NOTE:
This is a version of Documentation/stable_api_nonsense.txt into Japanese.
This document is maintained by IKEDA, Munehiro <m-ikeda@ds.jp.nec.com>
and the JF Project team <http://www.linux.or.jp/JF/>.
If you find any difference between this document and the original file
or a problem with the translation,
please contact the maintainer of this file or JF project.

Please also note that the purpose of this file is to be easier to read
for non English (read: Japanese) speakers and is not intended as a
fork. So if you have any comments or updates of this file, please try
to update the original English file first.

Last Updated: 2007/07/18
==================================
これは、
linux-2.6.22-rc4/Documentation/stable_api_nonsense.txt の和訳
です。
翻訳団体： JF プロジェクト < http://www.linux.or.jp/JF/ >
翻訳日 ： 2007/06/11
原著作者: Greg Kroah-Hartman < greg at kroah dot com >
翻訳者 ： 池田 宗広 < m-ikeda at ds dot jp dot nec dot com >
校正者 ： Masanori Kobayashi さん < zap03216 at nifty dot ne dot jp >
Seiji Kaneko さん < skaneko at a2 dot mbn dot or dot jp >
==================================

Linux カーネルのドライバインターフェース
（あなたの質問すべてに対する回答とその他諸々）

Greg Kroah-Hartman <greg at kroah dot com>

この文書は、なぜ Linux ではバイナリカーネルインターフェースが定義
されていないのか、またはなぜ不変のカーネルインターフェースを持たな
いのか、ということを説明するために書かれた。ここでの話題は「カーネ
ル内部の」インターフェースについてであり、ユーザー空間とのインター
フェースではないことを理解してほしい。カーネルとユーザー空間とのイ
ンターフェースとはアプリケーションプログラムが使用するものであり、
つまりシステムコールのインターフェースがこれに当たる。これは今まで
長きに渡り、かつ今後も「まさしく」不変である。私は確か 0.9 か何か
より前のカーネルを使ってビルドした古いプログラムを持っているが、そ
れは最新の 2.6 カーネルでもきちんと動作する。ユーザー空間とのイン
ターフェースは、ユーザーとアプリケーションプログラマが不変性を信頼
してよいものの一つである。

要旨

あなたは不変のカーネルインターフェースが必要だと考えているかもしれ
ないが、実際のところはそうではない。あなたは必要としているものが分
かっていない。あなたが必要としているものは安定して動作するドライバ
であり、それはドライバがメインのカーネルツリーに含まれる場合のみ得
ることができる。ドライバがメインのカーネルツリーに含まれていると、

他にも多くの良いことがある。それは、Linux をより強固で、安定な、成熟したオペレーティングシステムにすることができるということだ。これこそ、そもそもあなたが Linux を使う理由のはずだ。

はじめに

カーネル内部のインターフェース変更を心配しなければならないドライバを書きたいなどというのは、変わり者だけだ。この世界のほとんどの人は、そのようなドライバがどんなインターフェースを使っているかなど知らないし、そんなドライバのことなど全く気にもかけていない。

まず初めに、クローズソースとか、ソースコードの隠蔽とか、バイナリのみが配布される使い物にならない代物[訳注(1)]とか、実体はバイナリコードでそれを読み込むためのラッパー部分のみソースコードが公開されているとか、その他用語は何であれ GPL の下にソースコードがリリースされていないカーネルドライバに関する法的な問題について、私は「いかなる議論も」行うつもりがない。法的な疑問があるのならば、プログラマである私ではなく、弁護士に相談して欲しい。ここでは単に、技術的な問題について述べることにする。（法的な問題を軽視しているわけではない。それらは実際に存在するし、あなたはそれをいつも気にかけておく必要がある）

訳注(1)
「使い物にならない代物」の原文は "blob"

さてここでは、バイナリカーネルインターフェースについてと、ソースレベルでのインターフェースの不変性について、という二つの話題を取り上げる。この二つは互いに依存する関係にあるが、まずはバイナリインターフェースについて議論を行いやっつけてしまおう。

