LI18NUX2000適合テストスイートの開発

知念 充, 吉田 忠行, 鷲澤 正英

日本アイ・ビー・エム株式会社 ソフトウェア開発研究所

1. はじめに

Linux[1]上で扱われるアプリケーションは，これまでシングルバイト処理を前提としたコードをベースにしていたこともあり，マルチバイト処理については各言語／各地域においてのみ有効なパッチとしての対応に留まっていた。このため，Linuxの国際化は，他の商用OSに比べて遅れていた。

また一方で，Linux上で国際化対応を進めているアプリケーションが徐々に増えてきてはいる。しかし，POSIXの定めるロケールモデル[1]に基づかず，独自の実装で国際化対応を進めているものが少なくない。独自の方法による対応では，各アプリケーションの開発者は各言語・地域に対する深い知識が要求されることになり，開発作業において大きな負担を強いられることになる。また，各開発者の国際化に対する認識のずれにより，アプリケーションごとに異なる動作をする恐れがある。

以上の問題は，各アプリケーションの基盤となるライブラリやユーティリティの国際化が立ち遅れていたことが原因として考えられる。したがって，問題の解決には，この基盤となるライブラリやユーティリティの国際化機能を確固たる物とすることが必須である。

こうした状況を打開するために，Free Standards Group[2]の1コミュニティであり，Linuxにおける国際化の推進を目標とするLi18nux[3]は，2000年夏にLinuxがアプリケーションプラットフォームとして準拠すべき国際化仕様をLI18NUX2000国際化規約[2]として策定し発表した。

そして2001年の初め，LI18NUXはこの規約に対するディストリビューションの適合度を測るため，第1回目の適合認定パイロットプログラムを行った。

しかし，この認定作業は難航した。LI18NUX2000国際化規約はPOSIX, Single UNIX Specification Version 2 など複数の規格を参照する箇所が多量に含まれており，確認すべき項目が膨大な量に上ったためである。結局，この認定作業には複数人で一週間程度費やすこととなった。正式プログラムが稼動した際には複数のディストリビューションを何度もテストすることになり，一度のテストにこのように多量の時間を費やすことはできない。

そこで我々は，適合認定を短時間で行うためのオートマチックテストスイートを開発した。このテストスイートにより，適合判定に要する時間を1時間程度に短縮することに成功している。

本稿ではこのテストスイートの概要・開発上の工夫点・実行結果・考察を報告する。2章ではLI18NUX2000国際化規約および国際化適合認定について概略を述べる。3章ではテストスイート全体の話題として，テストフレームワークの選定，検査項目であるアサーションについて述べる。4,5,6章では，テストスイート開発にあたり特に工夫した点である，ロケール機能，テキスト入力，テキスト出力のテスト方法について述べる。そして，7章にて本テストスイートの結果と効果を報告し，最後にまとめを述べる。

---

[1] 本論文ではLinuxはカーネルのみでなく，カーネルと周辺アプリケーションをあわせたOS環境を示す。

[2] Free Standards Group(FSG)は，米国の非営利団体で，Linux以外も含めたオープンソース技術の標準化および技術開発を行なう。http://www.freestandards.org/ 参照。

[3] Li18nux (Linux Internationalization Initiative), http://www.li18nux.org/ 参照。

2. LI18NUX国際化規約と適合認定

2.1 国際化規約

LI18NUX2000国際化規約では，国際化されたアプリケーションを実行するために，OSにおいてサポートされなければならないインターフェースや機能を次の9カテゴリに分類し，それぞれに対して準拠すべき内容を定義している。

1. Base Library
2. Shells and Utilities
3. Programming Language
4. Graphical User Interface
5. Input Methods
6. Output Methods
7. Network Servers
8. Internet Tools
9. Printing

従来規格との互換性のため，それぞれのカテゴリの内容には他の規格を参照するよう記述されている箇所が多く含まれている。例えば"1.Base Library" では，「この関数は，ISOのC言語規格(ISO/IEC 9899)の国際化に関する規定に従うこと」等の記述がある。

