LSB を活用した
効率のよいアプリケーション開発

2008 年 12 月 9 日

The Linux Foundation

LSB にようこそ

Linux Developer Network (LDN)は、Linux プラットフォーム向けのアプリケーション開発に従事している技術者や独立ソフトベンダー（Independent Software Vendor = ISV)、および、この分野への新規参入を考えている方々のためのオンライン・コミュニティです。アプリケーション開発の核となるツール群が Linux Standard Base (LSB)にまとめられています。

本書は、LSB を活用してアプリケーションのポータビリティを上げる手順について説明しています。全体は、大きく 4 つの部分で構成されています。

- ポータビリティとは何か、アプリケーション開発者にとって、それが何を意味するかを学習する。
- 対象アプリケーションをポータビリティの観点から調査し、ポータブルにするには多大な作業が必要か、小修正だけで済むかを見極める。本書は、アプリケーションのポータビリティを調べるためのツールやガイドラインを紹介しています。
- アプリケーションをポータブルにするための方策を講ずる。いかにそれを実行するかを詳述します。
- LSB を基礎にして、アプリケーションをポータブルにすることができた後、さらにあなたのアプリケーションのユーザー基盤を拡大するため、世界の Linux ユーザーにあなたのアプリケーションを紹介する方法を説明します。

ポータビリティについて知る

Linux 向けのアプリケーションの開発者の増加に伴い、ポータビリティやアプリケーション検証に関する質問が増えています。Linux でポータビリティの高いアプリケーションを作成する方法は？自分のアプリケーションは LSB 認証を受けるべきか？自分のアプリケーションに最適な方法は？

本章では、より良い Linux アプリケーション作成のプロセスを説明して、これらの疑問にお答えします。

「LSB 概論」のセクションでは、LSB とは何か、アプリケーションが LSB 認証を受けることの意味等を説明します。認証は当面見合わせたい場合でも、複数の Linux ディストリビューション上で動作するポータブルなアプリケーションの作成方法を理解するのに役立ちます。

「ポータブルなアプリケーションの開発方法」のセクションでは、アプリケーションをディストリビューション間共通で動作可能にする方法の概略を説明します。LDN 提供の最新ツールを利用すれば、ポータビリティは容易に実現可能です。LSB 認証取得の手続きも以前より大幅に容易になっています。

LSB 概論

一般に、オペレーティングシステムの成功度は、そのオペレーティングシステム上で動作するアプリケーションの数と質に密接に関係します。ところが、Linux 用にアプリケーションを開発する場合、異なるディストリビューションが存在することに起因する Linux 特有の問題に対応する必要が生じます。例えば、ディストリビューションごとにライブラリのバージョンが違ったり、重要ファイルの保存先が異なったりすると、アプリケーション開発者は、それらのディストリビューションの Linux ユーザーを照準に入れようとする場合、それらディストリビューション固有の開発をしなくてはなりません。Linux プラットフォームを対象とする開発は、それらを正しく認識することが必要です。

Linux を採用したいのに、そうすると事態がややこしくなる ― これはアプリケーション開発者にとっていくぶん皮肉な状況です。しかし、複数の Linux ディストリビューション向けのアプリケーション開発に要するコストとリソースを安易に考えることもできません。

この問題の解決、すなわち Linux プラットフォームをサポートするのに必要なコストの削減を目的に策定されたのが LSB です。Linux ディストリビューション間の差異を縮小することにより、アプリケーションの移植コストを大幅に低減し、また、商品化後のユーザーサポートの省力化・コスト節減も可能にします。<inter-distro compatibility>

また、LSB による標準化は、各ディストリビューションの版数アップ時の互換性維持にも役立ちます。Linux カーネルは、常に活発な機能追加が行われていますが、それにもかかわらず、LSB3.0 対応のディストリビューション以降、アプリケーションインターフェースの後方互換性が維持されています。これは、アプリケーション開発者にとっても、また、Linux ユーザーにとっても、いったん開発した、ないしは、導入したアプリケーションが、ディストリビューションの版数アップ時にも、一定のテストを経て継続利用できることを意味します。<intra-distro compatibility>

要約すれば、LSB による標準化は、Linux プラットフォームで開発に取り組む際の費用効率の向上、すなわち移植、サポート、テストに係わる負担の削減を可能とし、世界市場に向けてのアプリケーション開発を支援しています。

LSB のメリット

LSB が提示するソリューションは、ISV やアプリケーション開発者を楽にするだけでなく、Linux エコシステム全体に対しても非常に大きなプラス効果をもたらします。

さらに、Linux が UNIX オペレーティングシステムと同じ運命をたどることも防いでいるということもできます。かつて UNIX で起きたような、商業的利益のために、ひとつのオペレーティングシステムがいくつもの改良型に分裂し、それぞれの間に互換性が成立しないなどということは Linux の世界では起こりません。

ISV にとって重要な、具体的なメリットを以下に列挙します。

- ディストリビューション間の移植コストが低減でき、移植時の作業としてテストに専念できます<inter-distro compatibility>。

- 各ディストリビューションの版数アップ時の移植コストも低減でき、移植時の作業としてテストに専念できます<intra-distro compatibility>。

- Linux をサポートするための複雑さが減少し、サポート負荷の削減につながります。

- 特定のディストリビューションが優勢な地域別マーケットが存在する状況において、より多くに地域への参入が可能となり、ユーザー数の増大が見込めます。

- 開発・マーケティングに、Linux Foundation および LDN の支援を受けることができます。

LSB ソリューションが、Linux 環境の開発者の大きな力になることは明白です。では、LSB ソリューションとは正確に言うとどんなもので、どのように開発者の役に立つのか？答えは LSB を構成する各コンポーネントを理解することで得られます。

LSB を理解する

LSB は、アプリケーションとプラットフォーム間の相互運用性を定めた Linux オペレーティングシステムの中核標準です。この標準には、文書化されたバイナリインターフェース仕様、標準に従って開発されたディストリビューションおよびアプリケーションをテストするためのツール一式、およびテスト目的のサンプル実装が含まれています。

ディストリビューションベンダーは LSB 標準の主役です。彼等の参加なくして、LSB は意味がなかったと言えます。また、標準策定に協力した後も LSB を支持し続け、主要ディストリビューションは全て LSB 認証も取得しています。

LSB 推進の一環として、Linux Foundation は、ディストリビューションベンダーやサーバ

ーベンダーの協力を得て、LSB ワークグループを運営し、ベンダー間の合意形成と多様な要求間のバランスを図りつつ、有益な標準をタイムリーに提供することを目指して活動しています。

Linux 開発コミュニティのサポートも重要な要素です。オープンソースコミュニティは、複数のソフト開発プロジェクトを融合させ、単一のコンピューティングソリューションにまとめ上げることをなしとげました。ここで大切なのは、プロジェクトのメンテナー全員が、LSB を始めとするコンピュータ関連の現行標準をきちんと認識しているということです。LSB の拡充には、ディストリビューションベンダー、コミュニティ開発者、アプリケーション開発者が一丸となり協力し合っています。

次に LSB の技術要素の説明をします。

LSB 標準

まず、LSB の重要な基盤は、標準的なアプリケーション構築法を Linux アプリケーション開発者向けに解説したバイナリインターフェース仕様書で、下記が記載されています。

- 共通パッケージとインストールガイドライン
- 共通共有ライブラリとその選択
- コンフィグレーションファイル
- ファイル配置
- システムコマンド
- アプリケーションレベルおよびプラットフォームレベルのシステムインターフェース用 ABI（Application Binary Interfaces）

アプリケーションと実行環境間のバイナリインターフェースは、その多くが既存の標準に基づいて[1]記述されています。LSB が利用している主な標準には、Single UNIX Specifications (SUS)、Standard C++ ABI、System V ABI があります。その他、PAM(Pluggable Authentication Module)、X11、freedesktop.org が提供するデスクトップ標準等も参考にしています。

LSB は、個々のライブラリのインターフェースの装備、および、、各インターフェースに関連したデータ構造・定数を規定しています。ここに含まれるのは、開発者が必要とする共有ラ

1 IEEE POSIX(Portable Operating System Interface for UNIX) 1003.1 と LSB の差異、および、相互のポータビビティについては、POSIX and Linux Application Compatibility Design Rules (by C. Douglass Locke, Ph.D., 2006)に詳細な説明があります。

イブラリ（C++を含む）、ファイルシステム階層（ファイルの格納場所を定義）、各インターフェースの動作仕様、アプリケーションパッケージの詳細、アプリケーションの動作（インストール前およびインストール後）等です。

LSB は、LSB3.0 以降、ソースコードのレベル、および、バイナリのレベルの両方で、後方互換性維持の方針を原則としています。言葉を換えると、あるアプリケーションが、LSB の版数 x.ｙ（ただし、x.y>=3.0）を照準として開発すると、そのアプリケーションは、LSBx.y あるいはそれよりも新しい版数の LSB 版数で認証された、あるいは、準拠したディストリビューションでも動作することを意味しています。例えば、LSB3.0 を照準としたアプリケーションは、LSB3.0 で認証された、あるいは、準拠したディストリビューションのみならず、LSB3.1、LSB3.2、LSB4.0 のディストリビューションでも動作するのです。

このような後方互換性を達成するため、LSB 標準においては、後続の LSB 版は仕様の追加のみ、言葉を換えると、インターフェースは追加されるのみで、急に削除されることはないということです。LSB のポリシーに、インターフェース削除のメカニズムを有していますが、本当に削除されるのは、そのインターフェースに「削除予定(deprecated)」と表示される状態を少なくとも LSB の版数で３版を経た後、あるいは、およそ６年を経た後としています。このポリシーのおかげで、アプリケーションベンダーは、アプリケーションを LSB の特定の版数で認証を受けると、それ以後の版数で再認証する必要はなくなりました。もちろん、新しい版数に含まれる Linux 機能を積極的に利用するアプリケーションは、新たな LSB に対して認証を受けるのは当然です。

このアプリケーション互換性のポリシーは LSB3.0 で提供されたことにはご注意ください。LSB1.x、2.x においては、バイナリ互換性はメジャー版数内に留まっていました。

LSB は、ISO/IEC 23360 として、国際標準化機構（ISO）と国際電気標準会議（IEC）で正式に認められた国際基準です。従って、LSB は常に最上級の標準として維持されるだけでなく、その内容は常にオープンな性格のものです。

アプリケーション開発者にとって必要なあるインターフェースが LSB に収録されていないとしたら、その理由は様々です。例えば、ライブラリの作者またはメンテナーが、そのライブラリが未だ開発途上で、その動作が不安定であったり、今後の機能向上により、時とともにそのインターフェースが変化する可能性が高いと考えているとか、インターフェースの文書化が完了していない、あるいは、LSB ワークグループの LSB への定義追加作業が遅れている等です。

LSB は以下のアーキテクチャーに対応しています。

- Intel IA32
- Intel IA64
- x86-64/EM64T
- IBM PPC 32
- IBM PPC 64
- IBM 31-bit S/390
- IBM 64-bit zSeries

LSB ディストリビューションの認証

Linux Foundation は、LSB 標準に準拠しているディストリビューションについて認証を提供しています。認証された製品のみが LSB 認証のトレードマークを使用することが許可されます。この認証マークは、開発者とエンドユーザに、「LSB 認証」されたアプリケーションが LSB 認証されたディストリビューション上で正常に動作することを示します。図1に示すように、ディストリビューション固有のインタフェースを使用するアプリケーション C は、他のディストリビューション上で動作できませんが、認証アプリケーション D は、全ての認証ディストリビューションで動作します。Linux Foundation は、LSB 認証されたディストリビューション製品の一覧表を公開しています。

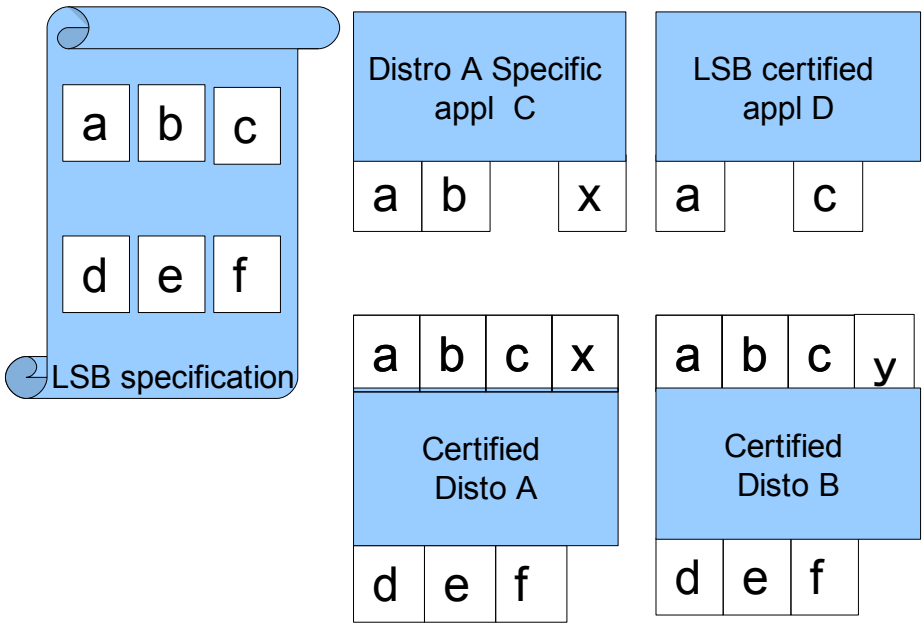

図１．認証ディストリビューションと認証アプリケーション

ディストリビューションがLSBに準拠している事を確認するために、ディストリビュータは、LSB Distribution Testkitを使ってディストリビューションのテストを実施し、テストに合格したディストリビューションのみがLinux Foundationによって認証されます。この際、LSB標準の個々のコンポーネントがどのようにテストされるかについては、LSB Database Navigatorによってアクセスされる LSB Databaseにて公開されています。

LSBアプリケーションの認証

これらのLSB仕様に合わせて開発されたアプリケーションは、LSB認証に近い位置にあると言えます。認証取得には、この段階でテストを開始し、どの程度LSBに準拠しているかを正確に把握することをお勧めします。テストは各ディストリビューションのネイティブビルドでも行えますが、コンパイラやリンカーとしてLSBソフト開発キット（SDK）を使用した方が不要なエラーを無くすことができるので、開発・テストの早い段階からLSB SDKの使用することを推奨します。