バイナリカーネルインターフェース

もしソースレベルでのインターフェースが不変ならば、バイナリインターフェースも当然のように不変である、というのは正しいだろうか? 正しくない。Linux カーネルに関する以下の事実を考えてみてほしい。

- あなたが使用するＣコンパイラのバージョンによって、カーネル内部の構造体の配置構造は異なったものになる。また、関数は異なった方法でカーネルに含まれることになるかもしれない（例えばインライン関数として扱われたり、扱われなかったりする）。個々の関数がどのようにコンパイルされるかはそれほど重要ではないが、構造体のパディングが異なるというのは非常に重要である。
- あなたがカーネルのビルドオプションをどのように設定するかによって、カーネルには広い範囲で異なった事態が起こり得る。
  - データ構造は異なるデータフィールドを持つかもしれない
  - いくつかの関数は全く実装されていない状態になり得る
    （例：SMP向けではないビルドでは、いくつかのロックは中身がカラにコンパイルされる）
  - カーネル内のメモリは、異なった方法で配置され得る。これはビルドオプションに依存している。
- Linux は様々な異なるプロセッサアーキテクチャ上で動作する。

あるアーキテクチャ用のバイナリドライバを、他のアーキテクチャで正常に動作させる方法はない。

ある特定のカーネル設定を使用し、カーネルをビルドしたのと正確に同じCコンパイラを使用して単にカーネルモジュールをコンパイルするだけでも、あなたはこれらいくつもの問題に直面することになる。ある特定のLinux ディストリビューションの、ある特定のリリースバージョン用にモジュールを提供しようと思っただけでも、これらの問題を引き起こすには十分である。にも関わらず Linux ディストリビューションの数と、サポートするディストリビューションのリリース数を掛け算し、それら一つ一つについてビルドを行ったとしたら、今度はリリースごとのビルドオプションの違いという悪夢にすぐさま悩まされることになる。また、ディストリビューションの各リリースバージョンには、異なるハードウェア（プロセッサタイプや種々のオプション）に対応するため、何種類かのカーネルが含まれているということも理解して欲しい。従って、ある一つのリリースバージョンだけのためにモジュールを作成する場合でも、あなたは何バージョンものモジュールを用意しなければならない。

信じて欲しい。このような方法でサポートを続けようとするなら、あなたはいずれ正気を失うだろう。遠い昔、私はそれがいかに困難なことか、身をもって学んだのだ・・・

不変のカーネルソースレベルインターフェース

メインカーネルツリーに含まれていない Linux カーネルドライバを継続してサポートしていこうとしている人たちとの議論においては、これは極めて「引火性の高い」話題である。[訳注(2)]

訳注(2)
「引火性の高い」の原文は ″volatile″。
volatile には「揮発性の」「爆発しやすい」という意味の他、「変わりやすい」「移り気な」という意味がある。
「（この話題は）爆発的に激しい論争を巻き起こしかねない」ということを、「（カーネルのソースレベルインターフェースは）移ろい行くものである」ということを連想させる ″volatile″ という単語で表現している。

Linux カーネルの開発は継続的に速いペースで行われ、決して歩みを緩めることがない。その中でカーネル開発者達は、現状のインターフェースにあるバグを見つけ、より良い方法を考え出す。彼らはやがて、現状のインターフェースがより正しく動作するように修正を行う。その過程で関数の名前は変更されるかもしれず、構造体は大きく、または小さくなるかもしれず、関数の引数は検討しなおされるかもしれない。そのような場合、引き続き全てが正常に動作するよう、カーネル内でこれらのインターフェースを使用している個所も全て同時に修正される。

具体的な例として、カーネル内の USB インターフェースを挙げる。USB サブシステムはこれまでに少なくとも 3 回の書き直しが行われ、その結果インターフェースが変更された。これらの書き直しはいくつかの異なった問題を修正するために行われた。
- 同期的データストリームが非同期に変更された。これにより多数のド

ライバを単純化でき、全てのドライバのスループットが向上した。今やほとんど全ての USB デバイスは、考えられる最高の速度で動作している。
- USB ドライバが USB サブシステムのコアから行う、データパケット用のメモリ確保方法が変更された。これに伴い、いくつもの文書化されたデッドロック条件を回避するため、全ての USB ドライバはより多くの情報を USB コアに提供しなければならないようになっている。

