

JavaVM 上で動作する x86 ユーザーモードエミュレータの実装と評価

川口 直也[†] 並木 美太郎^{††}

x86/Linux アプリケーションの異種 OS・アーキテクチャ間における可搬の利用を目的とした JavaVM 上で動作するユーザーモードエミュレータを実装し評価を行った。ユーザーモードエミュレータは機械語命令のエミュレーションのみを行い、システムコールは実機上の OS が提供する機能を利用するエミュレータである。筆者らは x86/Linux アプリケーションが利用する機能のうち、ファイル I/O、ネットワークソケット、GUI、マルチスレッド、共有ライブラリのロードといった機能をサポートするエミュレータを実装した。また、本エミュレータは xlib をリンクした GUI プログラムを Java アプレットとして Web ブラウザ上で実行することを可能としている。本エミュレータの性能は実機に対し演算性能で 2 万倍から 40 万倍程度、I/O 性能で 1600 倍から 2.8 万倍程度低速である。演算性能は低速であるが、I/O 性能が重要となる GIMP などのアプリケーションにおいては本エミュレータが有用であると考えられる。

Development and evaluation of an x86 user-mode emulator on JavaVM

NAOYA KAWAGUCHI[†] and MITARO NAMIKI^{††}

This paper described a development of a user-mode emulator on JavaVM to execute existing x86/Linux native applications on different OS or architectures, and evaluation of its performance. The user-mode emulator emulates x86 machine instructions only and system calls passed to native OS executing the emulator. The emulator can provide File I/O, network socket, GUI, multi-thread, and shared-library linking functions to emulated applications. Furthermore, the emulator makes GUI applications linking an xlib-library run on WEB browsers as Java-applets. The computing performance of the emulator is slower about 2,000 to 40,000 times which compared with a real machine, and the I/O performance is slower about 1,600 to 28,000 times. The I/O performance indicates that the emulator is useful to emulate GUI applications such as GIMP.

1. はじめに

ネイティブコードからなるユーザーモードアプリケーションの異種環境間での可搬の利用を目的とした、ユーザーモードエミュレータとして QEMU¹⁾ や NestedVM²⁾, VEELS³⁾ が存在する。ユーザーモードエミュレータは、ユーザーモードのプログラムが利用する機械語命令のエミュレーションのみを行い、システムコールは、エミュレータの動作環境に存在する実機上の OS が提供する機能を利用する。このため、エミュレータの動作環境に存在する実機の I/O 資源を、エミュレーションするアプリケーションから直接利用

可能である。特に JavaVM 上で動作する NestedVM や VEELS は、対象とするアプリケーションのみならずエミュレータ自身も含めて異種 OS・アーキテクチャ間での可搬の利用を可能としている。さらに VEELS は、Java アプレットとして Web ブラウザ上で利用可能である。しかし、これらのユーザーモードエミュレータは、マルチスレッドや GUI、共有ライブラリの動的リンクといった今日の一般的なユーザーモードアプリケーションが利用する機能をサポートしていない。

筆者らは、マルチスレッドや GUI、共有ライブラリの動的リンクといった機能を利用する x86/Linux アプリケーションを異種 OS・アーキテクチャ間で可搬的に利用可能な、JavaVM 上で動作するユーザーモードエミュレータを実装し評価を行った。その結果、本エミュレータは libc や xlib を動的にリンクする x86/Linux 用の GUI プログラムや、マルチスレッドプログラムを Windows 環境において利用することが可能となっている。また、既存の GUI プログラムを Java アプ

[†] 東京農工大学大学院工学府情報工学専攻
Computer and Information Sciences, Tokyo University
of Agriculture and Technology

^{††} 東京農工大学大学院共生科学技術研究院
Institute of Symbiotic Science and Technology, Tokyo
University of Agriculture and Technology