LSB認証に際しては、Linux Application Checkerを適用したアプリケーション単独のテストの他に、LSBサンプル実装および認証済みディストリビューション上で実際にアプリケーションを走らせてのテストも必要です。Linux Application Checkerが、どのようにアプリケーションのテストしているかについては、「Linux Application Checkerの技術的特長」にて説明しています。

正式認証を受けるためにLinux Application Checkerのテスト結果を提出する用意ができた時点で、LSBの認証システムに製品を登録してください。製品の登録が完了してから、テスト結果をアップロードしてください。その後、LSB商標ライセンス契約（Trademark License Agreement = TMLA）を結び、所定の申請費用をお支払い頂きます。テスト結果の審査は、TMLA締結と費用の支払い確認後に行い、合格の場合は正式にLSB認証されます。

LSBを利用する

LSBと、それがLinuxおよびLinuxアプリケーション開発にもたらす多大なメリットについては既にご理解頂けたと思いますので、LSBの適切な利用法に話を進めます。LDNの主要任務のひとつは、この方法を周知させることにあります。

アプリケーションのLSB認証を求めるのか、今回は見送るのか、目標を定めてください。LSBは非常に重要な標準ですが、誰もが認証取得を目指す標準と言うわけではありません。例えば、小規模のISVや個人の開発者の場合、認証に費やす時間が捻出できないことがあるかも知れません。アプリケーションによっては（特にカーネルを修正している場合には）、

現時点で準拠させるのが難しい場合もありえます。

しかし、そのようなアプリケーションでも、LSB から得るものが無いという訳ではありません。ちなみに、ISV が抱える複雑な問題の多くが、LSB のガイドラインに従うことで解消可能です。LSB ワークグループが作成・維持している LSB ソフト開発キット（SDK）には、ビルド環境に加え、LSB SDK に適合する形でソフトを効率良く移植するためのツールも含まれています。

LDN から入手可能なリソースは以下の通りです。

Linux Application Checker
アプリケーションのポータビリティレベルのチェック、ポータビリティ向上に向けたアドバイスが得られます。

LSB Database Navigator
ポータブルにすることを阻害する要因を識別し、その対策または代案を立てるのに利用。C または C++を使って Linux 上でプログラミングしようと考えているプログラマーにとっては、LSB Database Navigator は情報の宝庫とも言えます。

LSB ビルドツール
LSB SDK を用いて、開発者は、バイナリと RPM パッケージが LSB に準拠しているかどうかを検証し、また、ビルド実行中にアプリケーションによる API の使用状況をモニターして LSB 適合性を保証することができます。

LSB サンプル実装（LSB SI)
LSB に準拠した最小限の実行環境で、テスト目的に使用。ディストリビューション固有の動作の影響を受けていないことを保証するために、LSB 準拠のアプリケーションは必ず LSB SI 内でテストする必要があります。LSB 認証プログラムでも LSB SI の使用が義務づけられています。

チュートリアルおよびブログ
ポータビリティ向上、LSB 適合性検証、一般的な Linux アプリケーション開発等に関し、ハウツーを含む最新情報を掲載。

フォーラムおよびメーリングリスト
開発に関する問題の解決法、その他ヒントや助言をリアルタイムで入手可能。

LSB 認証およびマーケティングの支援
LSB 認証、ポータビリティ向上のいずれを選択した場合でも、開発したアプリケーションを Linux エコシステムで効果的に世界に露出する方法を決めるための情報を提供。なお、認証されたアプリケーションは、認証製品リストとして掲載されます。

LSB がアプリケーション開発者に提供するのは、世界に広がる Linux の優位性を生かすためのツール群と標準化基盤です。LDN は、LSB のポテンシャルを最大限に活用して、アプリケーションのポータビリティ向上を目指すアプリケーションベンダーや開発技術者を支援することを目的としています。

ポータブルなアプリケーションの開発方法

ディストリビューション間のポータビリティを達成するのが面倒だからと、Linux アプリケーションの開発や既存アプリケーションの改良をためらってはいませんか？LSB には、ポータビリティ実現に役立つツールや開発技法が多く存在しています。

図２.は LSB を活用したポータブルアプリケーションの開発手順です。青色で表したものが LSB の用意するツールです。

LSB ツールの一つ目は、LSB に完全準拠したアプリケーション作成を可能とする開発環境、LSB ビルドツール(LSB SDK)です。普通 Linux ディストリビューションが用意した開発環境には、そのディストリビューション固有のインターフェースが混在しますが、LSB ビルドツールを使用して開発を行えば、使用できるインターフェースは、LSB 準拠ディストリビューション全てに認められているものだけになります。この方法により開発したアプリケーションは、全ての LSB 準拠ディストリビューション上で動作できるようになります。

二つ目は、Linux Application Checker です。本ツールを利用してアプリケーションをテストすれば、ポータビリティ改善のための技術的なテクニックやヒントを LDN から入手できます。ポータビリティの観点から使用を推奨するインターフェースの情報も Linux Application Checker に盛り込まれています。

なお、Linux Application Checker は、特定のディストリビューション上で開発された既存のアプリケーションパッケージに対して適用することもできますので、Linux 上の既存アプリケーションがあれば、先ずは本ツールを適用してポータビリティの状況を確認できます。その上で、アプリケーションが使用する非 LSB ライブラリの数を減らす方策を採れば、特定の少数アプリケーションだけを戦略的にポータブルにすることができます。

Linux Application Checker のテストで非 LSB インターフェースの使用を排除できたら、三つ目のツール、LSB サンプル実装(LSB SI)上でそのアプリケーションの動作テストを行います。LSB SI は、LSB 準拠インターフェースだけが実装されていますので、そのアプリケーションが全ての LSB 準拠ディストリビューションで動作することの有効なテストとなります。

LSB SI 上での動作が確認できれば、そのアプリケーションを LSB アプリケーション認証に提出することができます。

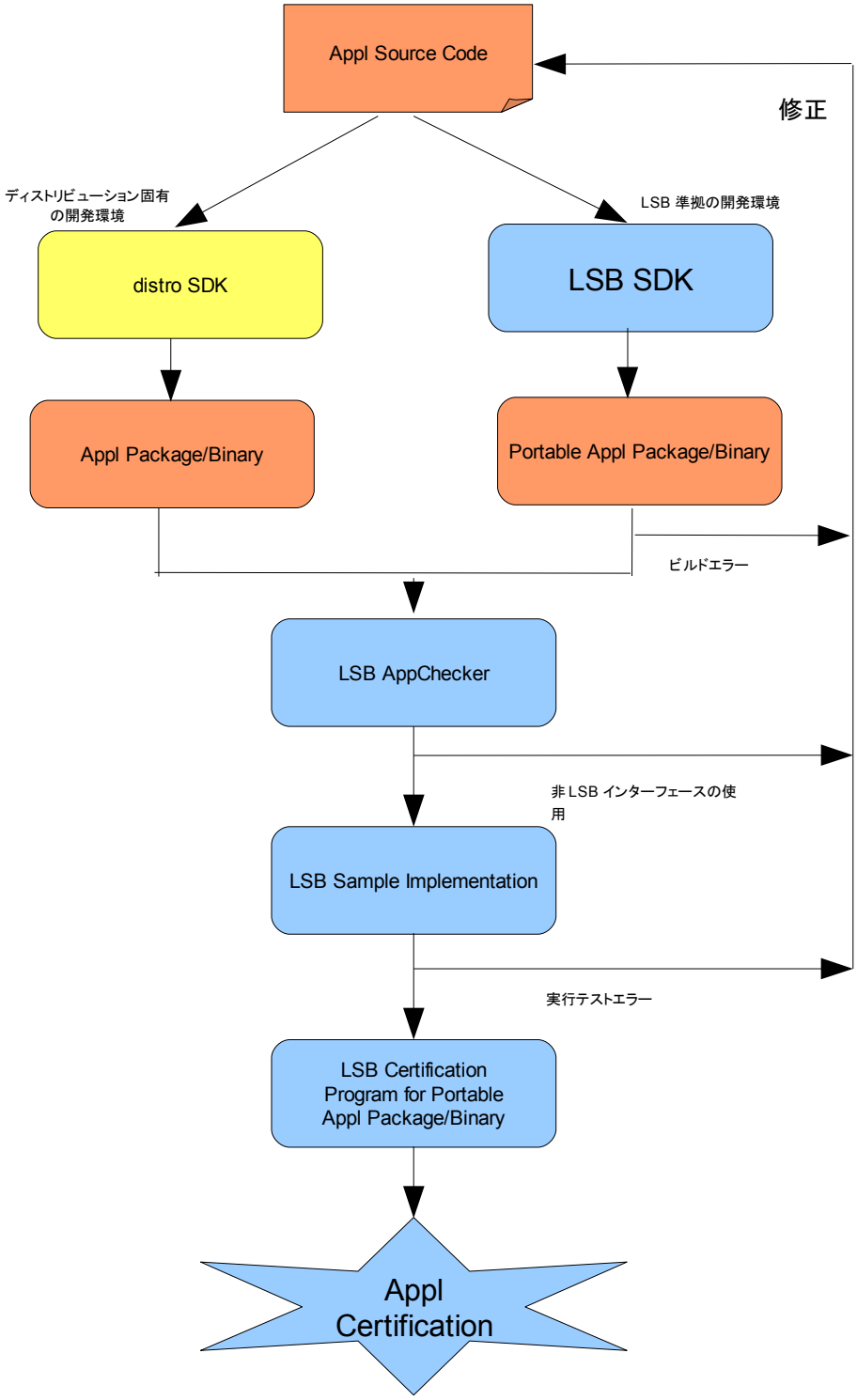

図２．　ポータブルなアプリケーションの開発手順

アプリケーションのポータビィティを調査

Linux 上のアプリケーションがあれば、最初にやるべきことは、アプリケーションの現在の状態 ― 特にそのポータビリティレベル ― を知ることです。

この用途のために、LDN は Linux Application Checker という素晴らしいツールを用意しています。このツールを使用すれば、アプリケーションのポータビリティの現状だけでなく、LSB 標準にどの程度近づいたかまで確認することができます。

「Linux Application Checker のすべて」のセクションで、この新しいツールの主な特長を説明します。このツールは、Linux Foundation が、ロシア科学アカデミー（Russian Academy of Sciences）と協力して開発したもので、LSB ワークグループの活動を通して Linux 用ソフト開発者を支援することを目指しています。

「Linux Application Checker 使用方法」のセクションでは、このツールのインストールの方法、このツールを使ってのポータビリティのチェック実施、そして、アプリケーションソースコードへのポータビリティの実現について解説します。

Linux Application Checker のすべて

Linux Application Checker（または AppChecker）は、Linux 向けソフト開発支援のために設計された強力な最新ツールです。本ツールの広範なテスト機能は、LSB ワークグループの支援と相まって、ポータビリティ実現を効果的にサポートします。

形式的な説明はさておき、Linux 用アプリケーションを書こうとしている開発技術者にとって、このツールの機能は具体的にどんな意味を持つのか、もっと直截的に言えば、ポータビリティ実現をどれほど容易にするのか、という実際的な疑問にお答えします。

従来は、LSB モデルに可能な限り厳密に従って開発することが、Linux アプリケーションをポータブルにする最良の方法でした。この方法は現在も有効ですが、問題は、アプリケーションベンダーは、通常、特定のディストリビューションのみを想定して開発を進めことです。

そこで、アプリケーションの LSB 認証の意図の有無に関わらず、便利なツールとして AppChecker を開発しました。本ツールは、LSB の範囲を越えてアプリケーションが使用しているシンボルやライブラリを分析し、対策をガンダンスします。

AppChecker の使いやすさは、このツールに最初に出会った瞬間から顕著に感じられるはずです。ターゲットとするアーキテクチャーに対応したバージョンをダウンロードしたら、tarball ファイルを格納フォルダ上で解凍・保存するだけで実行可能になります。用意された Web サーバには、あらゆるアーキテクチャーに対応した広範なインターフェースに対応し備わっています。

テストしたいアプリケーションのバイナリに AppChecker をポイントすると、アプリケーションが使用しているインターフェースとライブラリのシンボルテーブルが、LSB に準拠したディストリビューション（総数 30）の既知情報と自動的に比較されます。比較完了後には、そのアプリケーションはどのディストリビューション上で動作させるべきかの詳細なレポートが表示されます。つまり、多少の追加テストや修正を行えば、そのアプリケーションが実行可能になる全ディストリビューションの情報を得ることができます。

このマッチングテストで肯定的な結果が出たからといって、そのアプリケーションがあらゆるディストリビューション上でそのまま使用できるという保証にはなりません。しかし、ポータビリティ向上に向けて進むべき道は示されますし、ひいては移植費用の削減も可能になります。そして、開発者は、リスクや負担の少ないやり方で、より大きなマーケットを対

象に、より多くの選択肢をユーザーに提供できるようになります。

AppChecker が本領を発揮するのはミスマッチが検出された場合です。特定のディストリビューションに問題が存在するかだけでなく、ディストリビューションに欠けているインターフェース、互換性のないインターフェースを特定し、それらのインターフェースからのリンクとして、ポータビリティ改善のための的確な対応策（インターフェースの修正または置換の方法）の検討まで可能にします。

もちろん、AppChecker 本来の機能として、LSB 準拠までの到達度も確認できますので、認証取得を目的にしている場合には必須のツールです。

Linux Application Checker の技術的特長

主な機能は以下の通りです。

- アプリケーションの各 Linux ディストリビューションへのポータビリティをチェックし、そのディストリビューション上でどの程度動作するかを解りやすく表示します。
- ポータビリティ改善方法に関するガイダンスを行います。
- LSB 認証にどこまで近づいているかを解りやすく表示します。

Appchecker は以下の 5 ツのテストツールで構成されています。

- *bin/lsbappchk* - LSB tests for ELF binaries.
- *bin/lsbpkgchk* - LSB tests for RPM packages.
- *bin/lsbappchk.pl* - LSB tests for Perl scripts.
- *bin/lsbappchk.py'* - LSB tests for Python scripts.
- *bin/lsbappchk-sh.pl* - LSB tests for Shell scripts.