またLI18NUX2000国際化規約では，システムを国際化する作業の困難さから，国際化の要求を次の2つのレベルに分けて定義している。

レベル1: 規約制定時点(2000年8月)において，すでにオープンソースの実装が存在する。もしくは，十分に安定しているオープンソースのβ版の実装が存在する。

レベル2: 規約制定時点において，オープンソースの実装が多少足りないか，LI18NUX2000国際化規約の要求を満足する実装は存在するが，それはオープンソースではない。

2.2 国際化適合認定

Li18nuxでは，LI18NUX2000国際化規約に基づいてLinuxディストリビューションの国際化適合度を判定し，適合が確認されるとそのディストリビューションに対し，認定証を発行する。この認定には先に述べた国際化規約のレベル分けに対応して，適合レベル1と適合レベル2の2つのレベルが存在する。

この国際化適合認定は次のプロセスで行われる。

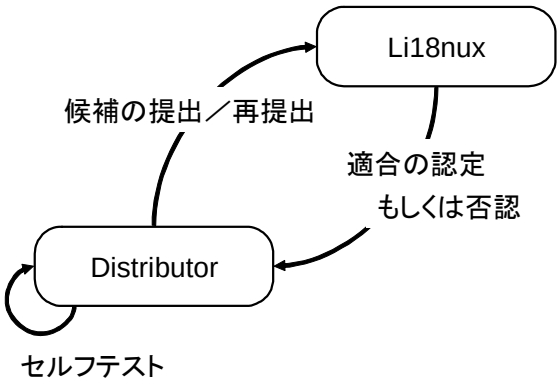

図1. 国際化適合認定プロセス

1. ディストリビュータが，認定候補のLinuxディストリビューションをLi18nuxに提出
2. Li18nuxにおいて適合テストを実施
   - 合格なら認定証発行
   - 不合格なら問題箇所を提示
3. ディストリビュータは不合格だった際，問題箇所を修正，セルフテストを行った後再提出。

適合レベル1の認定を受けたディストリビューションにはLI18NUX2000適合レベル1適合のロゴの使用が，適合レベル2の認定を受けディストリビューションにはLI18NUX2000適合レベル2適合のロゴの使用が，それぞれ認められる。

現在，適合認定パイロットプログラムが行われているが，このパイロットプログラムにおいて，既に認定を受けたディストリビューションには，パイロットプログラムにおいて適合認定を受けたことを示すロゴ(図2)がパッケージに記載されている。

図2. LI18NUX2000 Level.1 パイロットプログラム適合認定ロゴ（サンプル）

3. テストスイートの基本構成

3.1 開発コンセプト

国際化テストスイートにおいて，次の3つの点を念頭におき，開発作業を行った。

1. 国際化のテストにかかる時間を短縮する
2. 問題点が絞り込めるよう検査項目をできる限り詳細に分ける
3. Linuxディストリビュータが，自身のディストリビューションの国際化適合度を確認できるものにする

3.2 テストフレームワークの選定

国際化テストスイートでは，テストの枠組みとしてTET/VSXテストフレームワークを採用した。TET[4] は，もともとOSF，UNIX International およびX/Openによって開発された，マルチプラットフォーム対応のテスト用フレームワークである。一方，VSX[5]は，TETのフレームワークを利用し，X/Openが使い易くカスタマイズしたTETを包含したフレームワークである。

このフレームワークの選定理由は，次の3点である。

1) C, C++, sh等多くの形式でテストを作成できる

テスト対象は，CライブラリやXライブラリのAPIやUNIX基本コマンドなど多岐にわたるため，C，C++，shなど多くの形式でテストケースを記述できる自由度の高いテストフレームワークが必要であった。

2) マルチプラットフォームに対応している

比較のため，Linux以外のプラットフォームでのテストを実行することも考慮してマルチプラットフォームのテストを可能とするポータビリティのあるテストフレームワークが必要であった。

3) FSG内で使用するテストのフレームワークが統一できる

Li18nuxと同様にFSGの1コミュニティであるLSB[6]は，POSIX.1やFHS2.1等の仕様に対する適合テストスイートを開発している。これらのテストスイートの開発に当たり，LSBはTET/VSXのテストフレームワークを採用していた。LI18NUX2000国際化規約に対するテストスイートに，同一のテストフレームワークを採用することで，テストの前提知識，操作方法，結果の読み方などの必要な知識を統一することができる。

