した実装が行われなければならない．そのため，各文字に対応するワイド文字表現は，その時に選択されている入力側と出力側のそれぞれのコードセットに依存したままのワイド文字表現で扱われ，UCS 正規化のようにフレームワークとしてある特定の文字集合への正規化は行わず，そのインデックス値が直接変換される (図 4)．CSI の実装では，新規の文字集合の追加が容易である半面，実装が多少抽象的で複雑になる傾向がある．また，UCS 正規化のところで述べた通り，単純な方法では変換のバリエーションが文字コードの数の二乗となってしまう．そこで，いかにしてこのバリエーションを減らすかが CSI 実装の鍵となる．

既存の iconv 実装では，CSI なものの代表が Solaris の実装である．

4 設計目標

iconv を実装するにあたり，筆者は以下のような指針で設計を行った:

文字集合独立 あらゆる国際化関数を文字集合独立な形で実装するというのが Citrus の目標の一つであり，これは iconv も例外ではない．

コードの共有 iconv の行う処理は，他の国際化関数の機能と重なる部分がある．このような部分の共有を目指す．

拡張可能性 すべてのデータや変換方式およびエンコーディング方式の処理ルーチンは，アプリケーションやライブラリ (libc) 本体を再コンパイルことなしに追加できるようにする．

冗長な変換テーブルの削減 前述したとおり，内部に共通の CCS を持つコードセットの組が存在する．これらのコードセットから別のある単一のコードセットへと変換する場合に，それぞれのワイド文字表現に関連した別々の変換表を持っていると冗長なので，これを共通化する．

図 4: CSI 実装における iconv の処理

変換バリエーションの削減　上で述べたとおり，単純な方法では変換のバリエーションが文字コードの数の二乗となってしまう．これをなるべく削減する方法を採用する．

4.1　CSID, index, ESDB

EUC-JP と Shift JIS のように，同じ CCS のある文字に対して違う内部表現をもっているケースを考える．ここで，両方が共通して持っている文字集合，たとえば JIS X 0208 文字集合のある文字を, 別の文字集合, たとえば Unicode 文字集合の対応する文字へと変換しようとする際に，内部表現のままで扱おうとすると, EUC-JP と Shift JIS とで別々の変換表が必要になってしまい無駄である．

これを解決するために Citrus iconv では，ある一文字に対応する内部表現として単純なワイド文字ではなく，**文字集合識別子** *(Character Set Identifier, CSID)* と**インデックス** *(index)* という二つの 32bit 整数値のペアを用いる．ここで，どの CSID がどの CSID 値に対応するのかはそれぞれのコードセットおよび CES によって異なる．そのため，Citrus iconv では**エンコーディングスキームデータベース** *(Encoding Scheme Database, ESDB)* と呼ばれるデータベースによって，各コードセットの CSID と CCS 名の対応関係を保持している．図 5 に，例として EUC-JP の ESDB を示す[3].

CSID 値はコードセットに依存しているため，同じ CCS を表わすための CSID でも，コードセットが異なれば CSID 値は一般に異なる．一方で，同じ CCS の同じ文字に対する index 値は常に同一である．たとえば, 前節で挙げた「あ」という文字を例に取ると，EUC-JP の場合には (CSID, index) = (0x8080, 0x2422) であるのに対し，Shift JIS では (0x0002, 0x2422) となる．両者の CSID の値は異なるものの index 値は一致している．このような場合，それぞれの CSID が同じ CCS を示しているのかどうかは，それぞれのコー

[3] (実際にはこれをコンパイルしたバイナリ形式が用いられる．また，ESDB にはマルチバイト⇔ワイド文字変換モジュールの指定と，そのパラメータも含んでいる (「ENCODING」と「VARIABLE」がそれである).

```
NAME            "EUC-JP"
ENCODING        "EUC"
VARIABLE        "1 0x0000 2 0x8080 2 0x0080 3 0x8000 0x8080"
DEFCSID         "ISO646-US"             0
DEFCSID         "JISX0208:1990"         0x8080
DEFCSID         "JISX0201-KANA"         0x0080
DEFCSID         "JISX0212"              0x8000
INVALID         0xA2AE  # GETA
```

図 5: EUC-JP の ESDB

ドセットに対応した ESDB を参照して判別する．この例では，EUC-JP における 0x8080 と Shift JIS における 0x0002 が両方とも「JISX0208:1990」という名前であるということからこれが等しいものだと判断できる．