AppChecker（lsbappchk)は、C/C++シンボル（主としてグローバル変数を含む関数）と、アプリケーションが必要とし、ディストリビューションが提供するダイナミックリンクライブラリに存在するライブラリを解析することによって動作します。殆どの実行ファイルはデバッグ機能を含まないので、アプリケーションが期待するタイプ情報がディストリビューションのライブラリが供給するオブジェクトのタイプと合致するかどうか確認するのは不可能です。

例えば、データ構造の配置が変わっている可能性もありますし、関数パラメータが 32 ビット整数から 64 ビット整数に変化している場合もあります。関数の動作詳細がバージョン間で変化している場合も、AppChecker では検出できません。

ディストリビューションの LSB 認証は、これらの問題を防止します。各ライブラリインターフェースのタイプ情報と動作を細かく規定して文書化しているだけでなく、LSB インターフェースの動作確認テストも行うからです。このテストは、LSB 認証ディストリビューションが LSB で規定されている全てのインターフェースを装備していること、これらのインターフェースの背後にある関数が正しいタイプシグネチャを持ち、正しく動作していることを検証するために行います。

今までの説明で Linux Application Checker に関する知識は十分と思いますので、「使用方法」のページに移り、実際に使ってみましょう。

Linux Application Checker 使用方法

Linux Application Checker (AppChecker)でできることは既に学習しましたので、ここでは実際に使ってみましょう。インストールから実行、得られた結果の解釈まで、一般的な手順をご紹介します。AppChecker のインストール、Web インターフェースの使い方、コマンドラインインターフェースの使い方、テスト結果リポジトリの管理、および、AppChecker 終了・削除について説明していきます。

AppChecker のインストール

Linux Application Checker のダウンロードページから、Linux Application Checker tarball（tar.gz）パッケージをダウンロードしてください。

AppChecker は、ノンインストール Perl スクリプトとして動作するので、非 root ユーザーとしてこれを実行することができます。以下にその手順を示します。

1. コマンドラインに下記を入力してパッケージを解凍します。
   tar xzf Linux-app-checker-local-<version>.<architecture>.tar.gz
2. 下記スクリプトを実行して AppChecker を開始します。
   ./app-checker/bin/app-checker-start.pl [port-number]

AppChecker は、カレントユーザー権限で実行できます。
上記スクリプトは、Linux Application Checker に組み込まれている Web サーバを起動し、このサーバへの正しいアドレスを持つユーザーのブラウザを探して立ち上げます。

注：ポート番号の引数はオプションです。（デフォルトでは 8889）

上記スクリプトを実行してもブラウザが見つからない場合は、マニュアルでブラウザを開き、アドレスに http://localhost:8889/を指定してください。（ポート番号は前記注に述べた通りです。）

下記 URL を指定すると、任意のリモートコンピュータから Linux Application Checker に接続できます。

http://<test-machine-name>:8889/

Web インターフェースの使い方

Linux Application Checker 開始後、Web ベースのアプリケーションのメインページは、デ

フォルト Web ブラウザに表示されます。最初のテストを行う手順は以下の通りです。

1. Application Check リンクをクリックして、Application Check ページを開きます。

2. Name 欄にテストレポートの名前をタイプします。

3. チェックしたいアプリケーションへのファイルパスを Components 欄に入力します。
注：Components の欄には、アプリケーションの全コンポーネント名を入力してください。

- 個々のファイル
- 全ディレクトリ
- インストールした RPM パッケージ（pkg:で始まるもの）
- RPM およびで deb パッケージファイル
- TAR.GZ および TAR.BZ2 アーカイブ
  アーカイブは自動的に解凍されます。

  テストされるのは、以下のタイプのファイルです。
  ELF (実行ファイルと共有オブジェクト)
  Perl スクリプト
  Python スクリプト
  シェルスクリプト

４．コンポーネントを簡単に入力したい場合は Select Application Components ボタンをクリックして、Web 上の Application Components Selection ダイアログを開けます。

５. ディレクトリ名をクリックして、アプリケーションコンポーネントがある場所へと移動します。

６. 希望するコンポーネントのチェックボックスをクリックして選択します。

７. Finish をクリックしてダイアログを閉じます。Components 欄に、選択したコンポーネントへのファイルパスが表示されます。

８．Additional Options アイコンをクリックすると追加オプションが表示されます。

9. LSB Version を選択します。

10. LSB Profile を選択します。

11.このテストを再度行う予定がある場合は、テストプロファイルの名前を User Profile Management 欄に打ち込んで、Save をクリックしてください。

12. テストを開始するには、Run the Test ボタンをクリックします。テストは自動的に進行し、終了すると結果が Test Report ページに表示されます。
注：テストを実行するにはいくつかの必要条件があります。（<Linux-app-checker dir>/README ファイルを参照。）これらの条件が満足されていない状態でテストを実行すると、ページの上部にエラーメッセージが表示されます。この場合、問題解決後に再度テストを実行してください。

13. Test Report ページにある各種タブをクリックすると、アプリケーションのポータビリティに関する下記項目が確認できます。

- Distribution Compatibility：アプリケーションと、Linux Foundation による分析が完了しているディストリビューションとの間の互換状態。
- Required Libraries：アプリケーションが必要とする外部ライブラリのリスト（DT_NEEDED ELF セクションに依拠）。
- Required Interfaces：アプリケーションが必要とする外部インターフェースのリスト（ELF シンボルに依拠）。
- LSB Certification：互換性に関して問題がない場合は、テストしたアプリケーションの認証手続きを開始できます。Apply for Certification リンクをクリックすると、認証システムにリンクされます。

14. テスト終了時には、Results History リンクをクリックしてください。行われたテストごとの詳しいレポートが表示されます。

15. 特定のレポートを参照したい場合は、Data/Time 欄の対応するリンクをクリックしてください。レポートの一部を削除したい場合は、対応するチェックボックスで選択して Remove Selected Entries ボタンをクリックします。

コマンドラインインターフェースの使い方

Application Checker によるテストは、コマンドラインを使っても実行できます。最初のテストを開始する場合は以下の手順に従ってください。

1. ./linux-app-checker/utils に移動します。.

2. 下記コマンドを実行します。
./run_tests.pl --paths=<paths-list>
ただし、<paths-list>は、テストするファイルやディレクトリをコロンで区切ったリスト（Web インターフェースの Files and Directories テキストボックスに相当）とします。

3. インストールされているパッケージをテストするには、下記コマンドを使用します。
./run_tests.pl --packages=<packages-list>
ただし、<packages-list>は、コロンで区切ったパッケージ名のリストを示すものとします。

下記オプションも有効にお使い頂けます。

-n <name>：アプリケーション名

--lsb=<LSB-version>：テストに使用する LSB のバージョンを指定。

-T <profile>：テストに使用する LSB プロファイルを指定。使用可能な値は、core,c++ または core,c++,desktop（デフォルト値）のいずれかです。

-r <results-dir>：テスト結果を保存するディレクトリを指定。

現行バージョンで利用可能な全オプションを表示するには、
./run_tests.pl –help
を実行してください。

テスト結果リポジトリの管理

実行したテストの結果が保存されるディレクトリは./linux-app-checker/results です。テスト結果はそれぞれサブディレクトリ（<arch>-<machine>-<application>-<date>-<time>）に個別に格納されます。

Web インターフェースの Results History ページで見られるようにしたいテスト結果は、.../results/HISTORY ファイルに記載してください。

テスト実行に Web インターフェースを利用すると、HISTORY ファイルは自動的に管理されます。他のマシンやディレクトリから手動でテスト結果をコピーした場合は、正しいディ

レクトリ名をこのファイルに手動で書き込む必要があります。

AppChecker の終了・削除

Web サーバを終了するには、Web インターフェースの Administration ページに進んで、Stop Server ボタンをクリックします。下記スクリプトを実行しても終了できます。
<linux-app-checker dir>/bin/app-checker-stop.pl [port-number]

ポート番号はオプションですが、ひとつのマシン上で複数の Linux Application Checker サーバが起動している場合には設定が必要です。（指定を怠ると、ポート番号を尋ねるメッセージが現れます。）

Linux Application Checker を削除するには、それが解凍されているディレクトリを削除してください。

警告：このディレクトリには、Linux Application Checker だけでなく、全てのテスト結果も格納されています。

AppChecker をダウンロードするには、Linux Application Checker ページに移動し、そこから AppChecker をダウンロードしてください。

LSB Database Navigator

LSB Database Navigator は、LSB Database の Web フロントです。アプリケーション開発者はデータベースを検索して、アプリケーション開発に活用することができます。

LSB Database には LSB 標準とそれをとりまく Linux エコシステムに関する技術情報を集積しています。LSB Database Navigator は、Web インタフェースによってこれらの情報を表示します。Linux 開発者、Linux ディストリビュータ、あるいは、LSB ワークグループ参加者に有効であり、いろいろな情報を取り出したり、分析したり、情報を追加したりできます。LSB Navigator は、Linux Developer Network(LDN)の一部です。以下に Navigator の提示するデータを説明します。

（1）LSB エレメント

LSB 標準を構成する各エレメントをリストしています。LSB アプリケーションバイナリーインターフェース(ABI)を構成するライブラリ、ヘッダー、C++クラスなど、ELF の構成要素、RPM の構成要素、コマンド、スクリプト言語でなっています。

（2）ディストリビューションとアプリケーション

これまでに認証されたディストリビューション、および、これまでに Application Checker にてテストされたアプリケーションのポータビリティの詳細が保存されています。

（3）LSB ワークグループサービス

アプリケーションが使用しているインタフェースの調査状況やディストリビューションの LSB 認証におけるテストのカバーレジ情報が示されます。

アプリケーションのポータビリティを高くする

アプリケーションをポータブルにする方法は、特定のディストリビューション上で完成した既存アプリケーションのポータビリティをチェックすることから開始することもできれば、そのアプリケーションを LSB ビルトツールにて完成させることを目指した開発を行うこともできます。

「新しいアプリケーションの作成」のセクションでは、LSB ビルトツールを使い、LSB 準拠アプリケーションの開発を進めるためのツールと情報を提供します。

> 「LSB 準拠のアプリケーションの作成」では、LSB に準拠した新しいアプリケーション作成方法を説明します。
>
> 「LSB サンプル実装 (LSB SI)」は、アプリケーションの標準適合性テストに使用します。

「既存アプリケーションの修正」のセクションは、既に完成しているアプリケーションをお持ちのベンダーのために用意されています。

> 「複数ディストリビューションへの移植」の項では、ディストリビューションを選択して移植する方法について記述しています。

「LSB ツールー実際に使ってみようー」のセクションでは、一連の LSB ツールを使ったアップリケーションのチェック、テストの一例を示します。

新しいアプリケーションの作成

適切なツールを使って作成・テストすれば、新しいアプリケーションを LSB 認証可能にすることは、それほど難しいことではありません。

LSB Software Development Kit (SDK)は、バイナリと RPM パッケージの LSB 準拠性を開発者自身で検証する手段を提供します。ビルド進行中は、アプリケーションによる API の使用状況をモニターして、検証の確実性を保証します。

「LSB 準拠のアプリケーション作成」の項では、LSB 準拠のアプリケーション作成に必要なツールの入手方法およびインストールおよび実行方法の概略を述べ、実際に開発に着手できるようガイドします。

「LSB サンプル実装(LSB SI)」は、LSB に準拠した最小限の実行環境で、アプリケーションのテストに使用します。ディストリビューション固有の動作の影響を受けないように、LSB 準拠を目指すアプリケーションは必ず LSB SI でテストする必要があります。LSB 認証プログラムでも、この LSB SI の使用が義務づけられています。

LSB 準拠のアプリケーション作成

LSB 準拠の新しいアプリケーションを作ることを決定したら、次にどうするか？LSB に向けての開発は、多少のセットアップ作業を伴います。具体的には以下の通りです。

- LSB Software Development Kit (SDK)のダウンロード
- LSB SDK のインストール
- Linux Application Checker のダウンロードとインストール
- LSB SI のダウンロードとインストール

開発がスタートしてからは、アプリケーションを LSB に準拠させるための良い戦略があります。それには Linux Application Checker ツールを利用しますが、詳しくは後程このページで説明します。

LSB SDK のダウンロード

LSB SDK は、LSB 認証にふさわしいアプリケーションを作成するための開発環境を提供します。このキットは tarball 形式で、インストールスクリプトも入っています。LDN の Web サイトのダウンロードページからダウンロードできますので、該当するアーキテクチャーに見合ったものをダウンロードして保存してください。

LSB SDK のインストール

自分のアーキテクチャーに見合った LSB SDK パッケージ（tar.gz）のダウンロードが完了したら、下記の手順でインストールしてください。

1. パッケージを解凍します。

tar xzf lsb-sdk-<version>.<architectue>.tar.gz
例：tar xzf lsb-sdk-3.2.0-4.ia64.tar.gz

2. tarball 解凍時に作成されたサブディレクトリ（lsb-sdk）に移動します。

3. インストールスクリプト（./install.sh）を実行します。
注：root 権限が必要です。

インストール終了後、PATH 環境変数に/opt/lsb/bin を追加しておくことを推奨します。アプリケーションをコンパイルしてリンクする際、lsbcc を入力するだけで済みます。

もうひとつ変更できる変数はMANPATHです。この変数をアップデートしておくことにより、LSBに関連したmanページにアクセスできます。MANPATHに/opt/lsb/manを追加しておいてください。

ia64 LSB SDKをインストールした場合に得られる実際の出力は次の通りです。

```
cal@bilbo:~/lsb-sdk> ./install.sh

This system appears to be a RPM-based distribution, such as those from
Red Hat, SuSE/Novell, Mandriva, Asianux, etc.
Is this correct? yes
In order to install these packages, you need administrator privileges.
You are currently running this script as an unprivileged user.
You have sudo available. Should I use it? yes
Using the command "sudo /bin/sh -c" to gain root access. Please type
the appropriate password if prompted.

Installing packages...
root's password:
There may have been problems with the package installation. Check
error-log.txt for more information

cal@bilbo:~/lsb-sdk>more error-log.txt
warning: lsb-build-base-3.2.1-1.ia64.rpm: Header V3 DSA signature:
NOKEY, key ID a0530ad1

cal@bilbo:~/lsb-sdk>
```

Linux Application Checkerのダウンロードとインストール

Linux Application Checkerをダウンロード/インストールする方法は、「Linux Application Checker使用方法」のページを参照してください。