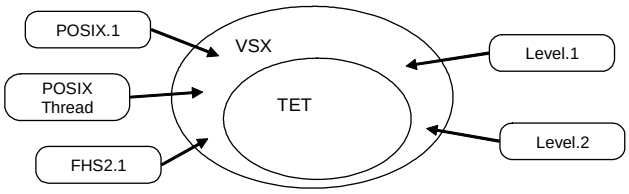

図3. FSG内のテストスイート関係図

3.3 アサーション作成

テストスイート開発にあたり，問題点を明確にするための検査項目[7]（以降アサーションと称する）をまとめた。

アサーションは，問題点をできるだけ細かく提示できるよう，LI18NUX2000国際化規約およびこの規約が参照している他の規格の内容をできる限り詳細に分解したものとなっている。各テストはこのアサーションに一対一対応し作られている。

既にパイロットプログラムが行われているレベル1のテストにおいては，レベル1としての規定がある6カテゴリに渡り，約1,200のアサーションが作成された[8]。

現在，開発中のレベル2のテストにおいては，アサーション数は約1,800に達する見込みである。

---

[4] TET(Test Environment Tools), http://tetworks.opengroup.org/Products/tet.html 参照
[5] VSX (Validation Suite for X/Open), http://www.opengroup.org/testing/testsuites/VSXgen.htm 参照
[6] LSB(Linux Standard Base)はLinuxディストリビューションの互換性向上のための標準を策定するコミュニティ。http://www.linuxbase.org/ 参照
[7] 全てのアサーションはLi18nuxサイト（http://www.li18nux.org/subgroups/testsuites/assertion/index.html）で公開。
[8] 現在行われているパイロットプログラムは，LI18NUX2000国際化規約追補2を参照している（現在追補4）。そのため，アサーション数が約1,400と異なっている。

4. ロケール機能のテスト方法

4.1 既存ロケールによるテストの問題点

POSIXの定義では，ロケールとは国，言語，習慣によって違うルールや情報をまとめたものである[3]。各ロケールはロケール・データベースに蓄積されており，実行時にユーザーがロケールを選択すると，対応するロケールデータがデータベースから取り出される。

すなわちロケールとは，アプリケーションを国，言語，習慣に依存するメッセージやコードセットなどの部分と，非依存であるロジックの部分とを切り分けるためのものであり，国際化プログラミングの根幹となるものである。

ライブラリやユーティリティの国際化機能をテストする際，人的リソースや時間的制限からテストの範囲を，使用される頻度の高い言語に絞ったり，テスターの使用する言語のみで済ませてしまうケースがある。

LI18NUX2000国際化規約においても，ロケールに対する要求においてコードセットこそ最小限の要求をUTF-8としているものの，120種以上のロケールをサポートすることとしている。これら各ロケールそれぞれに対してテストを作成することはあまりに煩雑な作業であり，テスト対象となるロケールの絞りこみは有用であるように思える。

しかし，このように対象とするロケールを絞り込んだテストに合格しても，その言語ではたまたまテストを合格したという偶然性を排除することができず，潜在的なバグを発見できない可能性がある。

例えば，国際化機能として実装された部分が，

```
if (LANG = "en_US") YESEXPR = "Yes";
else if (LANG = "ja_JP")  YESEXPR = "はい";
else ....
```

というように，ロケールの名前とそのロケールにおけるリソースの値が直接コーディングされているとする。

このとき，国際化機能のテスターが，“ja_JPロケール環境のもと，YESEXPRの表示文字列が，「はい」と表示されること”を国際化機能の判定基準としている場合には，その実装が国際化ではなく単なるその場しのぎの地域化に留まっていることを見抜くことはできない。

4.2 擬似ロケールの有用性

そもそもISOのC規格やPOSIXで定められた国際化関数は，ロケール・データベースに登録された各ロケール固有のリソースを参照することで機能する。したがって，特定の自然言語によらない値を各リソースに定義したロケールを作成し，その値を正しく参照しているかどうかを調べることで，各ライブラリやユーティリティの国際化機能の検証ができ，さらに先述の例のように国際化部分と地域化部分を分離していない実装箇所を抽出することが可能となる。このようなロケールを擬似ロケールと呼ぶことにする。