このような方法により，Citrus iconv ではコードセットから CCS と CES を分離している．分離された後の index 値は，もともとのコードセットとは独立した形で利用することができる．先の例では，元のコードセットが EUC-JP であろうが Shift JIS であろうが，その index 値 0x2422 を JISX0208 文字集合の「あ」という文字を表わすものとして統一的に扱うことができるのである．

なお，マルチバイト表現と (CSID, index) ペアとを相互変換する部分は，既存の locale の LC_CTYPE カテゴリを再利用している．

4.2 文字集合マッパー

CSID と index のペアに分離されたあと，ある CSS の index 値を別の CSS の同じ文字の index 値へと変換するのが**文字集合マッパー** *(Character Set Mapper, CSMapper)* である．たとえば, JIS X 0208 文字集合を Unicode 文字集合に写像したり，その逆を行ったりする．文字集合マッパーは，入力および出力の CCS の名前から対応する写像を見つけるためのデータベースと，実際に変換を行うマッパー部の二つからなる．マッパー部はプラグインで追加できるようになっている．

4.3 ピボットコード

文字集合マッパーは，入力および出力の符号化文字集合の名前を与えるとそれに対応するマッパーを開くしくみになっている．そのため，単純な方法では符号化文字集合の二乗の数のマッパーが必要になる．一方 UCS 正規化方式では，すべての変換は一度 UCS を経由することによって表の数を抑えている．

Citrus iconv では，UCS 正規化方式のこの利点を取り入れるために，**ピボットコード**という概念を導入している．文字集合マッパーに付随するデータとして**ピボットデータベース**というものを持っており，これは，マッパーによってたがいに隣接している CCS と，その変換の**距離** *(norm)* を記述したものである．文字集合マッパーに与えられた二つの CCS に対応する直接の変換が存在しない場合でも，このピボットデータベースから中継可能なコード (ピボットコード) が見つけられれば，そのコードを経由して自動的に変換が行われる．複数のピボットコードの候補がある場合には，距離の合計が一番小さなものが選ばれる．なお，処理が遅くなりすぎることを防ぐため，多段階の中継は行わず，たかだか一段のピボットコードが使われるように制限されている．

まとめると，ピボットコードは UCS という概念を CSI に則した形で拡張したものであり，逆に，UCS 正規化はこのピボットコードの特殊な例ともいえる．

4.4 マッパーの集成

二つのコードセット間で変換を行う場合，両者がそれぞれ含んでいるすべての CCS に対応するマッパーを見つけ出す必要がある．これを**マッパーの集成**と呼ぶ．

Citrus iconv では，入力側と出力側のコードセットに対応する ESDB がそれぞれのコードセットが持つ CCS のリストを保持しているため，このすべての組み合わせについてマッパーを開くことにより，マッパーの集成の自動化を行っている．図 6 に, EUC-JP を Unicode 系のコードセット (UTF-8 など) に変換するケースでの，集成後のブロック図を示す．

もし，入力側のある一つの CCS に対して，出力側の複数の CCS にそれぞれ対応するマッパーが存在する場合には，出力側の ESDB に記述されている CCS の順序にしたがって順に変換が試みられ，最初に成功した変換が採用される (図 7)．