LSBサンプル実装(LSB SI)のダウンロードとインストール

LSBサンプル実装(LSB SI)の詳細は、「LSBサンプル実装(LSB SI)」のページを参照してください。

LSB 準拠に向けての戦略

LSB ビルドツールおよびテストツールが揃ったら、実際のアプリケーション開発を進めます。ビルドプロセスが終了したら、Linux Application Checker でテストします。このツールを用いれば、LSB 準拠に向かう道程の、どの辺の位置に今アプリケーションがあるかを知ることができます。テストレポートの LSB Certification ページを開けると、アプリケーションが使用しているライブラリとインターフェースの、どれが LSB 準拠を阻んでいるかまで特定することができます。そして、それらが確認できたら、次に取るべき方法は４つあります。なお、これらの方法は、特定のディストリビューションに対してのポータビリティを実現するもので、LSB 認証を可能とするものではないことにご注意ください。

LSB 準拠阻害要因の排除方法

- 代替インターフェースを使用する。

（例：memalign()の代わりに posix_memalign()を使用）

- 代替ライブラリを使用する。

（例：openssl の代わりに libnss を使用）

注：これは、LSB 準拠を目的とする場合、および、ディストリビューション間共通のポータビリティを実現する場合のいずれにもふさわしい方法です。openssl は、いろいろなディストリビューションが異なったバージョンに対応した ABI 群を提供しているからです。

- ライブラリをスタティックにリンクする。

警告：一部のライセンスソフトウェア、例えば GPL 等は、スタティックにリンクされたライブラリをメインアプリケーション本体の一部とみなし、メインアプリケーションにまでライセンス条件を拡大適用する場合があるので注意が必要です。

- ISV が提供する共有ライブラリにダイナミックにリンクする。

それぞれの方法はそれぞれのベストプラクティスが含まれていますが、これらは近い将来、LDN においてさらに拡張していきます。

LSB サンプル実装 (LSB SI)

LSB SI は、汎用ディストリビューションではありません。これは、LSB 仕様に準拠したアプリケーション構築の可能性を実証するテスト環境なのです。

殆どの Linux ディストリビューションは多数のパッケージで構成され、そのどれをインストールするか、または後で追加・削除・アップグレードするかはユーザーに任されています。これは至極普通のことですが、ポータブルアプリケーションを作成する場合には少々問題が起こります。アプリケーションをインストールする予定のシステム上に存在することを許されていない何かをアプリケーションがリンクしてしまっていないことの確証が得にくいからです。

アプリケーション開発に LSB を利用すれば、この問題の発生を効果的に抑えることができます。LSB 認証されたディストリビューションは全て、一定のライブラリとインターフェースで構成され、LSB に文書化されている通りに動作するからです。

LSB SI は、LSB ディストリビューションの中核となるこれらのライブラリとインターフェースの必要最小部分だけを実装したもので、プログラムの出荷前テストに使用されます。

例えば、自動検出ビルドスクリプトを有するソフトウェア（GNU configure 等）では、システムにインストールされているもの全てが、それが LSB の一部であるかどうかとは無関係に、ビルドで利用する可能性があります。そこで、得られたバイナリを LSB SI で実行すれば、LSB 不適合のフィーチャーが紛れ込んでいるケースを検出できることになります。

AppChecker はスタティックツールなので、他のプログラムを呼ぶ場合のような、実行時の問題を見つけることはできません。一方、LSB SI では、LSB に適合した環境でアプリケーションを動作させることが可能になります。

LSB SI は、LSB に準拠したシステム構築が可能であることの「概念証明(proof-of-concept)」と考えることもできます。その基本方針は、パッケージメンテナーによって公開されているパッケージと最小限のパッチだけを使用することにあります。これら一部のパッチは、LSB SI 環境でビルドを行う時に（大抵の場合は、ビルドに必要なの機能を補うために）必要となります。バグを修正したり、LSB 適合性を満たすために用意されているパッチもありますが、これらは本来、上流メンテナーが次のリリースに盛り込むべき性質のものです。

LSB SI は、LSB 仕様の進化に合わせて変化し続けます。新しいフィーチャーは、それらが仕様として確定された時点で、LSB SI のテストバージョンに組み込まれます。新しい仕様が承認されるのとほぼ同時に、その仕様に沿った LSB SI のバージョンが発表になるということは、いずれかのディストリビューションのフルサポートを決める前に、新しいフィーチャーをプレビューする機会を開発者に与えるということでもあります。

LSB SI は、スタンドアロンの Linux ディストリビューションとして設計されてはいません。あくまでサンプルであり、単なるテスト環境です。通常のディストリビューターから得られる付加価値 － 性能やセキュリティに関する問題の追跡と解決等 － は、LSB プロジェクトでは提供していません。

インストール

LSB サンプル実装（LSB SI）ツールは、ふたつのパッケージで構成されています。

lsbsi-chroot：SI chroot 用コアファイルシステムを提供
lsbsi-tools：SI chroot 設定・実行用のツールを提供

これらのパッケージは、rpm、deb、tarball のいずれかのフォーマットで入手可能で、互いに独立してインストールできます。実際のところ、それぞれを異なったマシンにインストールし、一緒に動作させることもできます。

rpm/deb 形式でのインストール

ダウンロードしたパッケージファイルのタイプにより、rpm、dpkg その他のパッケージマネージャーを使用する必要があります。例えば：

```
rpm -i lsbsi-chroot-3.2.0-3.i586.rpm lsbsi-tools-3.2.0-3.i586.rpm
```

デフォルトでは、lsbsi-chroot は/opt/lsb/si/chroot に、lsbsi-tools は/opt/lsb/si/tools にインストールされます。lsbsi-chroot インストール後に、lsbsi グループがシステムに追加されます。

lsbsi グループに自分のユーザー名を追加するのを忘れないでください。LSB SI ツールは、lsbsi グループのユーザーまたは root ユーザーしか使用できません。また、lsbsi グループの全てのユーザーが、chroot や mount を実行できるようにするために/etc/sudoers が修正されます。

xdg-utils がシステムにインストールされている場合は、lsbsi-tools パッケージをインストール後に、LSB SI を実行するためのメニューがデスクトップに表示されます。 その他の場合は、/opt/lsb/si/tools/si を使用して実行してください。

tarball 形式でのインストール

1. lsbsi-chroot-3.2.0-3.i586.tar.gz ファイルの展開：

sudo tar xzfv lsbsi-chroot-3.2.0-3.i586.tar.gz

2. lsbsi-chroot-3.2.0 フォルダに移動してコマンドを実行：

sudo ./install.sh.

このスクリプトで、lsbsi グループが追加され、/etc/sudoers ファイルが修正されます。

3. lsbsi-tools-3.2.0-3.i586.tar.gz を展開：

tar xzfv lsbsi-tools-3.2.0-3.i586.tar.gz

4. lsbsi-tools-3.2.0 フォルダに移動してコマンドを実行：

sudo ./install.sh.

インストール中に、chroot へのパス（lsbsi-chroot が展開されているフォルダへのパス）を指定してください。

LSB SI ツールの実行

LSB SI ツールは、デスクトップメニューから選択、または[lsbsi-tools-path]/si-GUI を実行して動作させます。[lsbsi-tools-path]は、rpm または deb を利用した場合は/opt/lsb/si/tools で、tarball 形式でインストールした場合は lsbsi-tools が展開されているフォルダへのパスとなります。これにより、設定調整・chroot 環境起動用の GUI インターフェースが立ち上がります。chroot 環境を起動するには、[lsbsi-tools-path]/si スクリプトを使用する方法もあり、これは、Qt4 ライブラリが搭載されていないシステムの場合に便利に利用できます。

設定オプション

chroot 環境起動時のオプション設定にはグラフィカルインターフェースを利用します。これらオプションの殆どは、chroot 環境へリモートアクセスするための ssh 接続のためだけに使用されます。

設定可能なオプションは以下の通りです。

- Path：ローカルまたはリモートシステム上の chroot 環境へのパス
- Host：lsbsi-chroot のあるサーバのホスト名
- Port：サーバ上の ssh デーモンへのポート番号
- User：ssh 接続されるユーザー名
- PreferredAuthentications：認証モード（publickey、keyboard-interactive、password、または hostable)
- X11forwading：X11 転送オプション（normal または trusted）

chroot 環境を立ち上げるためのオプション調整を行う一番簡単な方法はグラフィカルインターフェースを利用することですが、si_run.conf ファイルに全てのオプションを指定するという方法もあります。si スクリプトをご使用の場合は、si スクリプトの中に引数を指定することもできます。（詳しくは、./si--help を参照のこと。）

オプションの設定が全て終了したら、Run SI ボタンをクリックしてください。SI chroot 環境でシェルが動作しているターミナルウインドウ（gnome 端末または konsole）が開きます。

LSB SI トラブルシューティング

現行バージョンの LSB SI は SE Linux をサポートしていません。LSB SI を立ち上げる前に SE Linux を無効にすることを推奨しています。

LSB SI をデスクトップメニューから起動できない場合は、ターミナル（[lsbsi-tools-path]/si）から起動して、何が問題なのか、エラーメッセージで確認してください。

LSB SI をデスクトップメニューから起動することが、不正な chroot 環境の原因になることがあります。この問題は Fedora Core 9 の場合に特に起こりやすく、理由は、/etc/sudoers が Defaults requiretty の文字列を含んでおり、それが、ターミナル以外からの sudo コマンド実行の邪魔をしているからです。この状況は、下記に述べるような対策を取ることで回避できます。

- /etc/sudoers 内の Defaults requiretty の行をコメントアウトする。
- 該当するメニュー項目に関し、Launcher Properties 内のアプリケーションのタイプを変更する。タイプを Application in Terminal に変更する。（Gnome の場合）

- ターミナル経由で LSB SI を起動する。rpm や deb 形式でインストールした場合は /opt/lsb/si/tools/si を使用。または lsbsi-tools.tar.gz ファイルを展開したフォルダの./si を実行。

LSB SI のアンインストール

rpm または deb の場合は、rpm ¯e または dpkg ¯r コマンドを使用します。tarball 版をアンインストールする場合は、該当するフォルダに移動し、sudo ./uninstall.sh を実行します。

既存アプリケーションの修正

既に開発されたLinuxアプリケーションを持っている開発者やISVは、先ず何をすれば良いのか？他のLinuxディストリビューションに移植するか、それとも、いろいろなディストリビューション間共通で使用できるようにするか？LSB認証を取得するか、それとも見合わせるか？

ここでは、このような選択のいかんに拘わらず、アプリケーションの移植効率を大幅に向上させるための方法とツールをご紹介します。

「複数ディストリビューションへの移植」の項では、複数のディストリビューションへの移植作業を正しく行うためのガイドラインを提供します。

複数ディストリビューションへの移植

複数ディストリビューションへのアプリケーションの移植は、多大なエンジニアリングコストとサポートコストというイメージにつながりやすく、このためにLinuxプラットフォームの真の価値が損なわれるのは残念なことです。LDNは、ディストリビューション間共通のポータビリティ向上の技術やLSB認証を確実にするための技術に重点を置きながら、開発コストと工数の大幅削減を可能にするスキルとツールを提供して行くことを第一目標としています。

ポータビリティを向上させる最も良い方法は、LSB Application Checkerを利用してテストしてみることです。このツールを用いれば、LSB準拠まで、あとどれくらいで到達できるかを知ることができます。テストレポートのLSB認証ページを開けると、アプリケーションが使用しているライブラリとインターフェースの中で、どれがLSB準拠を阻んでいるかまで特定することができます。そして、それらが確認できたら、LSB準拠を実現するために推奨できる方法は4つあります。例えLSB準拠が目的ではない場合でも、LSBに向かって歩むことは、アプリケーションのポータビリティを格段に高める結果につながります。

LSB準拠阻害要因の排除方法

- 代替インターフェースを使用する。

（例：memalign()の代わりにposix_memalign()を使用）

- 代替ライブラリを使用する。

（例：openssl の代わりに libnss を使用）

注：これは、ディストリビューション間共通のポータビリティ実現を、はっきりと目標に定めている場合に最適な方法です。openssl は、いろいろなディストリビューションが異なったバージョンに対応したABI群を提供しているからです。

- ライブラリをスタティックにリンクする。

警告：一部のライセンスソフトウェア、例えばGPL等は、スタティックにリンクされたライブラリをメインアプリケーション本体の一部とみなし、メインアプリケーションにまでライセンス条件を拡大適用する場合があるので注意が必要です。

- ISVが提供する共有ライブラリにダイナミックにリンクする。

これらの他にも、アプリケーション移植時に考慮すべき戦略もあります。。

- エンディアンネスの検討（bigendian または littleendian）

アーキテクチャーレベルで考慮すべき問題です。

- 32 ビット/64 ビット対応性

どちらか一方にだけ依存しないでください。

- ハードコードされたアドレスの排除

- オブジェクトサイズを取得する場合の sizeof の使用

32 ビットから 64 ビットに変換する場合は特に重要です。

- char が符号付き（または符号無し）と仮定しない。

- 浮動小数が double と同じサイズと仮定しない。

- #ifdef を使用する場合は注意する。

- 関数プロトタイプを必ず作成し、コンパイル時には-Wstrict-prototypes オプションを用いて、その使用をコンパイラに厳密にチェックさせる。

- POSIX インターフェースとデータ型を使用する。

- C および C++ライブラリに含まれる標準関数のみを使用する。

- ディストリビューション固有のライブラリの使用や、ライブラリの拡張はしない。

- アプリケーションのビルドには LSB SDK が提供する LSB ツールを使用する。

- ファイルシステムの階層構造は FHS に準拠させる。

LSB ツールー実際に使ってみようー

Linux はオープンなオペレーティングシステムなので、目的に合わせた設定と構築が可能です。しかしながら、種類が豊富で選択の余地が広いということがユーザーにとって有利である反面、このような異種混交状態にソフト開発者がてこずっているというのも事実です。基本は似ていても、それぞれ微妙に違うプラットフォームに向けてパッケージを設計しなくてはならない面倒があるからです。ただし、アプリケーションが Linux Standard Base (LSB)に準拠しており、各 Linux ディストリビューションも同様であれば、アプリケーションは各準拠ディストリビューション上で実行可能になります。この項では、LSB の良さを理解した上で、この標準に適合する形で実際にコードを移植する方法を学習して行きましょう。

本チュートリアルは、Linux ソフト開発者を読者に想定して書かれています。LSB とは、Linux ソフトウェアディストリビューション間の互換性を向上させるための仕様、および、その目的のために開発されたツールとテスト用パッケージソフトで構成されています。LSB 仕様に準拠し、ユーザーに互換性を保証していると認められたアプリケーションやディストリビューションは LSB 認証を取得することができます。以下に LSB の概略と、アプリケーションコードの LSB 準拠性を検証する方法を解説します。