ライブラリやユーティリティの国際化が正しくなされていることを検証する要件をまとめると以下のようになる。

1. ロケールデータ生成機能の動作確認
2. 擬似ロケールにおける国際化機能の動作確認

この2つを厳密にテストすることにより，任意のロケールにおいて，国際化機能が正しく動作することを論理的に保証できる[9]。

4.3 擬似ロケールの定義方法

前節に述べたように，実装が正しくロケールのリソースを参照していることを確かめるには，特定の自然言語に拠らないように各リソースの値を定める必要がある。したがって，擬似ロケールは次の点を考慮し作成されなければならない。

1. リソースの値は，複数言語の文字を含む
2. リソースの値は，さまざまなバイト長のマルチバイト文字を含む
3. サポートするどの言語のリソースにもないものでリソースの値を定義する。すなわち，自然言語にて意味不明な文字列をあえて用いる
4. 文字クラス（空白文字，数字，アルファベット

[9] もちろん，このテストでは各ロケール・データのバグまでは調べることはできない。しかし，これは国際化後の地域化におけるバグであり，別途テストするべきであろう。

など）は複数言語の文字を含む

5. 文字クラスには，どの自然言語でもその文字クラスに属さない文字を含む
6. どの自然言語にも存在しない文字クラスを定義する
7. 文字の照合順序は，コードポイントの並びとも，自然言語の並びとも異なるものとする

今回作成したテスト用ロケールで上記の点が反映されている箇所の例を次に示す。

a. 曜日の表記（nl_langinfoでいうDAY_{1-7}の戻り値）を以下のようにした

   ±, 朒ﾞ愁, ‡ψ, E™, p, ⇒vB, あ

b. 空白文字(space)クラスに，全角アンダースコア（＿）を追加した
c. pfeel という ！(EXCLAMATION MARK)のみの文字クラスを作成した
d. 数字の文字としての照合順序を昇順とした

aは考慮点の1，2，3に，bは4，5に，cは6に，dは7にそれぞれ該当する。

4.4 実施例と効果

ここでは，GNUのtextutilsパッケージに含まれるwcユーティリティを例に擬似ロケールにより得られた効果を示す。

1) wcユーティリティについて

wcユーティリティは，以下のオプションを持つ。

- -c バイト数のカウント
- -m 文字数のカウント
- -w 単語数のカウント。単語は空白文字により区切られる。
- -l 行数のカウント

GNUのtextutilsパッケージのwcユーティリティは当初シングルバイトのみを考慮した実装となっていた。すなわち，1バイトを1文字として考えており，ロケールのコードセットがeucJPやUTF-8であった場合も -m オプションがバイト数を返すという正しくない動作をしていた。

textutilsのバージョン2.0.8においてロケールを参照するようになり，-m オプションが正しく文字数をカウントするようになった。すなわちロケールのコードセットがeucJPやUTF-8であっても，-m オプションで，バイト数ではなく文字数が正しく返る。他のオプションでも期待通りの値が返り，これで国際化が終了しているように思われた。

2) 擬似ロケールの適用

我々の定義した擬似ロケールを用い，wcユーティリティを空白文字にしたがって単語数をカウントするオプション -w を用いテストした。このとき，対象としたテキストファイルには，図4の文字列を使用した。

Paj＿ńiïŧ＿ЖÑ≡Hж＿りじざ＿ャメ＿㊔㈲㊃＿22BÞFẐラ＿9／ャJ

図4. 複数言語の文字を含むテストケースの例

擬似ロケールにおいて，空白文字クラスに`＿'が属しているので，実行結果は

```
$ wc -w target.txt
   8 target.txt
```

となるはずである。しかし，単語数として1が返った。

これにより -w オプションをつけた際の戻り値は，ロケールで設定されている空白文字をセパレータとするように動作していないことが判明した[10]。

擬似ロケールを使ってロケールの空白文字クラスの定義を参照していないというwcの国際化機能上のバグを発見できた。

これにより擬似ロケールを使えば，開発者の想定範囲外のバグも正しく抽出できることがわかる。

[10] textutils-2.0.21を調べてみたところ，このバグは修正されていた。

5. テキスト入力のテスト方法

5.1 テキスト入力におけるテストの問題点

LI18NUX2000国際化規約において，Input Methodsのカテゴリは，X window環境およびその他の環境で各言語の文字を入力する方法を規定している。LI18NUX2000国際化規約のLevel. 2での要求は，Unicode3.0全体を入力できることである。