このできごとは、数多く存在するクローズソースのオペレーティングシステムとは全く対照的だ。それらは長期に渡り古い USB インターフェースをメンテナンスしなければならない。古いインターフェースが残ることで、新たな開発者が偶然古いインターフェースを使い、正しくない方法で開発を行ってしまう可能性が生じる。これによりシステムの安定性は危険にさらされることになる。

上に挙げたどちらの例においても、開発者達はその変更が重要かつ必要であることに合意し、比較的楽にそれを実行した。もし Linux がソースレベルでインターフェースの不変性を保証しなければならないとしたら、新しいインターフェースを作ると同時に、古い、問題のある方を今後ともメンテナンスするという余計な仕事を USB の開発者にさせなければならない。Linux の USB 開発者は、自分の時間を使って仕事をしている。よって、価値のない余計な仕事を報酬もなしに実行しろと言うことはできない。

セキュリティ問題も、Linux にとっては非常に重要である。ひとたびセキュリティに関する問題が発見されれば、それは極めて短期間のうちに修正される。セキュリティ問題の発生を防ぐための修正は、カーネルの内部インターフェースの変更を何度も引き起こしてきた。その際同時に、変更されたインターフェースを使用する全てのドライバもまた変更された。これにより問題が解消し、将来偶然に問題が再発してしまわないことが保証される。もし内部インターフェースの変更が許されないとしたら、このようにセキュリティ問題を修正し、将来再発しないことを保証することなど不可能なのだ。

カーネルのインターフェースは時が経つにつれクリーンナップを受ける。誰も使っていないインターフェースは削除される。これにより、可能な限りカーネルが小さく保たれ、現役の全てのインターフェースが可能な限りテストされることを保証しているのだ。（使われていないインターフェースの妥当性をテストすることは不可能と言っていいだろう）

これから何をすべきか

では、もしメインのカーネルツリーに含まれない Linux カーネルドライバがあったとして、あなたは、つまり開発者は何をするべきだろうか? 全てのディストリビューションの全てのカーネルバージョン向けにバイナリのドライバを供給することは悪夢であり、カーネルインターフェースの変更を追いかけ続けることもまた過酷な仕事だ。

答えは簡単。そのドライバをメインのカーネルツリーに入れてしまえばよ

い。（ここで言及しているのは、GPL に従って公開されるドライバのことだということに注意してほしい。あなたのコードがそれに該当しないならば、さよなら。幸運を祈ります。ご自分で何とかしてください。Andrew と Linus からのコメント＜Andrew と Linus のコメントへのリンクをここに置く＞をどうぞ）ドライバがメインツリーに入れば、カーネルのインターフェースが変更された場合、変更を行った開発者によってドライバも修正されることになるだろう。あなたはほとんど労力を払うことなしに、常にビルド可能できちんと動作するドライバを手に入れることができる。

ドライバをメインのカーネルツリーに入れると、非常に好ましい以下の効果がある。

- ドライバの品質が向上する一方で、（元の開発者にとっての）メンテナンスコストは下がる。
- あなたのドライバに他の開発者が機能を追加してくれる。
- 誰かがあなたのドライバにあるバグを見つけ、修正してくれる。
- 誰かがあなたのドライバにある改善点を見つけてくれる。
- 外部インターフェースが変更されドライバの更新が必要になった場合、誰かがあなたの代わりに更新してくれる。
- ドライバを入れてくれとディストロに頼まなくても、そのドライバは全ての Linux ディストリビューションに自動的に含まれてリリースされる。

Linux では、他のどのオペレーティングシステムよりも数多くのデバイスが「そのまま」使用できるようになった。また Linux は、どのオペレーティングシステムよりも数多くのプロセッサアーキテクチャ上でそれらのデバイスを使用することができるようにもなった。このように、Linux の開発モデルは実証されており、今後も間違いなく正しい方向へと進んでいくだろう。:)

この文書の初期の草稿に対し、Randy Dunlap, Andrew Morton, David Brownell, Hanna Linder, Robert Love, Nishanth Aravamudan から査読と助言を頂きました。感謝申し上げます。