前提条件

本チュートリアルを利用するには、C または C++プログラミング言語、標準的な Linux ソフトウェア開発環境、およびそこで使用されるコンパイラ、リンカ、システムライブラリ、各種設定、ビルドユーティリティ、パッケージ化ソフト等の一連のツールに関して、知識と経験を持っていることが必要です。

さらに、コマンドラインを使用してのソフトウェアのインストール、また、ファイルシステムの構築、ネットワークサービスの停止、システムサービスの追加等を始めとする Linux システムの保守・管理に関しても、ある程度の経験が必要です。

Debian Linux を利用している場合は、APT パッケージマネージャーに関する経験も必要となります。

本チュートリアルを開始するに当たっては、ソフトウェアパッケージをいくつか Linux にインストールしなくてはなりません。インストールが必要なものは以下の通りです。

Linux ディストリビューション用 KVM パッケージ

注：KVM パッケージの代わりに、VMWare Workstation を Microsoft® Windows® XP にインストールして、バーチャル化した Linux のインスタンスを XP プラットフォームで動作させることもできます。VMWare Workstation のお試し版は無料でダウンロードできますので、それを使ってバーチャルマシン（VM）を作成し、セーブすれば、VMWare Player（無料）に切り替えて再実行が可能です。

LSB サンプル実装(LSB SI)

LSB ビルド環境

LSB アプリケーションテストパッケージ

対象ハードウェアプラットフォームの LSB 仕様（バージョン 3.2.0）も同時にダウンロードして、目を通しておくことをお薦めします。代表的な 7 個のプロセッサ（IA32、IA64、PPC32、PPC64、S390、S390X および AMD64）に対応したハードウェア別 LSB 仕様書が用意されています。

アプリケーションの認証を受ける場合は、Linux Foundation の LSB 認証プログラムのページを訪れてください。

本チュートリアルに記載されているサンプルプログラムを実行するには、オペレーティングシステムに Linux または Windows XP を採用しているコンピュータが必要です。VMare Workstation は、これらいずれのプラットフォーム上でも動作し、以降の作業に必要な Linux 環境を提供します。

アプリケーションが、自分のプロセッサに対応した LSB 仕様書で承認されていないソフトウェアライブラリに依存している場合、それらのライブラリもバーチャルシステムにインストールする必要があります。混乱を防ぐため、アプリケーション固有のライブラリは標準ライブラリと分離してインストールするのが理想的です。

標準化の必要性

DistroWatch の Web サイトの記載によると、現在少なくとも 25 の Linux ディストリビューションが存在しています。これだけでも驚異的な数字ですが、サイズを最適化したもの（例：Damn Small Linux）、使い勝手を改良したもの（例：Knoppix）、コミュニティユーザー主体で開発したもの（例：Fedora Core）、特定の国用にカスタマイズされたもの（例：Red Flag Linux）等を加えると、この数字は一足飛びに 2 倍にも 3 倍にも増加しま

す。これだけ多くのバリエーションがある近代的なオペレーティングシステムは、Linux 以外には見当たりません。

Linux の「ご自分でどうぞ」的文化は、多様化と専門化を助長します。他方、このような拡散の仕方は、Linux アプリケーションを開発・販売・サポートしている ISV にとって、対抗しがたい状況を創出しています。1980 年代の「UNIX® 戦争」が ISV の成功のチャンスを打ち砕いたとするなら、これだけ多くの Linux バージョンの存在も、Linux アプリケーションの量産市場を閉塞させるだけとしか言いようがありません。複雑なアプリケーションを、多数の Linux バリエーションのために開発・サポートするために費やさなければならないコストを想像してみてください。

LSB 用語
LSB について論じる時、適合（conformance）、準拠（compliance）、認証（certification）という言葉が頻繁に使われます。

LSB には、ランタイムとアプリケーションを LSB 仕様に「適合」させるための必要条件が定義されています。

該当する全ての必要条件が開発者によって満たされ、仕様に適合しているアプリケーションおよびディストリビューションは、LSB に「準拠」しています。

LSB 準拠性が正式な審査で証明された場合、アプリケーションおよびディストリビューションは「認証」を受けます。

本チュートリアルを、LSB に準拠したアプリケーション作成方法を習得するのに利用してください。テストを繰り返す度に認証取得の可能性が確実に高まりますが、自分で自分のアプリケーションを認証することはできません。

相互運用性向上のために

Linux Standard Base (LSB)は、多様化の促進と一貫性の維持を目指して、Free Standards Group（Linux Foundation の前身）によって策定されました。LSB の Web サイトでは、「Linux ディストリビューション間の互換性を高め、全ての LSB 準拠システム上でソフトウェアアプリケーションの実行を可能にするための標準制定と推進を目的とするオープンソースプロジェクト」と定義されています。

LSB では一貫性のある Linux プラットフォームを規定しており、このプラットフォームを（少なくとも）実装している Linux ディストリビューションは、LSB 準拠と認められます。（このことは、付加的な特徴を組み込むことを否定するものではありません。）LSB プラットフォームが提供するサービスだけを使用する Linux アプリケーションも、同様に LSB に準拠しているとみなされます。アプリケーションが追加で必要とするライブラリは、インストールまたはスタティックリンクのいずれかの方法で利用可能にします。（これらのライブラリも、自給自足的で LSB に準拠している必要があります。）

LSB が標準を定めている（または、いずれ定めようとしている）分野を図 3.に示します。具体的には、全ての Linux アプリケーションの基礎となるライブラリ、重要ファイルの格納先指定や現地語化への対応方法を含む実行環境、システムの初期化、LSB 準拠システムで使用する共通コマンドやユーティリティ、および、ユーザー/グループ管理が含まれます。

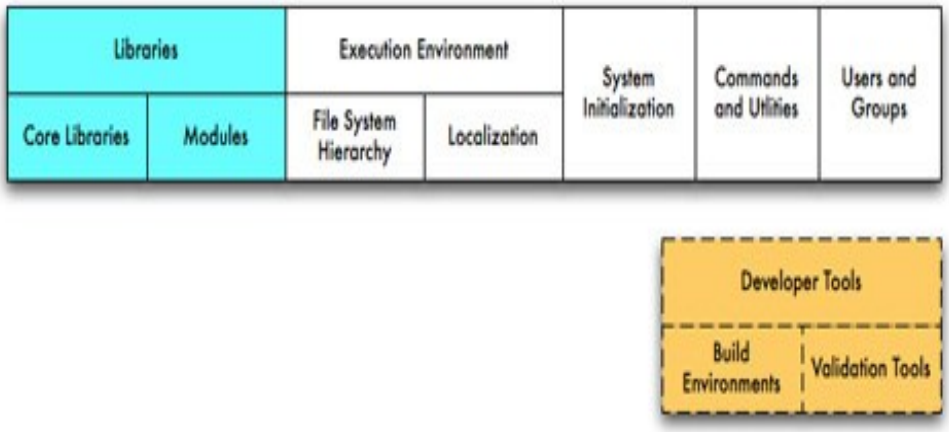

図３：LSB のコンポーネント

アプリケーション開発者にとって最大の関心事は、ブルーとオレンジでマークした部分かと思われます。

青色で示した部分には、LSB 準拠のアプリケーションが使用できるコアライブラリと追加モジュールを規定しています。コアライブラリには、libc、libm、libpthread、libpam、libcrypt、libz、libncurses、librt、libgcc_s が含まれます。モジュールには、X Window System Version 11 用グラフィックス（libX11、libXt、libXext、libSM、libICE、libGL）および C++標準ライブラリ（libstdc++）があります。

オレンジの部分は LSB 仕様の一部ではありませんが、アプリケーションやパッケージが、LSB に準拠しているかどうかを検証するための重要なツールを示しています。

バイナリ互換性

LSB 標準はバイナリ互換です。LSB 準拠のアプリケーションバイナリ（コンパイル後の実行コード）は、LSB 準拠の任意の Linux ディストリビューション上で実行可能です。（ただし、アプリケーション、ディストリビューションともに、同一のプロセッサ用に開発されたものである必要があります。）

バイナリ互換性はソース互換性とは大きく異なります。ソース互換のアプリケーションの場合、ローカルシステムのコンパイラとライブラリを使用してのリコンパイルが必要で、問題は、これらふたつが予測不能な動作やばらつきの源であるという点です。ソース互換の場合、実行ファイルが動作すること（ましてや期待通りに動作すること）が保証されている訳ではありません。

整合性確保のメリット

LSB は、ディストリビューションとアプリケーション間に締結された「拘束力のある協定」と解釈することができます。ディストリビューションは、仕様に従ったライブラリとインターフェースを提供することを約束し、アプリケーションは、これらのライブラリとインターフェースだけを使用し、足りないものは自分で用意することに合意します。両者がこの協定を遵守することを保証する手段として、各種ツール類、ガイドライン、ディストリビューションおよびアプリケーションに対する検証・認証サービスが LSB から提供されています。

LSB に従うことは、ディストリビューションベンダー、ISV、ユーザーの三者全ての利益につながります。

ディストリビューションベンダー

LSB に準拠することにより、ユーザーの保有する製品上で、多くのアプリケーションが終始一貫、確実に動作することを保証できます。いったん認証を取得してしまえば、その後は差別化や新しいフィーチャーの開発に全力を傾注できます。

ISV

認証されたバイナリは、LSB 準拠のディストリビューションに次々展開して行くことが可能です。認証取得後に必要なことは、そのバイナリを自分でテストしサポートして行くことだけです。もし、そのバイナリが国際的に認められれば、たったひとつのアプリケーションの単一バージョンだけで、世界中のユーザーを満足させることも可能になります。

ユーザー

競合するいくつもLSB準拠のディストリビューションの中から、自分に最も適したLinuxを選ぶこともできますし、ひとつのベンダーだけに偏ることも避けられます。Linuxアプリケーションの選択の幅も広がります。

このようなダイナミックスは、ディストリビューションベンダー、ISV、ユーザーの全てが、それぞれのLinuxに対する投資から大きな恩恵を受けられることを意味します。もちろん、Linux自身も報われます。

LSB構築環境とサンプル実装

LSBは、ディストリビューションとアプリケーションが互いにLSB仕様に従うことを約束した「協定」です。しかし、Linuxディストリビューションを LSBに準拠させることがいかに複雑かを知らずして、アプリケーションを準拠させるのは至難の技かも知れません。

アプリケーションの開発者は、自分が使っているコードのことは当然熟知していますし、それをLSBに準拠させるために積極的に働いてもくれるでしょう。しかし、そのコードがLSBに準拠していないコンポーネントに依存しているかどうかまでは明確に解らない場合も多いのではないでしょうか？特に、Linuxディストリビューションの場合、その中には何千個というコンポーネントがゴロゴロしているのですから。

例えば、そのコードが、特定のディストリビューションだけが採用しているLSB未準拠のストリングライブラリに依存しているかも知れません。また、そのストリングライブラリが、つい最近、システムアドミニストレーターによってローカルに追加されたばかりという場合もあり得ます。別の例では、ライブラリのバージョンの問題があります。あるライブラリのバージョンWがLSB仕様に準拠しているからといって、バージョンYまでそうであるという保証はありません。

マイグレーション用ツール

このような依存性を検出・排除するために、つまり、コードをLSBに準拠させるための重要な手段として、LSBはふたつのツール ― ビルド環境とサンプル実装（LSB SI） ― を提供しています。

ビルド環境は、ソースコードがLSB未準拠のライブラリとアプリケーションバイナリインターフェースに依存していないかどうか検査するためのツールで、ヘッダファイル、スタブライブラリ、それとLSB準拠のLinuxシステム上でのソフトウェアビルド環境をシミュレーションするためのコンパイララッパで構成されています。このツールは、あらゆるLinux

システム（LSB に準拠していないシステムも含む）にインストール可能です。

ビルド環境には 2 種類あります。ひとつは、開発者が現在使用中のローカルオペレーティングシステムで動作するユーティリティのセットで、LSB 環境におけるビルドのシミュレーションに使用します。もうひとつは完全なファイル階層となっており、chroot 環境の代わりに使用します。前者は、LSB 準拠性を手軽に評価する際に便利です。後者では、現在のビルド環境を新しい root に取り込む必要がありますが、迷子ファイルの問題は起こりません。

サンプル実装(LSB SI)は、LSB 準拠の実行環境でアプリケーションが正しく動作することを検証する手段です。LSB 準拠のディストリビューションを代用する、一種のミニ Linux と考えて頂くと良いでしょう。実際にサンプル実装からブートすることもできます。

もちろん、Linux に準拠しているディストリビューションそのものを使って検証することも可能ですが、サンプル実装の使用を推奨する理由がいくつかあります。まず、LSB で定義されているものだけで構成されています。また LSB 仕様のバージョンに合わせて、異なるバージョンのサンプル実装が入手可能です。例えば、アプリケーションが LSB3.0 と LSB3.2 に準拠していることを検証したい場合には、該当するバージョンのサンプル実装を同じテストシステムにインストールするだけで済みます。これは、Linux システムをふたつ用意するよりは余程簡単です。さらに、新たに承認された LSB 仕様に対する検証も、その仕様に準拠したシステムが導入されるまで待つ必要もなく、即座に実行に移せるというメリットもあります。

LSB 標準の各バージョンに対応して、ビルド環境とサンプル実装はセットで提供されています。本チュートリアルは、LSB3.2.0 に準拠したプラットフォームの使用を前提として書かれています。（このチュートリアルを作成した時点での最新バージョンです。現在普及している Linux のディストリビューションの多くは、旧バージョン（LSB3.1.0、LSB3.0.0）の LSB のインスタンスを用いて認証されている可能性が高いと思われます。）

バーチャルマシン（VM）を用いての移植のメリット

LSB3.2.0 サンプル実装を利用するには、これを chroot 環境で実行できるオペレーティングシステムが必要です。本チュートリアルでは、コミュニティ開発の Linux ディストリビューションのひとつ、Fedora 9 (FC9)を、この目的に使用します。

FC9 を持っていない場合、また、旧 LSB の多数のインスタンス、もしくは複数の Linux ディストリビューション上でこのサンプル実装を実行したい場合はというと、最も簡単でコストのかからない方法は、それぞれの Linux ディストリビューションを VM 上で走らせるこ