第一回の認定パイロットプロジェクトでは，手動によるコード入力を行っており，非常に多くの時間を費やしていた。そこで，自動的に入力するテストプログラムの作成を検討した。TET/VSXフレームワークを利用しない独立したプログラムとなってしまったが，テストを自動化することを可能としたので，本章で述べる。

5.2 自動入力方法の決定

Input Methodsの種類やアプリケーションに依存した形でのテストプログラム作成は，今後の保守やプログラム変更の手間が増える可能性があるため，できる限り汎用的な方法を模索した。

まず，標準のXlib関数であるXSendEvent()を使い，実際にキープレスを行わずにキー入力イベントをシミュレートさせることにした。ところが，XSendEvent()によって送信されたイベントをアプリケーションにの実装によって無視する場合があること[11]がわかった。

次にXのエクステンションの1つであるXTESTエクステンションを用いるアプローチを検討した。XTESTエクステンションは，Xサーバーをユーザーが介さずにテストするために必要なクライアントとサーバーのエクステンションの最小限のセットである。これは，必ずしも全てのXサーバーでサポートされているとは限らないが，Linuxディストリビュータが多く採用しているXFree86サーバーではサポートしていることがわかり，このアプローチでプログラムを作成することにした。

5.3 構成と原理

自動入力プログラムは，「自動キー入力プログラム」と外部ファイルである「入力キーデータ」の二つより構成されている。

入力キーデータファイルには，入力させたいキーを記述する。このデータはアプリケーションにあわせて作成する必要がある。自動入力プログラムは，これを読み込み，XTestFakeKeyEvent()を用いることで，キー入力から生成されるシーケンスと同じキーシーケンスを生成する。

5.4 実施と効果

LI18NUX2000国際化規約Level.2の要求を満たすIMとしてIIIMXCF[12]のxiiimp.soを候補に上げ，このテスト用のデータファイルを作成し，実行した(図5)。

図5. 自動入力画面

これにより，文字入力確認作業にかかった時間を劇的に減少し，効率よくテストが行えることが確認できた。例えば，100文字をコードポイントによって入力する場合で比較すると，手動による文字入力の場合，個人差はあるにせよおよそ200秒程度かかる。これに対し，自動入力プログラムの場合は，約45秒程に短縮でき，手動に比べると約4分の1になった。

また，大量のコードポイントを人の手で打つと，どうしてもミスが混入する。これを排除することもできた。

---

[11] XAnyEvent構造体のsend_eventフラグがセットされている場合。
[12] IIIMXCF(IIIM X Client Framework - IIIMはInternet/Intranet Input Method), http://www.li18nux.org/subgroups/im/ 参照

6. テキスト出力のテスト方法

6.1 テキスト出力に対する要求の整理

LINUX2000国際化規約のOutput Methodsのカテゴリは X Window System上で各言語の文字を表示する方法について規定している。

Output Methodに対するLI18NUX2000国際化規約におけるLevel. 2の要求は，X Window SystemにおいてISO/IEC 10646-1:2000で定義されたUCS implementation level1をカバーするマルチバイト文字およびワイド文字のインターフェースを持っていること，とされている。 これは，Xlibの描画関数である，XmbDrawString() や XwcDrawString() を使い，定義された各コードポイントの文字が表示できることを要求しているに等しい。

従来行われたテストでは，それぞれの関数を使って表示させた文字を目視により確認していたため，非常に長い時間を費やし，また見間違える可能性がかなり高かった。そこで，描画した文字を自動的に確認できる方法を検討した。