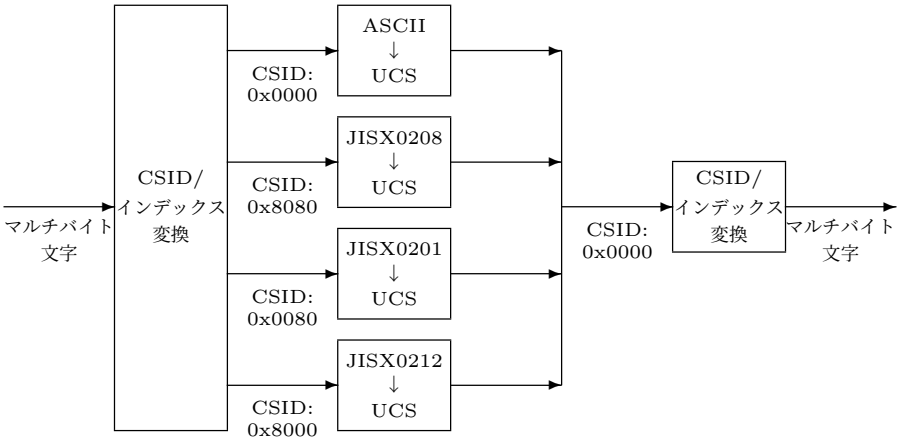

図 6: EUC-JP → UTF-8 変換時のマッパーの集成

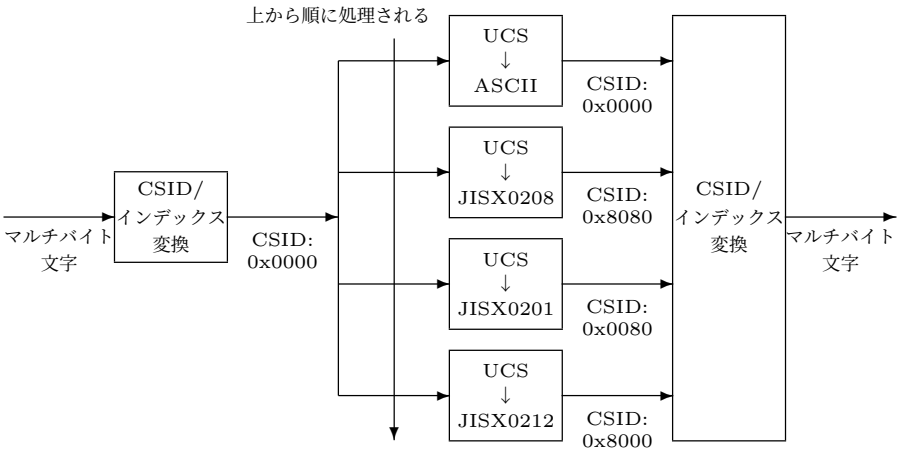

図 7: UTF-8 → EUC-JP 変換時のマッパーの集成

5 実装

Citrus iconv は，これまで述べたようなものをこれを iconv の API に則した形で組み合わせて実装されている．実装は既に NetBSD の公式なソースにマージされており，最新の NetBSD-current を取得することにより利用できる．現在利用可能なコードセットは，ISO-8859 系や EUC 各種などを含む約 130 種類である．現在はさまざまなアプリケーションで利用されており，多くの問題は修正ずみで現在は特に大きな問題は報告されていない．

6 ベンチマーク

Citrus iconv と GNU libiconv でそれぞれ 4 つのケースでの簡単なベンチマークを行った結果を表 2 に示す．内容はそれぞれ，

1. ISO-2022-JP から UTF-8 への変換を単に開いて閉じる
2. ISO-2022-JP の文字列「あいう abc」の 500,000 回繰り返しを UTF-8 へ変換
3. 同様に EUC-JP から UTF-8 への変換
4. 同様に Shift JIS から UTF-8 への変換

を行うためにかかった時間である．

これを見ると，どの項目も GNU libiconv よりも性能が良くない．特に項目 1 が顕著である．これは，変換の初期化に時間がかかることを意味している．その他の項目は実際の変換の性能差を示しているが, おおむね数倍の差となっている．GNU libiconv がすべてのデータを静的に保持しているなど，拡張性や柔軟性という点で劣っているのに対し，Citrus iconv はプラグインによる拡張性や，これまでに述べた各種機構による柔軟性の確保のために速度が犠牲になっている．これは純粋に速度と柔軟性のトレードオフであると言える．

7 結論

設計目標に挙げた三つの点について検討すると，これまでに述べたような方法で CSI な iconv を作成することが可能であったこと，プラグインなどによる拡張性の確保，そして，ISO C locale の LC_CTYPE 部などとのコード共有が行われており，当初の目標を達成したことが確認できた．一方で，GNU libiconv と比べて軒並み性能が良くないという点は，今後解決すべき課題であろう．

項目	GNU libiconv(秒)	Citrus iconv(秒)
1	0.005	0.191
2	0.734	3.899
3	0.639	2.738
4	0.679	2.692

環境:富士通 FMV-BIBLO LOOX S80C(Transmeta Crusoe TM5800 800MHz)

表 2: ベンチマーク

8 謝辞

Citrus iconv を作る土壌を与えてくださった NetBSD Project の方々や, 完成した Citrus iconv のテストに関係してくれた方々に感謝します．特に Thomas Klausner 氏にはマニュアルページの構成やバグレポートの捕捉などを行っていただきました．また，埜中公博氏の報告が性能改善のヒントとなりました.

参考文献

[1] IEEEStd 1003.1-2001 POSIX / Single UNIX Specification Version 3.
http://www.unix-systems.org/

[2] Citrus Project.
http://citrus.bsdclub.org/

[3] NetBSD Project.
http://www.netbsd.org/

[4] GNU libiconv.
http://www.gnu.org/software/libiconv/

[5] Unicode Consortium
http://www.unicode.org/

[6] The Report of the IAB Character Set Workshop
held 29 February - 1 March,1996
RFC2130.

[7] Character Encoding Model
Unicode Technical Report No.17.
http://www.unicode.org/unicode/reports/tr17