とです。（方法としては多少煩雑になりますが、マシンをマルチブートに変更したり、ハードウェアを増設したりすることでも対応できます。）

KVM は、Ubuntu Hardy を始めとする殆どのディストリビューションがサポートしているバーチャル化技術です。Intel-VT や AMD-V (AMD SVM エクステンションとも呼ばれる) をサポートしている比較的新しい x86 系 CPU が搭載されたシステムで利用できます。KVM をサポートしている（または、していない）CPU のリストは Xensource の Web サイトに掲載されています。システムによっては、バーチャル化のエクステンションを有効にするために、BIOS コンフィギュレーションの変更が必要となります。BIOS コンフィギュレーションを変更した場合は、変更をフラッシュメモリに保存後、電源を入れ直す必要がもあります。

KVM をサポートしていない Linux システム、または、Windows XP などのシステムをお使いの場合は、代わりに VMWare をインストールします。この場合、Thought Police の Web サイトにある Fedora Core 9 イメージが必要となります。

VMWare Workstation のダウンロードとインストール

VMWare Workstation を Debian Linux にインストールし、FC9 を VM で実行します。これで、LSB3.2.0 のビルド環境とサンプル実装が実行可能になります。

VM イメージのダウンロード

FC9 VM イメージは Linux Foundation の Web サイトからダウンロードできます。このイメージはオペレーティングシステム全体を含み、従って 1400MB と、かなり大容量です。

KVM のインストール

Ubuntu Hardy をご使用の場合、KVM パッケージは簡単にインストールできます。ターミナルウインドウで下記コマンドを使用します。

```
sudo apt-get install kvm
```

グラフィカルインターフェースを利用する場合は、Synaptics パッケージマネージャーを使用します。

Debian または Ubuntu ディストリビューションでは、root ユーザー以外が KVM を利用する場合には、自分のユーザー ID を"kvm"グループに登録しなくてはなりません。テキストエディタで/etc/group を編集してスーパーユーザーになるのがひとつの方法です。Ubuntu の場合は、メニュー選択という方法もあります。システムのメニューバーから、System-

>Administration->Users->Groups と選び、Unlock ボタンをクリック。次にパスワードを入力して Manage Groups をクリックすると Group Settings ウインドウが開きますので 、"kvm"を探し、Properties をクリックします。Group Members リストが表示されたら、自分のユーザー ID にチェックマークをつけて OK ボタンをクリックします。"kvm"グループに変更を加えた後は、必ず一度ログアウトして、再度ログインしてください。

KVM Workstation の実行

KVM バーチャルマシンを起動するには、ターミナルウインドウで下記コマンドを実行します。

kvm fedora-9-i386.qcow2

Ctrl+Alt を押さない限り、KVM アプリケーションはマウス/キーボードからの入力を全てトラップします。バーチャル環境に戻るには、KVM ウインドウ上をクリックしてください。

おめでとうございます！以上でめでたく FC9 が VM 上で動作することになります。

LSB へのコードの移植

セットアップ完了したら、LSB へのコード移植プロセスに進みます。

一般的な移植方法

LSB へのアプリケーション移植は次の手順で行います。

コードを新しいビルドシステムにコピーします。
新しいビルドシステムとは、別個のハードウェア（本チュートリアルでは VM）上で動作している、LSB 準拠の Linux ディストリビューションのことです。

コードをビルド後、Linux Application Checker (AppChecker)ツールを用いてバイナリをスキャンし、LSB 仕様にないシンボルが含まれていないことを確認します。スタティックアーカイブもスキャンして、それらが LSB 準拠のアプリケーションでの使用にふさわしいかどうかも確認します。

無効なシンボルが発見された場合は、コードまたはコードアセンブリを LSB 仕様に適合するよう変更します。
例えば、LSB 仕様に合致しないライブラリはスタティックリンクで接続して、コードがライブラリ内で自己完結するようにします。（ライブラリ内のコード自体は LSB に準拠してい

ると仮定した場合。）全ての問題を解決したら、次のステップに進みます。

LSB ビルド環境を使用して、クリーンかつ標準に準拠した環境でコードをビルドします。コードが LSB 未対応のライブラリを使用している場合は、それらをインストールまたはスタティックリンクでつなぐ等の変更が必要になります。（全てのライブラリは LSB に準拠している必要があります。）

LSB ビルド環境によるコードのビルドに成功したら、LSB サンプル実装（LSB SI)でコードを実行します。

対象の Linux システムが LSB に準拠している場合は、そのシステム上で実行することも可能です。

アプリケーションをパッケージ化します。
LSB 準拠の RPM は、LSB 準拠のシステムに必ずインストールできます。ただし、この方法だけに拘る必要はなく、LSB に準拠したシステム上で動作可能という条件の下で、他のパッケージ化技術も利用できます。例えば、tarball を付属したシェルスクリプト、（LSB 準拠という条件で）お手持ちのインストーラなども利用可能です。

では、簡単なアプリケーションを実際に作ってみましょう。

LSB ビルド環境ユーティリティのインストールと実行
chroot バージョンの LSB ビルド環境を利用する前に、ビルド環境ユーティリティを試してみます。これらのユーティリティ AppChecker（lsbappchk、lsbpkgchk）は、殆どの Linux システムに簡単かつ迅速にインストールでき、アプリケーションや RPM パッケージ（標準 LSB フォーマット）が LSB 準拠にならない原因の確認に役立ちます。

LSB ビルド環境ユーティリティのダウンロードとインストール
ビルド環境ユーティリティはふたつの RPM（lsbappchk 用と lsbpkgchk 用）で構成されています。このテストシステムの例は Debian Linux を使用しているので、まず deb フォーマットに変換し、Debian の dpkg パッケージマネージャーが使えるようにします。

1 $ sudo apt-get install wget

2 $ wget http://ftp.linuxfoundation.org/pub/lsb/test_suites/released-3.2.0/binary...

3 $ wget http://ftp.linuxfoundation.org/pub/lsb/test_suites/released-3.2.0/binary...

4 $ wget http://ftp.linuxfoundation.org/pub/lsb/lsbdev/released-3.2.0/binary/ia32...

5 $ wget http://ftp.linuxfoundation.org/pub/lsb/lsbdev/released-3.2.0/binary/ia32...

6 $ wget http://ftp.linux-foundation.org/pub/lsb/lsbdev/released-3.2.0/binary/ia3...

7 $ sudo apt-get install alien
8 $ alien *.rpm
9 $ ls -t -1
lsb-pkgchk_3.2.2-2_i386.deb
lsb-build-cc_3.2.1-1_i386.deb
lsb-build-c++_3.2.1-1_i386.deb
lsb-build-base_3.2.2-1_i386.deb
lsb-appchk_3.2.4-1_i386.deb
lsb-pkgchk-3.2.2-2.i486.rpm
lsb-appchk-3.2.4-1.i486.rpm
lsb-build-c++-3.2.1-1.i486.rpm
lsb-build-cc-3.2.1-1.i486.rpm
lsb-build-base-3.2.2-1.i486.rpm
10 $ sudo dpkg --install *.deb
11 $ ls /opt/lsb
bin doc man
12 $ ls /opt/lsb/bin
lsbappchk lsbc++ lsbcc lsbpkgchk

コマンド 1 は wget のインストール（システムにまだ搭載されていない場合）、コマンド 2-6 は LSB ユーティリティの最新版のダウンロードを行います。
lsb-appchk は、与えられたバイナリが LSB で定義され、かつダイナミックにリンクされたシンボルだけを使用しているかどうか検証します。
lsb-pkgchk は、LSB 準拠のシステム上にソフトウェアをインストールするためのアプリケーションパッケージが有効かどうか検証します。lsb-pkgchk は RPM だけを対象としたツールです。ただし、LSB では、ソフトウェアのインストールに RPM だけを使用するよう限定している訳ではありません。
lsb-build-base は、スタブライブラリとヘッダファイルを提供します。スタブライブラリは、

LSB に定義された機能を実装してはいませんが、LSB システム上に存在するダイナミックライブラリと同様に機能し、LSB 準拠のアプリケーションのビルドを可能にします。
lsb-build-c++ は、ビルド環境に C++を追加します。

lsb-build-cc は lsbcc を内蔵しています。lsbcc は、LSB 準拠のアプリケーションを作成するためのコンパイラ＝GNU Compiler Collection (GCC)用のラッパです。アプリケーションが GNU タイプの configure スクリプトを使用している場合は、デフォルトの CC コンパイラ（GCC）の代わりに lsbcc を利用するよう簡単にスクリプトを変更することができます。場合によっては、makefile 等で GCC を直接 lsbcc に置き換えることも可能です。
コマンド 7 の機能は、alien（RPM を Debian deb パッケージに変換するユーティリティ）をインストールすることです。コマンド 8 は alien を実行し、コマンド 9 はその結果を表示します。コマンド 10 は、コマンド 11 と 12 が示すように、全てのソフトウェアを/opt/lsb/ディレクトリにインストールします。

RPM のビルド・検証に関する解説は、本チュートリアルでは省略します。代わりに lsbcc を使って小さな C アプリケーションを作成し、その結果として得られたバイナリに無効なシンボルが存在しないか、lsb-appchk を使ってスキャンしてみることにします。

サンプルプログラム
リスト 1 に、コマンドライン引数を出力する小規模なプログラムを示します。
（エラー処理は意図的に除外してあります。）

リスト 1：単純な C プログラム

```
#include <stdio.h>
#include <unistd.h>

main(argc, argv)
int argc;
char *argv[];
{
   int i = 0;

   for (i = 1; i < argc; i++) {
    fputs(argv[i], stdout);
```

```
    putchar(' ');
  }

  putchar('\n');

  exit(0);
}
```

ファイル名 echoargs.c のファイルにコードをコピー&ペーストしてから、Debian システムにインストールされているコンパイラを使用してビルドしてください。

```
$ cc -o echoargs echo.c
$ ./echoargs hello there, world!
hello there, world!
```

コードは意図した通りに機能しますが、LSB に適合しているかどうかは不明です。チェックするには lsbappchk コマンドを実行します。

```
$ /opt/lsb/bin/lsbappchk echoargs
Checking binary echoargs
Incorrect program interpreter: /lib/ld-linux.so.2
Header[ 1] PT_INTERP       Failed
Found wrong interpreter in .interp section: /lib/ld-linux.so.2 instead of: /lib/ld-lsb.so.3
```

echoargs が LSB に不適合なアプリケーションなのは明らかなので、LSB のコンパイラを用いて、アプリケーションを適合させます。
さらに、LSB コンパイラ（lsbcc）を使用してコードをビルドし直し、その結果得られたバイナリ上で lsbappchk を実行します。

```
$ /opt/lsb/bin/lsbcc -o lsb-echoargs echoargs.c
$ /opt/lsb/bin/lsbappchk lsb-echoargs
Checking binary lsb-echoargs
```

かなり良くなり、LSB にも適合しました。
このバイナリは、LSB スタブライブラリ（システムのヘッダファイルの代わりに LSB include ファイル（ヘッダファイル）を使用）を利用してビルドされています。では、実際

にアプリケーションは実行できるのでしょうか？

```
$ ./lsb-echoargs
-bash: no such file or directory: ./lsb-echoargs
```

残念ながら、このサンプルシステム（Debian Sarge）はLSB3.2に不適合で、バイナリの実行は不可能です。./lsb-echoargsコマンドが生成したメッセージ、-bash: ./lsb-echoargsもいささか奇妙です。そのようなファイルやディレクトリは存在せず、システムがバイナリをロードできなかったというエラーが発生していることを示します。（LSBサンプル実装を使って、このバイナリを実行する方法は後程解説します。）

別の例として、リスト2をご覧ください。このコードは、従来のC言語の正規表現に、オープンソースのPerl Compatible Regular Expressions (PCRE)を付加します。PCREは非常に便利ですが、LSB仕様には含まれていません。

リスト2：PCREアプリケーションの抜粋

```
#include <pcre.h>
..

int main()
{
   pcre *re;
   const char *error;
   int erroffset;

   ...

   re = pcre_compile("^[A-Z]", 0, &error, &erroffset, NULL);

   ...
}
```

LD_LIBRARY_PATHが/usr/local/libを含むと仮定した場合、コマンドcc -o pcre pcre.c -l pcreを使用してコードをビルドすることができます。lsbappchkを使用してコードチェックを行うと、エラーメッセージが生成されます。

```
$ cc -o pcre pcre.c -l pcre
$ /opt/lsb/bin/lsbappchk pcre
Incorrect program interpreter: /lib/ld-linux.so.2
Header[ 1] PT_INTERP       Failed
Found wrong interpreter in .interp section: /lib/ld-linux.so.2 instead of: /lib/ld-lsb.so.3
DT_NEEDED: libpcre.so.0 is used, but not part of the LSB
Symbol pcre_compile used, but not part of LSB_Core
```

PCRE によって宣言された関数は LSB に準拠していないため、警告が出されます。（PCRE ライブラリの残りの部分は LSB 準拠という前提です。）このエラーを避けるには、-static フラグを追加します。

lsbcc を使用すると、同じコマンドでもエラーが発生しません。

```
$ /opt/lsb/binlsbcc -o pcre pcre.c -l pcre
$ /opt/lsb/bin/lsbappchk pcre
Checking binary a.out
$ nm a.out | grep pcre
0809b680 R _pcre_OP_lengths
0809b900 R _pcre_default_tables
0809b6d4 R _pcre_utf8_table1
0809b6ec R _pcre_utf8_table1_size
0809b6f0 R _pcre_utf8_table2
0809b708 R _pcre_utf8_table3
0809b720 R _pcre_utf8_table4
0809b760 R _pcre_utt
0809b888 R _pcre_utt_size
080b385c B pcre_callout
0804b0a0 T pcre_compile
0804b0e0 T pcre_compile2
080b18e4 D pcre_free
080b18e0 D pcre_malloc
080b18ec D pcre_stack_free
080b18e8 D pcre_stack_malloc
```

PCRE コードは、lsbcc によって実行ファイルにスタティックにリンクされ、これは Linux プラットフォーム間のライブラリの違いを防ぐひとつのソリューションともなります。lsbcc は GCC コンパイラへのコマンドライン引数を修正して、LSB ヘッダファイルとライブラリの使用を可能にしているだけでなく、LSB 未準拠ライブラリへのダイナミックリンクを避けています。

GCC や自作またはよく使われるツールを用いてコードビルドを行う場合には、chroot LSB ビルド環境より lsbcc の方が適しているかも知れません。反対に、GCC 以外のコンパイラを使用する場合、または、特定のコンパイラ、コンパイラオプション、ライブラリ、include ファイルパス等への依存度が高い場合は、LSB ビルド環境の使用がお薦めです。次のセクションで、LSB ビルド環境のインストールや使い方について説明します。

