CLIを実装する次世代オペレーティングシステムの開発

高 橋 明 生†

CLI を基盤とした新しいアーキテクチャを持つ OS として新たに ”CooS” を開発し、その動作を実証した。CLI は Microsoft.NET の技術的な規格名である。提案手法によって CLI 本来の特徴を OS に対しても適用することが可能になり、システム全体をよりセキュアにできるなど、社会基盤として求められている性質を備えることができる。本研究では、CLI をカーネルにおいて実行する方法を提案する。

1. はじめに

近年、OS は劇的に高機能・複雑化している。これは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ある。共通して使われる機能を OS が実装することにより、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るが、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た。共通処理の実装は OS の役割ではあるが、それによって OS 自体の開発に支障が生じている。

これは、近年のセキュリティに関する問題として顕在化している。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するが、その中の大部分は「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ある。

従来はこのようなミスには手探りの対応や、ソースコード検査ツールなどを使った原始的な対策を行っていた。しかし、完全にエラーをなくすことは不可能であり、これらの手法による検証は限定的な効果に留まっている。

しかし近年、検証可能コードなどを特徴とする技術が .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た。特に.NET の技術体系が CLI (Common Language Infrastructure) である。これにより、前述のようなエラーがある場合でもソフトウェアが安全な動作をするようになっている。

2. 関連研究

2.1 Java OS

Java OS は思想的には CooS の Java 版であり、その研究動機に大きな違いはない。しかし、Java とその実行環境を調査した結果、次のような点で CLI をベースにした方が優位であると考えた。

- 規格的に、CLI の方が Java より広い。
- Java にはポインタがないのに対して、CLI ではポインタを使って処理を記述することができる。
- Java はメモリ領域を固定することができないが、CLI では pinned 修飾子や GCHandle を使用することで、確保したメモリ領域を一時的にガベージコレクションの対象から切り離し、物理的に固定することができる。
- Java プログラムは機械語を含むことができないが、CLI では機械語を含むことができる。

こういった問題点は、Java を使う既存研究[1] ではロード時に特殊なコードとリンクしたり、JNI (Java Native Interface) を利用して Java OS の外に追い出すことで解決している。

2.1.1 Singularity OS

Singularity[2] は Microsoft Research で研究されている、CooS と同様の特徴・性質を持つ（と予想される）OS である。Singularity も CLI をベースとして開発されている。

Singularity は予めネイティブコンパイラを使って中間言語を x86 命令にコンパイルするため、動作可能イメージが生成された段階では中間言語がなくなっていると思われる。CooS は動作中に中間言語を扱うことを目標にしている。

3. 提案手法

本研究では、従来仮想マシンなどによって実装されていた CLI を OS そのものの記述に用いることによって、既存手法よりも優れた OS 開発環境を実現し、ソフトウェアの品質向上を図る。

品質向上のための性質自体は CLI が本来的に持つ

† 武蔵工業大学大学院工学研究科
Musashi Institute of Technology

ものであり、本研究によるものとは言えない。しかし、現状では CLI の利点を受けることが出来るのはアプリケーションプログラムに限られており、CLI の特徴をもっとも活かせるであろうシステム寄りのプログラムでは全く利用されてこなかったため、CLI をシステムソフトウェアに適用することで得られる利点がある。

本研究は、今まで行われていなかった CLI の OS への適用を実践し、その実現可能性を確かめることを目的としている。

4. カーネルの実装

CLI をカーネルに実装するにあたって、もっとも困難な部分は「中間言語をいかにカーネルとして実行するか」と「起動から中間言語の実行環境をどう構築するか」にある。まず、CooS を構成するプログラムについて述べた後にそれらを説明する。

4.1 ソフトウェア構成

CooS は複数のサブシステムから成り立つ。そのうち、特に OS が使用するものを表 4.1 に示す。ただし、ブートローダと Mono クラスライブラリは直接関係はないため、説明は割愛する。

表 1 主要なサブシステム一覧

サブシステム名	ファイル名	記述言語
ブートローダ	bootldr.img	アセンブラ / C
レガシーカーネル	kernel.img	C++
マネージカーネル	cscorlib.dll	C#
	cskorlib.dll	Managed C++
	csbridge.dll	Managed C++
Mono ライブラリ	mscorlib.dll	C#
	System.dll	C#

レガシーカーネルは通常と同様の手法によるカーネルプログラムである。マネージカーネルを起動するために必要な環境を整えるのが一番重要な役割である。レガシーカーネルは、マネージカーネルの起動後少しの間もマネージカーネルの補助のために活動するが、最終的にはメモリから消滅する。

マネージカーネルは CooS の真のカーネルである。マネージカーネルは中間言語にコンパイルされているため、レガシーカーネルが構築する環境の補助がなくては動作を開始できない。しかし、初期化中に自己を動かすための環境を自分の手で整え、最終的にその環境の中で動き始める。

5. マネージカーネルの実行

本手法ではマネージカーネルの実行のために、JIT コンパイルによって実行時に機械語を得ることにした。コンパイラはマネージカーネル内に C#言語で記述されており、レガシーカーネル内の中間言語インタープリタで動作する。

5.1 ロードとコンパイル

レガシーカーネルは各種初期化処理の後、マネージカーネルのファイルを読み込む。そして、内容を解析し、インタープリタが利用するメタデータを準備する。この時点で、インタープリタによる実行は可能になる。

ここで、単純にはすぐにコンパイラを実行できるように思える。しかし実際には、コンパイルにあたってコンパイラ自身から扱えるメタデータを準備する必要があり、このままでは動作できない。(メモリ構造が異なるため、インタープリタのメタデータはコンパイラからは扱えない。)

メタデータの構築には、メタデータがマネージメモリ空間に存在する必要があるため、中間言語の実行によって構築することが簡便である。

そこで、マネージカーネルにメタデータ構築用のコードを実装しておき、レガシーカーネルが必要に応じて呼び出すようにした。この初期化コードはメタデータが利用できない状態でも動作するように最小限の機能だけを記述しており、レガシーカーネルのインタープリタによって駆動される。

6. おわりに

現状では部分的にインタープリタが介在してしまうため、特に割り込みなどのプリミティブな処理で問題があることが分かっている。また、インタープリタによるコンパイラのコンパイルには時間が掛かりすぎるため、実用には問題になることも分かっている。

そこで、今後はコンパイラだけをプリコンパイルすることで、インタープリタの排除と起動時間の短縮を図ることを予定している。

謝　辞

CooS の開発には、独立行政法人情報処理推進機構 (IPA) による 2004 年度第 2 回未踏ソフトウェア創造事業の支援を受けた。

参考文献

1) 山本茂樹、早川栄一: Java を用いた OS 構成法、情報処理学会研究会報告 2005-OS-98(5), Vol.2005, No.16, pp.33-40, Feb 2005.
2) Galen Hunt and James Larus, An Overview of the Singularity Project, http://research.microsoft.com/os/singularity/ , 2005.