CLI を実装する次世代オペレーティングシステムの開発
— "Cooperative Operating System, CooS" —

1．背景

近年、OS は劇的に高機能・複雑化しています。これはコンピュータがあらゆる場面で使われているため、アプリケーションからの多様な要求を OS が満たすことが求められているからです。共通して使われる処理を OS が実装することで、アプリケーション開発者は独自の処理により集中して開発することができるようになります。

これはアプリケーションから見れば確かに好都合ですが、しかし一方で OS 自体の肥大化、複雑化という問題をもたらしました。共通処理の実装は OS としては正しいことですが、それによって OS 自体の開発に支障が生じています。

これは、近年のセキュリティに関する問題として顕著に表れています。ソフトウェアのバグによるセキュリティホールは何種類か存在しますが、その大部分は「メモリに関する単純なミス」や「チェックの抜け」など､非常に単純なものです。原因は単純であるのにミスがなくならないのは、ソフトウェアが複雑で人が管理しきれないからに他なりません。

従来はこのようなミスにはトライアンドエラーによる手探りの対応や、ソースコード検査ツールなどを使った原始的な対策が存在しました。しかし、プログラムをプログラムでチェックできなかったため体系的な検査が不可能で、限定的な効果に留まっていました。

ところが近年、検証可能コードなどを特徴とする技術が CLI (.NET) や Java などで提案され、これらの問題に対して体系的な回答を示しました。しかし、これらはアプリケーションでは使われているものの、その特徴をもっとも生かせるシステムソフトウェアにはほとんど適用されていません。

2．目的

そもそも CLI のような技術がなぜシステムプログラムに適用されないかというと、CLI それ自体が OS やミドルウェアのような基盤プログラムを必要としてしまうからです。そのため既存の CLI 実装系は絶対に OS として動作できません。

本プロジェクトでは OS として CLI 実装を動作させる方法を提案し、実際に開発することで手法の妥当性を実証しました。

3．開発の内容

本プロジェクトでは "CooS"（クース）として次のようなサブシステムを含む OS を開発しました。

- ブートストラップ・ローダ
- レガシーカーネル
    - 各種デバイスドライバ
    - 簡易リフレクション

・CIL インタープリタ
・マネージカーネル
・リフレクション
・CIL コンパイラ
・各種デバイスドライバ
・シェル

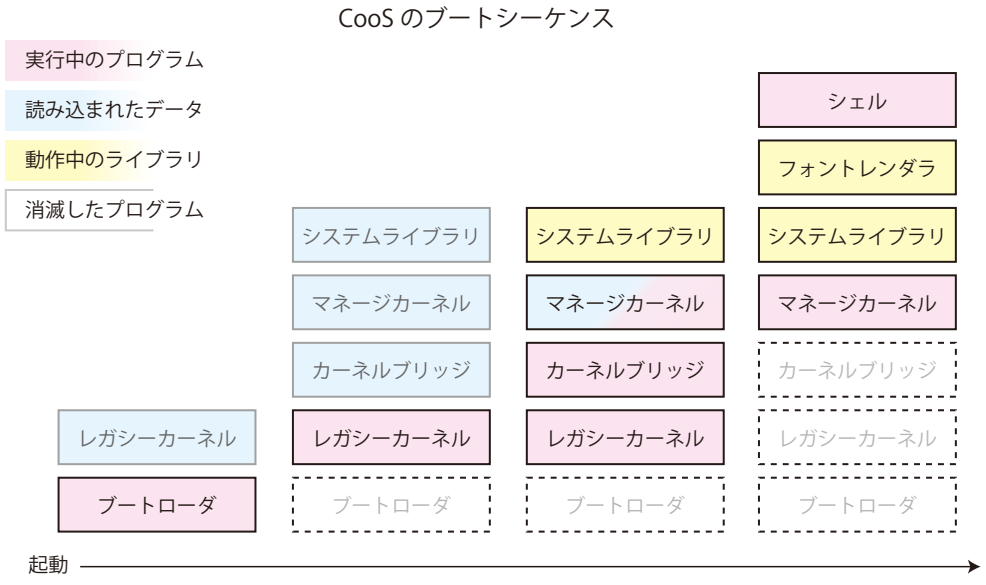
また、標準ライブラリとして次のようなものをポーティングし、上記のプログラムと統合しました。

・Mono システムライブラリ
・FreeType2 フォントレンダラ

さらに、デモンストレーション用にアプリケーションを開発しました。これらはデバイスドライバ、ファイルシステム、コンパイラ、フォントレンダラ、グラフィックスを動作させるものです。

3.1　動作の流れ

CooS は制御を「ブートストラップ・ローダ」から「レガシーカーネル」「マネージカーネル」へとバトンのように渡していきます。制御を渡し終わったプログラムはコンピュータから消滅します。