LSB ビルド環境とサンプル実装（LSB SI)のインストール

注：これ以降は、VMWare Workstation 上で動作しているバーチャル FC9 インスタンスの使用を前提に説明します。FC9 VM がサスペンド状態の場合、または VMWare Workstation が終了している場合は、これらの動作を復旧または再開し、FC9 インスタンスをスタートさせる必要があります。インスタンスが動作している場合は、root ユーザーとしてログインしてください。

chroot LSB ビルド環境を使用すれば、十分にコントロールされた隔離環境でアプリケーションのビルドを行うことができます。chroot ビルド環境をダウンロードしてインストールしたら、ユーティリティ、ライブラリ、アプリケーションのビルドに必要なその他のソースとともに展開してください。次に、chroot コマンドを使用して、現在有効な root ファイルシステムをビルド環境に変更します。これで、ビルド環境の root にあるファイル以外は一切見えなくなります。

chroot ビルド環境のダウンロードとインストール

LSB のダウンロードページから、LSB3.2.0 のビルド環境をダウンロードしてください。chroot ビルド環境は、複数の RPM の形で配布されます。

root プロンプトが表示されたら、一時ディレクトリを作成してください。そのディレクトリに移動するには cd を使います。次いで、下記コマンドを実行します。（wget は、インストールした FC9 VM に入っています。バックスラッシュを使った複数行に渡る長い行が出てきます）

```
$ wget http://ftp.linuxfoundation.org/pub/lsb/lsbdev/released-3.2.0/binary/ia32...
```

$ wget http://ftp.linuxfoundation.org/pub/lsb/lsbdev/released-3.2.0/binary/ia32...
$ rpm -i lsb-build-base-3.2.2-1.i486.rpm
$ rpm -i lsb-build-cc-3.2.1-1.i486.rpm

RPM をインストールすると、新しいディレクトリ、/opt/lsb-buildenv-ia32/が作成されます。このディレクトリの内容は既におなじみのことと思います。

bin boot dev etc home lib media mnt
opt proc root sbin srv tmp usr var

このディレクトリを隔離された環境として使用するには、chroot /opt/lsb-buildenv-ia32 /bin/bash コマンドを実行します。新しく現れるシェルプロンプト、bash-3.00#は入力を促すためのものです。なお、ファイルシステム周辺を参照した時に実際に見えるのは、現在有効またはトップレベルのディレクトリである/opt/lsb-buildenv-ia32/に格納されているファイルだけです。

では、とりあえず Ctrl+D を押して chroot から抜け出し、FC9 VM に戻ることにしましょう。そして、LSB サンプル実装をインストールした後でコードをビルドし、ビルド環境とサンプル実装上でそれぞれ実行してみて、本チュートリアルを終えることにします。

Chroot LSB サンプル実装のダウンロードとインストール

サンプル実装（LSB SI)は、基本となる tarball と、数個のテストツールから成る（オプションの）拡張 tarball の形で配布されます。（ユーザーモード Linux バイナリフォーマットでもダウンロード可能です。）

インストール開始時に、FC9 内でサンプル実装の保存場所を決めてください。通常は、/opt フォルダを使用します。次に、一時ディレクトリを作成し、tarball をダウンロードして解凍します。

1 $ mkdir /tmp/tgz; cd /tmp/tgz

2 $ wget http://ftp.linuxfoundation.org/pub/lsb/impl/released-3.2.0/binary/ia32/l...

3 $ wget http://ftp.linuxfoundation.org/pub/lsb/impl/released-3.2.0/binary/ia32/l...

4 $ wget http://ftp.linuxfoundation.org/pub/lsb/impl/released-3.2.0/binary/ia32/l...

```
 5 $ wget http://ftp.linux-foundation.org/pub/lsb/impl/released-3.2.0/binary/ia32/...

 6 $ cd /opt
 7 $ tar xjvf /tmp/tgz/lsbsi-core-ia32-3.2.0.tar.bz2
 8 $ cd /opt/lsbsi-core-ia32
 9 $ tar xjvf /tmp/tgz/lsbsi-graphics-ia32-3.2.0.tar.bz2
10 $ tar xjvf /tmp/tgz/lsbsi-boot-ia32-3.2.0.tar.bz2
11 $ tar xjvf /tmp/tgz/lsbsi-test-ia32-3.2.0.tar.bz2
```

コマンド1は、中間ファイル用のディレクトリを作成します。コマンド2-5は、LSBサンプル実装の構成要素をそれぞれダウンロードするためのコマンドです。
lsbsi-bootはブート可能なシステムで、サンプル実装のスタンドアロンモードでの起動（必要な場合）を可能にするためのカーネルその他のソフトウェアを含んでいます。
lsbsi-coreは基本的なランタイムで、LSB仕様に記載されているフィーチャーだけで構成されています。lsbsi-coreは、chrootでバーチャルなLSB準拠の環境に移行する場合に使用されます。
lsbsi-graphicsは、サンプル実装内でのグラフィックアプリケーションの作成を可能にします。この記事の冒頭[図2]で示したように、これらのライブラリはX、 OpenGL、その他の描画機能に対応しています。
lsbsi-testは、サンプル実装を、より適切なテスト環境にするためのソフトウェアの追加を可能にします。
コマンド6-11は、/optにtarballを解凍・展開し、/opt/lsbsi-ia32ディレクトリを作成します。コマンド8は省略しないでください。lsbsi-lsbsi-graphics-ia32-3.2.0.tar.bz2、 lsbsi-boot-ia32-3.2.0.tar.bz2、およびlsbsi-test-ia32-3.2.0.tar.bz2は、/opt/lsbsi-core-ia32に解凍・展開されなくてはなりません。

前出のビルド環境と同様、解凍・展開終了後のサンプル実装へは、chroot /opt/lsbsi-core-ia32 /bin/bashコマンドを実行することで移行できます。chroot環境を終了するには、新しいシェルプロンプトの位置でCtrl+Dを押してください。

ビルド環境・サンプル実装におけるコードのビルドと実行

リスト1に示した簡単なコードをビルド環境内でビルドし、サンプル実装上で実行してみます。

まず、ファイルをビルド環境へコピーします。

FC9 VM 内には、FC9、ビルド環境、サンプル実装の 3 タイプの環境が存在しています。そこで、便宜的に作業ディレクトリを 3 個作成し、そのそれぞれに環境変数を設定して、これらの間でのファイルコピーを簡単にします。

```
$ FC9=/tmp/work
$ BE=/opt/lsb-buildenv-ia32/tmp/work
$ SI=/opt/lsb-core-ia32/tmp/work
$ mkdir $FC9 $BE $SI
```

$FC9 ディレクトリは、ソースコードをダウンロードし、必要に応じて中間ファイルを保管するための作業ディレクトリです。例えば、LSB ビルド環境のユーティリティを用いて作成された LSB 準拠のバイナリを$FC9 にダウンロードし、次にそれを$SI ディレクトリにコピーして、LSB3.2.0 の仕様通りに実行できるかどうか確認するといった使い方をします。（全ては完璧に動作するはずです。）$BE ディレクトリは、chroot ビルド環境でビルドしたいソースコード用のディレクトリです。chroot /opt/lsbsi-buildenv-ia32 /bin/bash コマンド実行後は、/tmp/work として利用可能です。同様に、$SI ディレクトリは、ビルド環境でビルドしてサンプル実装上で実行したいバイナリの保存場所となります。chroot /opt/lsbsi-core-ia32 /bin/bash コマンド実行後は、/tmp/work として利用可能です。

従って、リスト 1 のコードを chroot ビルド環境でビルドし、chroot サンプル実装でテストする場合は、まずコードを FC9 環境にダウンロードし、次に以下のシーケンスを実行します。（解りやすいように、各コマンドラインプロンプトの頭にプリフィクスをつけてあります。）

```
(fc9) $ cd $FC9
(fc9) $ wget ftp://.../echoargs.c
(fc9) $ cp echoargs.c $BE
(fc9) $ chroot /opt/lsbsi-buildenv-ia32 /bin/bash
(be)  $ cd /tmp/work
(be)  $ cc -o echoargs echoargs.c
(be)  $ ./echoargs hello there
hello there
(be)  $ Control-D
(fc9) $ cp $BE/echoargs $SI
(fc9) $ chroot /opt/lsbsi-core-ia32 /bin/bash
(si)  $ cd /tmp/work
(si)  $ ./echoargs hello there
```

hello there
(si) $ Control-D
(fc9) $ echo $FC9
/tmp/work

FC9 は LSB3.2.0 に準拠していますので、ビルド環境で作成されたバイナリは FC9 上で問題なく動作します。

(fc9) $ cp $BE/echoargs $FC9
(fc9) $ cd $FC9
(fc9) $ ./echoargs this is cool
this is cool

まとめ

本チュートリアルで提示したサンプルコードは、いずれも数行で非常に単純ですが、それらのビルドやテストに使用したテクニックは、例えコードの行数が何千に増えたとしても全く同様に適用できます。LSB ビルド環境ユーティリティとスタンドアロン型の chroot LSB ビルド環境を利用してのコードのビルドを、是非お試しください。LSB3.2.0 に準拠した Linux をお持ちでない場合は、ポータビリティのテストに LSB サンプル実装がご利用頂けます。また、VMWare Workstation のようなツールを有効活用して、コンピューティングリソースの実質的な増強、自分に合った特色の Linux の幅広い活用を目指してください。

ほんの少しの努力で、アプリケーションを LSB に準拠させ、認証を得ることは可能です。一方、認証されたディストリビューションやアプリケーションは、Linux の快適さに対するユーザーの投資を増加させる一助となるはずです。LSB は同一性を促進するとともに、逆説的な言い方をすれば、選択の幅も広げます。

何と言っても、選択肢の多さが Linux の最大の強みなのですから。

出典

この文書は、IBM developerWorks に掲載された記事を、最新の技術の実態に沿うよう改訂したものです。アップデートにご協力下さいました Ted T'So、Jeff Liquia の両氏に感謝致します。

アプリケーションを世界に広める

このセクションでは、ポータビリティの目標を達成した後で行うべきことについて少し述べてみます。認証を受けるための手続きは？複数のディストリビューション間でマルチ互換性を実現した後になすべきことは？答えは、このセクションで見つけられます。

「アプリケーションの認証」では、Linux Standard Base の認証手続きを解説します。

「アプリケーションのリスティング」では、ポータブルにされたアプリケーションをマーケットに深く浸透させるための貴重な情報を提供します。

アプリケーションの認証

Linux Foundation は、LSB 標準に準拠しているアプリケーションについて認証を提供しています。認証された製品のみが LSB 認証のトレードマークを使用することが許可されます。この認証マークは、開発者とエンドユーザーに、「LSB 認証」されたアプリケーションが LSB 認証されたディストリビューション上で正常に動作することを示します。

登録

認証を取得するための第一ステップは、Linux Foundation の Web サイトで自分のアカウントを作成することです。

ログイン後、まずご自身と会社について明確にしてください。同一会社内で別の誰かが既に会社名のアカウントを持っている場合は、その人が追加の手続きを取る形で登録してください。さもないと、新しい会社名で登録しなくてはならなくなります。

テストと認証

アプリケーションが LSB 仕様に準拠していることを確認するために、Linux Application Checker を用いて適正な認証テストを実行してください。

テストで見つかった問題点（LSB の一部ではないリンクライブラリでコンパイルされている等）を修正してください。

もし、LSB 準拠テストプロセスで(例えば、合格するはずのテストに失敗したなどの)問題が発生した場合は、LSB 実行時、および/または、LSB SDK のバグに遭遇した可能性もあります。その場合は、まず List of Outstanding Problem Report を参照してください。もし、今回発生した障害が以前の報告事例にない場合は、新しい問題として報告してください。Linux Foundation で調査して、原因が本当に実行テストや SDK にあることが判明した場合はウェーバー（一時的回避）を認めテストの失敗が LSB 準拠認証に影響を与えないようにします。

最終審査

1. 正式認証を受けるためにテスト結果を提出する用意が整ったら、認証システムにアプリケーションを登録してください。登録手続きはオンラインまたは Linux Application Checker で行えます。

2. テスト結果は、製品の登録が完了した後でアップロードしてください。最も簡単な方法は Linux Application Checker を使用することですが、認証システム内のご自分の製品ページからも行えます。

3. その後、LSB 商標ライセンス契約（Trademark License Agreement = TMLA）に署名し、所定の申請費用をお支払い頂くことになります。TMLA は認証システム内の製品ページから入手可能です。さらに詳しい情報が必要な場合、または、これらの文書のコピーが必要な場合等は、info@linuxfoundation.org にご連絡ください。

4. Linux Foundation によるテスト結果の審査は、TMLA 締結と費用の支払いが確認された後で開始されます。（テスト結果が審査を通らなかった場合は、審査で明らかになった問題点を解決し、再テストを行い、その結果をもう一度提出する必要があります。）ウェーバーにより回避したテストににについては、Linux Application Checker に、そのことの認識されています。

5. 審査に合格した製品は、LSB 認証製品リストに記載されます。Linux Foundation は、その製品が LSB 認証の必要条件を継続して満足しているか、ときどき証拠の提出を求めることがあります。

6. LSB 標準との互換性をユーザーに知らせるために、商標登録された LSB 認証名とロゴを製品につけることをお勧めします。詳しくは、Linux Foundation 発行の Trademark Usage Guidelines を参照してください。

アプリケーションのリスティング

アプリケーションのポータビリティを最大化する旅を終えたあなたは今、幅広いLinuxユーザーが潜在顧客として目の前に存在することに満足を覚えていることでしょう。LSB認証を取得した後では、特にそう思うに違いありません。

しかしながら、自分のアプリケーションはポータブルであると認識することと、そのメリットを生かして活動することとは全く別物です。あるディストリビューションにあなたのアプリケーションが移植されたとして、そのアプリケーションをディストリビューションのパッケージセットに組み込んでもらうために誰とコンタクトを取り、どんな手順を踏めば良いのかご存じですか？

これからは、このページから目を離さないでください。LDNは、あなたのアプリケーションを可能な限り効率的に流通させるためのハウツーや具体的な連絡先情報を提供してまいります。