6.2 テスト方法の検討

表示文字の確認には各ディストリビューションが提供するフォントでは適していない場合が多い。

たとえば，そのディストリビューションが日本向けのものだった場合，JISの定めるいくつかの文字集合に含まれる文字に対してしか，フォントにグリフイメージが含まれていない場合がある。LI18NUX2000国際化規約においてOutput Methods に対する要求は，「描画できること」と能力を問うており，フォントにグリフイメージが含まれているかは要求していないため，すべての文字にグリフイメージが含まれていなくても問題はない。しかし，グリフイメージがなければ描画判定はできない。

そこで，テスト用に16ドットのBDFフォントを作成し，それを用いて描画確認を行うこととした。この際，作成したフォントのグリフイメージはそのイメージからコードポイントを解釈できるように16進のコードポイントとなっている。これは，もし本テストで不具合が発見され個別テストを行わなければならない場合，目視確認をしやすくするためである。

しかし，テストの仕組み自体はフォントから独立しているべきである。したがって，作成したフォント以外でもテストできるような実装方法を考案した。

6.3 実装方法

本テストは以下の順序でOutput Methods の自動検証を行う。

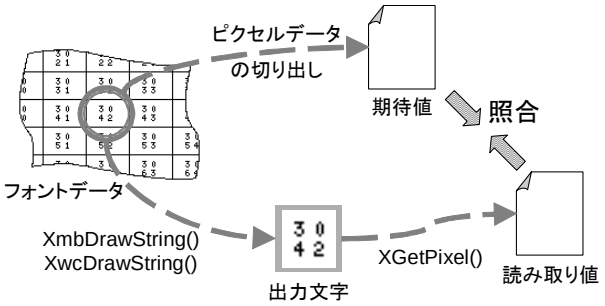

図6 Output Methods の検証の流れ

1. テストに用いるフォントから，各文字のグリフイメージをピクセルデータとして取得
2. Xlibの描画関数(XwcDrawstring,あるいはXmbDrawString())を使って文字を描画
3. XGetPixel() を使って描画文字のピクセルデータを取得
4. 1で作成したピクセルデータと3で取得したピクセルデータを照合

これにより，目視確認で行っていた表示の確認作業に費やす時間を劇的に減らすことができた。また，目視確認による判断ミスの混入を排除できた。

7. 国際化適合テストスイートによる効果

7.1 国際化適合認定に対する効果

第一回目の国際化適合認定パイロットプログラムでは，確認項目が相当な数に及ぶにもかかわらず，手作業による簡易的なテスト結果と収集した情報とをつき合わすことにより行われた。また100を超えるロケール全てについて調べることはできず，代表的なロケールの幾つかを対象に行われた。それにもかかわらず複数人で約一週間を要した。

それに対し国際化テストスイートでは，約1時間程度で自動的にテストを実行することができる。やがてパイロットプログラムが終わり，正式に国際化適合認定プログラムが始動するが，多くのディストリビューションが適合認定プログラムに望んでも，本テストスイートの使用により手際よく各ディストリビューションの認定作業をこなしていくことができるだろう。

またテストの厳密度の面でも，大幅に向上している。第一回適合認定パイロットプログラムで合格となったライブラリ関数やユーティリティでも，国際化テストスイートでは不合格になったものが多数あり，多くの人為的判断ミスがあったことがわかった。

このことから人による判断ミスを認定プロセスから極力排除できるという点でも効果があるのがわかる。

7.2 その他の効果

ここではテストスイートを使って，適合認定作業の効率化以外に為し得た事例を示す。

本国際化テストスイートにおけるBase Libraryのテスト群をLinuxディストリビューションで主として使われているCライブラリであるGlibc[13]に適用し，発見された不具合をGlibcのメンテナであるUlrich Drepper氏にバグレポートをした。

このバグレポートを元に，Ulrich Drepper氏に国際化における不具合を修正して頂いた。

Glibcのバージョンと国際化機能における不具合の数を表1に示す。

表1. Glibcにおける判定結果推移(Total: 875tests)

Version	2.2.3 (2001/04/27)	2.2.4 (2001/08/16)	2.2.5 (2001/01/20)
Succeed	824	858	864
Failed	39	6	0
Unsupported	0	0	0
Unresolved	1	0	0
Unreported	0	0	0
Untested	3	3	3
Further Information	8	8	8