従来の手法ではマネージカーネルに相当するものはレガシーカーネル相当の上で動作していましたので、提案手法ではこのバトンタッチが大きく違うところになります。

3.2　ブートストラップ・ローダ

ブートストラップ・ローダは既存のものを踏襲して開発しましたが、レガシーカーネル専用になっています。レガシーカーネルは細心の注意を払ってビルドされているため、ブートストラップ・ローダによって最適化された環境でないと動

作しないからです。

3.3　レガシーカーネル

レガシーカーネルには各種デバイスドライバを内包しており、BIOS に頼らず ATAPI CD-ROM ドライブからデータを読み込むことができます。これを使ってマネージカーネルを読み込み解析して、マネージカーネル内のコンパイラを解釈実行します。

よく OS の開発では C++ 言語ではなく C 言語が利用されていますが、レガシーカーネルは実行時に .NET アセンブリを解釈実行するなど、複雑な処理を実装する必要があります。これには C 言語では力不足であり、明らかに C++ が必須でした。

本プロジェクトでは通常の C++ 言語に加え STLport や Boost なども活用するなど、C++ のもつ機能をなるべく活かすことによって、レガシーカーネルを短期間に開発しました。

3.4　マネージカーネル

提案手法におけるマネージカーネルの主要な機能は、.NET アセンブリの詳細なリフレクションとコンパイラです。マネージカーネルはリフレクションによっては自分自身の解析を行い、その結果に基づいて自分自身をコンパイルします。

マネージカーネルは C# 言語で記述され中間言語 (CIL) へコンパイルされているため、そのままでは実行できません。そこで、中間言語をさらに機械語へコンパイルすることによって、マネージカーネルが自立して動作できるようにします。

3.5　その他

.NET システムライブラリとして用いた Mono はもともと Linux 向けであるため、実装の中に何カ所も OS の存在を仮定している箇所があります。当然そのままでは動作しませんので、CooS のための変更を施しました。

ただ、オリジナルを大きく書き換えることは保守の点で好ましくないので、可能な限り読み込み時に動的にパッチを当てるようにしました。実際にはソースコードの変更はアクセス修飾子の書き換えだけであり、ファイル I/O などはパッチによってマネージカーネル内のコードに移譲されるようになっています。

FreeType2 は OS 独立ですので Managed C++ 言語としてもともとコンパイル可能でしたが、メモリモデルが C 言語スタイルですので、GC を想定した CLI のメモリモデルとは適合しませんでした。

そこで、CLI によるメモリブロックを C 言語スタイルに変換するコードを加えることで、オリジナルには手を加えることなく CooS での動作を実現しました。

3.6　動作検証

実機での動作も視野に入れているものの、発表会などでの動作を鑑み、最終的な動作確認は VMware で行いました。

4. 従来の技術（または機能）との相違

従来手法による CLI の実装では、C 言語相当などで開発した普通の OS 上に仮

想マシンを動作させ、その中で CLI アプリケーションを実行しています。この方法を用いると、アプリケーションは CLI の利点を受けられるものの、システムソフトウェアが CLI を利用することは不可能です。

提案手法による CLI の実装では、ブート時に一時的に C 言語相当で開発されたプログラムが動作するものの、最終的には CLI を基盤とするプログラムだけがコンピュータ上で動作します。これによって、OS 自体も CLI を基本に開発することができるようになり、実行時だけではなく開発時にも利点が生まれました。

5．期待される効果

私見ですが、従来システムソフトウェアはある種の " 聖域 " であり、そこで起きたバグなどは天災として考えられることが多かったように思います。実際、アプリケーションプログラムに対する開発方法の議論はどこでもあるものの、システムソフトウェア・ミドルウェアに対しての適用を是とする話はあまり聞きません。

しかし私は、本プロジェクトの趣旨でもある「システムソフトウェアも正しく作られるべきだ」という認識を持っており、その一番具体的な表現が本プロジェクトで開発されたオペレーティングシステムということになります。

成果物の性質上、ただちに定量的な効果が出ることは残念ながら期待できませんが、上記のような取り組みを通じて、より安全なコンピューティングに貢献したいと考えています。

6．普及（または活用）の見通し

成果物の純粋な利用者については目処が立っていません。

ただ、開発したデバイスドライバなどのコードを公開するなどして、二次的なプロダクトの利用は計画しています。デバイスドライバなのに読みやすい C# で記述されているなど、開発者への参考資料としては価値があると思います。

また .NET プログラムの解析部分やコンパイラなどは再利用も可能なので、OS から分離して公開することも計画しています。

7．開発者名（所属）

高橋　明生　（武蔵工業大学大学院）

（参考）開発者 URL
http://www.coos.jp/