付録

LSB チャーター

ミッション

Linux Standard Base (LSB)は、Linux ディストリビューション間の互換性を向上させ、LSB の定める標準に適合したソフトウェア製品が任意の準拠システムで動作するこ とを可能にするような、一定の標準を作り上げることを目差したプロジェクトです。すなわち、Linux プラットフォーム用のソフトウェア製品を製品化する ベンダーは、いくつものディストリビューションを意識することなく、ただ一つのターゲットに向けて製品を作れば良くなるわけです。このような標準は、本プ ロジェクトに参加するメンバーの合意に基づいて形成されていきます。さらに、LSB は、標準準拠製品を開発するベンダーを支援する活動も行います。

プロジェクトの構造

LSB は、LF の主要な標準化活動であり、Linux プラットフォームのより専門化された領域（例、アクセシビリティ、国際化、印刷、コンパイラとその関連ツールなど）で標準化を推進するいろいろなワークグループの「傘」として位置づけられます。（注１）

LSB とその下位ワークグループは、単一のロードマップを共有しています。各ワークグループは、その成果として、一定の標準仕様と、その仕様の適用を推進するために必要な、適合テスト、開発ツール、あるいは（必要な場合は）レファレンス実装など、いろいろなソフトウェアを提供します。

それぞれの仕様および関連するソフトウェアは、LSB モジュールとして知られています。LSB モジュールは、相互に統合できるように設計された共通形式を 備え、それぞれ、必須モジュールまたはオプションモジュールとされます。必須モジュールは、一定の受入基準を満たすモジュールです。オプションモジュール は、受入基準をまだ満たしてなく、開発中または構築中のフィーチャーで、LSB 標準の将来のバージョンに含まれる予定のものです。

（注 1） 標準を作成しない LF ワークグループもあります。このようなワークグループは、上流の Linux カーネル、他のパッケージまたは文書、もしくはその両方に 向けてソフトウェアコードを作成します。本チャーターは、標準を作成するワークグループ、すなわち LSB ワークグループに対してのみ適用されます。

範囲

すべてのLSBワークグループは、最終的には、すべての主要なLinuxディストリビューションで幅広く採用され、利用可能になるような標準を作成するこ とが期待されています。幅広い採用に沿わない活動は、明確にLSBワークグループの範囲外です。また、すべてのLSBワークグループの範囲は、議長の承認 するものを除き、現に実践されいるものでなければなりません。特定の技術をLSBに含めるための唯一の基準は、ベスト・プラクティス（最良の実践）と呼ぶ もの、すなわち、すべての主要なLinuxディストリビューションで出荷され、安定したABI(Application Binary Interface)を持ち、さらに、その技術の必要性が高いことです。

成果

LSBは、LSBロードマップに沿って版数を上げて行きます。各リリースのとき、LSBは、すべての必要なモジュールをLSB仕様と呼ばれる単一の文書にまとめます。

LSB仕様には、製品がそのバージョンに準拠していることをベンダーがテストすることを可能にするテストキットが 付属します。製品の標準準拠の正式な表示を求めるベンダーに対して、LFはテストの結果に基づき認定を与えます。テストキットは、各認定につき1つです（例、LSB Distribution Testkitは、Linuxディストリビューションの認定用、LSB Application Testkit はアプリケーション認定用など）。

また、LSBワークグループは、LSB仕様に対するアドオンとして、オプションモジュールを公開することがあります。オプションモジュールは、LSBに対 する近い将来の追加について、幅広い公開レビューを容易にするために提供されます。また、オプションモジュールにより、ベンダーの将来のバージョンへの収 束を加速することができます。すなわち、どのモジュールがベスト・プラクティスとして登場し、将来のバージョンに含まれるようになるのか、「自分たちの仲 間とともに投票」できるわけです。

リーダーシップと統治

注：この項では、このチャーターがLSBとそのワークグループでより容易に共有されるように分かりやすい用語を使用します。

ワークグループは、選出された議長によってリードされ、ワークグループの幅広い利益を代表する運営委員会によって統治されます。

ワークグループは、IETF のような「大まかなコンセンサス」モデルで運営されます。「大まかなコンセンサス」は議長によって判断されます。議長による決 定が、貢献者に大まかな合意を反映していないと感じられるようなケースでは、最初に運営委員会に、そして必要に応じて、LSB 運営委員会に上告することが できます（LSB 運営委員会が実施するワークグループの場合、決定は LF 理事会に上告することができます）。

ワークグループの運営は、オープンフォーラム（通常は wiki、メーリング・リスト、電話会議、定期的な対面会議）で実施されます。会員資格に制限はあり ません。決定は、そのオープンフォーラムにおいて、未解決の問題も含めて明確に文書化されなければなりません。これは、将来の議論の基礎として使用されま す。

貢献者は、ワークグループに積極的に関わっている任意の個人です。「積極的に関わっている」と言っても主観的な判断になりかねないので、議長は貢献者と考 えられる個人のリストを作成します。個人は、貢献者リストへの掲載を議長に要請することができ、異議がある場合、上記のように上告できます。

議長は、プロジェクト全体のリーダーであり、貢献者によって選出され、LSB 運営委員会と LF 理事会による承認の対象となります。議長の任期は 2 年間で す。任期の回数に制限はありません。議長は、運営委員会、LSB 運営委員会、または LF 理事会のいずれかによる不信任決議により解任できます。その場合、 議長の席は空になり、90 日以内に選挙が実施されます。空の間は、運営委員会が議長代理を任命します。

運営委員会は、主要なワークグループ利害関係者の代表から構成されます。LSB の場合、これらの利害関係者には、ディストリビューター、 ISV、OEM、上流コミュニティの開発者、LSB チャーターの下で運営される LSB とワークグループの議長が含まれます。運営委員会の委員は、議長に よって任命され、ワークグループに積極的に参加し続け、かつワークグループの利益を最優先に業務を実施している限り、任期は無制限です。運営委員会の委員 は、運営委員会、LSB 運営委員会、または LF 理事会のいずれかによる不信任決議により解任できます。

議長選挙は、貢献者による臨時の選挙委員会によって管理されます。選挙委員は、議長の任期が切れるか、議長の席が空になったとき、運営委員会によって、こ れだけの目的のために任命されます。選挙委員会は、議長の任期が切れる少なくとも 30 日前、または議長の席が空になってから 10 日以内に、1 人または複数 の議長候補を選出する責任を負っています。選挙委員会の委員は、議長の候補になることはできません。選挙委員会は、資格のある投票者

のみが投票でき、投票 者1人につき1票のみが集計され、投票の秘密が保たれるような適切な手段を使用して、電子投票による選挙を実施します。投票期間は1週間です。選挙委員会は投票を集計し、その結果を運営委員会に提出します。運営委員会は結果を批准し、選挙結果を宣言します。

ワークグループ

LSB議長は、LSBチャーターの使命を実行するために、必要に応じてワークフループの結成と解散を行うことができます。新しいワークグループは、結成中 の段階では「インキュベータ」と考えられ、LSB運営委員会が標準化の正しい路線上にあると判断した段階で本格的なワークグループの地位に格上げされま す。この段階に到達するために、ワークグループは、本チャーターを採用し、十分な統治構造を持ち、幅広く採用される標準を開発するのに必要な主要な構成員 からなり、かつそのような幅広く採用される標準の開発に向けて積極的な進歩を続けていなければなりません。

ワークグループは、適切な程度の独立性を持って運営され、自身の詳細については自身で決定できます。各ワークグループには議長を置かなければなりません。 議長のおもな役割は、LSBプロジェクト全体の目標との整合性が保たれ、成果がプロジェクト全体のロードマップに沿ったものになるようにLSB議長と調整 を図ることです。それぞれのワークグループは、常に活動状態にあり、ロードマップに沿って成果を出さなければなりません。また、それぞれのワークグループ は、各ワークグループの統治構造に沿ってLSBチャーターを批准しなければなりません。ワークグループ議長は、LSBが「インキュベータ」段階を脱した時 点でLSB運営委員会の委員となります。

ワークグループが、自身の統治構造を持つことを望まない場合、LSBチャーターに記述された統治構造をそのまま使用することが推奨されます。ワークグルー プは、自身の運営委員会を持つことも持たないことも可能です。運営委員会を持たない場合、本チャーターに記述された機能（例、選挙管理、紛争解決など）を 実行するためにLSB 運営委員会を利用することができます。新しいワークグループを立ち上げるには、そのワークグループの発起人が大まかな合意により議長を選出するか、大まか な合意に達しない場合、LSB議長がそのワークグループの議長を任命します。

LFはLinux関連の組織ですが、LSBワークグループは、Linuxに限らず、より広い適用範囲を持つ標準に関する作業を行うことができます。これ は、たとえLSBの文脈の中でも、推奨されるものとみなされます。これは、クロスプラットフォームのLSBモジュールが増加するにつれ、ISVが Linuxプラットフォームを対象として行う必要のあるLinux固有の作業が少な

くなり、Linux 移植の費用効果が高くなり、さらに、Linux 移植 が行われる機会が増えているからです。

アクティビティ

2007 年 2 月 7 日：LSB 運営委員会承認

LSB ロードマップ

はじめに

LSB は、各種の Linux ディストリビューション間の「最大公約数」を提供すること、すなわちソフトウエア開発会社（ISV）が Linux プラット フォーム上のソフトウエアの作成または移植のための単一の開発目標を提供することを目的としています。ここで、「Linux プラットフォーム」とは、アプ リケーションが動作しなければならないディストリビューションの候補（かつ ISV によって異なる可能性のある）のことです。

有効な最大共通項としての役割を果たすには、LSB のバージョンからディストリビューションへおよびその逆へのマッピングが容易でなければなりません。ま た、アプリケーションが LSB のあるバージョンをターゲットとしたとき、そのバージョンだけではなく、将来のバージョンでも動作することが保証されている 必要があります。すなわち、LSB 3.0 をターゲットとしたアプリケーションは、LSB 3.0、3.1、3.2、4.0 準拠のディストリビューション上でも動作することが保証されます。

このような「わかりやすい対応」の要件を満たすために、LSB の各メジャー・バージョンは、エンタープライズ・ディストリビューションのメジャー・バー ジョン、すなわち「世代」と対応しています。したがって、たとえば、LSB 3.x は現行の世代（Red Hat Enterprise Linux ４、SUSE Linux Enterprise ９など）、LSB 4.x は次の世代（RHEL ５、SLE １０など）に、それぞれ対応しています。アプリケーション互換性の要件を満足するために、LSB バージョン 3.0 以降は、メジャー・バージョンとマイ ナー・バージョンの両方とも、前のバージョンと厳密な後方互換性を持っています。

後方互換性

このような後方互換性を達成するため、LSB 標準においては、後続の LSB 版は仕様の追加のみ、言葉を換えると、インターフェースは追加されるのみで、急に削除されることはないということです。LSB のポリシーに、インターフェース削除のメカニズムを有していますが、本当に削除されるのは、そのインターフェースに「削除予定(deprecated)」と表示される状態を少なくとも LSB の版数で３版を経た後、あるいは、およそ６年を経た後としています。

LSB では、LSB 仕様からあるインターフェースが削除されるときにアプリケーション開発者が対応する十分な時間をとれるようなインターフェース削除ポリシーを定めています。こ

のポリシーのために、開発者はLSB仕様に信頼を置くことができ、また、変更に対する計画的な対応が可能となります。

インターフェースの削除は次のようなプロセスを経て行われます。

- あるインターフェースはLSBワークグループによって削除することが決定されます。
- LSB仕様の次回改版時に「削除予定(deprecated)」と表示されます。
- 削除予定となってから3つ目のメジャー版の最後までインターフェースを残されます。
- LSB仕様書の中で「削除(deleted)」と表示され、標準からなくなります。LSBデータデース上では残っています。

ハイレベルなLSBロードマップは次のようになっています。

LSB 3.x (2006-2008)	LSB 4.x (2008-2010)
Asianux 2.0 Debian 4.0 ("etch") Mandriva Corporate 4.0 Red Hat Enterprise Linux 4 and 5 SUSE Linux Enterprise 9 and 10 Ubuntu 6.06 LTS ("dapper") [...]	Asianux 3.0 Mandriva Corporate 5.0 Red Hat Enterprise Linux 5 SUSE Linux Enterprise 10 Ubuntu LTS 8.04.4 [...]

おおまかなLSB4.0のロードマップは以下の通りです。より技術的な最新情報は以下のProjectPlaan40のサイトをご参照ください。

http://www.linuxfoundation.org/en/ProjectPlan40

LSB 4.0 (2008年)

スケジュール
2008年4月14日　　計画の確定

2008年8月1日	機能確定
2008年10月3日	ベータ1版
2008年10月31日	RC1
2008年11月11日	LSB4.0

機能の計画

* glibc 2.4
* LSB 3.xとのバイナリ互換性
* SDK使い易さ
* GtK・Cairoの新版
* Java
* LSB準拠のRPMpパッケージを容易に作製できるようにする
* 暗号化API

過去のロードマップ

過去の記録として参考まで。

http://www.linuxfoundation.org/en/Lsb32Roadmap

役に立つWebリンク集

LSB仕様書のダウンロードサイト。
http://www.linuxfoundation.org/en/Specifications

ディストリビューション、および、アップリケーションの認証の情報を提供するサイト。
http://www.linuxfoundation.org/en/Certification

認証済みのディストリビューション、および、アプリケーションのリスト。
https://www.linuxfoundation.org/lsb-cert/productdir.php?by_prod

主要Linuxディストリビューションのインターフェースの現状。LSB認証状況、採用しているカーネル、gcc、glibc、GNOME、KDEの版数など。
http://ldn.linuxfoundation.org/lsb/distro-component-matrix

アプリケーション、および、ディストリビューションの認証費用。登録商標LSB、および、そのロゴマークの使用許可費を含む。The Linux Foundationメンバーは半額。2008年度はアプリケーション認証費は免除。
http://www.linuxfoundation.org/en/Fee_Schedule

LSBフォーラム。LSB、アプリケーション開発、ApplChecker、アプリケーションのポータビリティ、アプリケーション認証、などについてLSBコミュニティが質問に回答する。
http://ldn.linuxfoundation.org/support

LSB Database Navigator
http://dev.linuxfoundation.org/navigator/commons/welcome.php

Linux Application Checker、LSB Software Developemnt Kit(SDK)、LSBサンプル実装(LSB SI)、などのダウンロードサイト。ソースコードのダウンロードサイトへのリンクもあり。
http://ldn.linuxfoundation.org/support/downloads