表中の結果の各カテゴリは以下のようになっている。

- Succeed
  成功したテストの個数を示す。
- Failed
  失敗したテストの個数を示す。
- Unsupported
  LI18NUX2000規約において，最初から満たしている必要はないが，どうやって要求を満たすのか，その方法を明示しなければならないものの個数を示す。
- Unresolved
  何らかの原因でテストが異常終了し，テスト自身の結果がわからないもの個数を示す。
- Unreported
  なんらかの原因でテストが異常終了し，かつその原因がわからなかったものの個数を示す。
- Untested
  LI18NUX2000国際化規約にて定められているが，テストのしようがないものの個数を示す。
- Further Information
  厳密にいうとLI18NUX2000国際化規約に適合していないが，実運営上何の問題もない箇所であるものの個数を示す。

上記結果から，バージョンが更新されるにつれてGlibcの国際化上の問題は修正され，不適合箇所が減少していることがわかる。したがって最新のGlibcでは，その国際化レベルが一定水準に達し，アプリケー

[13] the GNU C Library, http://sources.redhat.com/glibc/ 参照。

ションのポーティングに十分耐えられるようになったといえる。

7.3 実行結果の考察

国際化テストスイートを，あるLinuxディストリビューション上でバージョンごとに実行させた結果，GUIや入出力機構などにおいてもLinuxの国際化の度合いは徐々に高くなってきていることがわかった。しかし，幾つかの点でまだ十分ではない部分がある。代表的なものとしては，

- 日時の表示がロケール非依存
- 等価クラスが判定不可能
- 複数文字照合要素が判定不可能

が挙げられる。

しかしライブラリやアプリケーションのいくつかは，LI18NUX2000の参照している規格が曖昧な点や，実装するには下層のライブラリが足りない，等の理由で上記の問題を持っているものもあった。これらの点に関してLI18NUXワーキング・グループは適合判定時における免除項目にする等，別途対応を決めている[4]。

8. まとめ

今回，Linuxにおける国際化適合度を評価する国際化テストスイートを開発した。

これにより，第一回パイロットプログラムでは，複数人で約1週間を要した認定プロセスを，約1時間程度で自動的にテストを実行することができ，国際化適合認定プロセスを短時間で行なえるようになった。

また，Linuxオペレーティングシステム上における国際化における不具合を容易に特定することができるようになった。

加えて，各ディストリビュータの手により簡単にテストが行なえるようになり，不十分な国際化部分の把握および修正後の確認作業が容易になった。

以上の点から，今回開発した国際化テストスイートによりLinuxの国際化機能対応の基盤の強化が図れ，Linuxの国際化対応の更なるスピードアップが期待できるだろう。

― 謝辞 ―

POSIX等の仕様の解釈において，数多くの助言をして頂いたLi18nux System Architecture Subgroupの方々に深く感謝いたします。特に沼田 利典さんには，時々規格を曲解してしまう私たちの疑問点を合理・不合理に関わらず辛抱強く聞いてくださいました。大変ありがとうございます。

また英語の拙い私達の意見に対し，辛抱強く，かつ迅速に対応してくださった，GlibcメンテナのUlrich Drepper氏に厚く御礼申し上げます。

最後に開発に際し，多くの協力をいただいた日本アイ・ビー・エムの杉山 昌治氏，関場 治朗氏，長谷川 勇氏，稗方 和夫氏，木戸 彰夫氏に感謝いたします。

― 参考文献 ―

1) ISO/IEC 9945-2:1993 Information technology － Portable Operating System Interface (POSIX) － Part 2: Shell and Utilities, Chapter 2, Section 5

2) LI18NUX2000 Globalization Specification (http://www.li18nux.org/li18nux2k)

3) 末廣 陽一，清兼 義弘，“国際化プログラム I18Nハンドブック”，共立出版(1998)，7章

4) 木戸 彰夫，“Linuxの標準化通信:第6回 国際化仕様と適合試験”，LinuxWORLD 11月号，IDG Japan (Online:http://www.idg.co.jp/lw/serial/index.html)