レットとして Web ブラウザ上で実行することを可能とした。このため、ユーザーからの動的な操作を伴うリッチなユーザーインターフェース (UI) を持った Web アプリケーションを構成するための新しい手法を Web アプリケーションの開発者に対し提供する。本エミュレータの性能は実機に対し、演算性能で 2 万倍程度、I/O 性能で 1600 倍程度低速である。これは、まだ完全にエミュレーションを実現できていないものの、GIMP のような I/O を多用するアプリケーションを利用する場合における本エミュレータの実用性を示唆している。

2. 従来のユーザーモードエミュレータ

既存のユーザーモードエミュレータは、エミュレーション可能なアプリケーションの種類や、エミュレータが提供する可搬性において機能が限定される。

QEMU や EM86⁴⁾ は Linux アプリケーションをエミュレーション可能なネイティブコードで実装されたエミュレータである。これらのエミュレータは、エミュレーションするアプリケーションが、Linux 独自のシステムコールを利用する場合に、エミュレータの外部に存在する OS の機能を直接利用するため、エミュレータが動作する環境は Linux 上に限定される。

NestedVM は MIPS-R2000 バイナリをエミュレーション可能な、Java で書かれたエミュレータであり、アプリケーションに対し高い可搬性を提供している。一方で、NestedVM がエミュレーション可能なプログラムは、独自のシステムコール番号を利用する newlib を静的リンクしたバイナリであり、汎用 OS 上で実際に利用されているプログラムのバイナリファイルを直接ロードすることはできない。

VEELS は PowerPC と ARM 用の Linux プログラムをエミュレーション可能な JavaVM 上で動作するエミュレータである。VEELS は Java アプレットとして Web ブラウザ上でネイティブコードアプリケーションを利用することが可能である。VEELS がどのようなシステムコールをサポートし、また共有ライブラリの動的リンクやマルチスレッド機能を利用するプログラムの実行をサポートするかは示されていない。

以上の理由により、既存のユーザーモードエミュレータでは、今日一般的に利用されているアプリケーションプログラムを直接、異種 OS・アーキテクチャ間で可搬的に利用することや、既存のデスクトップ用の GUI プログラムに何らの変更を加えることなく、直接 Web ブラウザ上に配信することはできない。

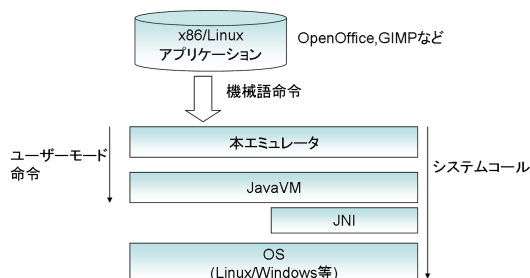


図 1 本エミュレータの構成

Fig. 1 Construction of the emulator

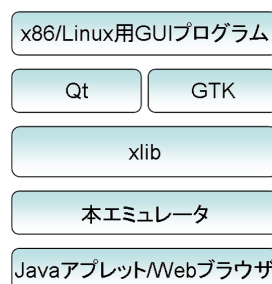


図 2 本エミュレータが提供する UI の構成

Fig. 2 Construction of UI the emulator providing

3. 本エミュレータの目標と概要

筆者らは、本エミュレータの実現に際し、以下を目標とした。

- x86/Linux 用の ELF バイナリをロード可能
- マルチスレッドや GUI、共有ライブラリの動的リンクといった機能をサポートする
- エミュレータ自身が JavaVM 上で動作し可搬性を持つこと
- GUI プログラムが Java アプレットとして Web ブラウザ上で動作すること

図 1 に本エミュレータの構成を示す。本エミュレータは x86/Linux 用のユーザーモードアプリケーションが使用する、ほとんどのユーザーモード命令をエミュレーションする。また、本エミュレータ上では基本的な入出力機能のほか、スレッド、メモリ管理、ネットワークなどに関連する Linux 用のシステムコールを利用可能である。本エミュレータはこれらのシステムコールのうち、POSIX 互換環境で利用可能なものは、Java Native Interface (JNI) を介して実機上で動作する OS の機能を利用する。Linux 独自のシステムコールは、本エミュレータが POSIX 互換システムコールを組み合わせることでエミュレーションする。

本エミュレータは既存の x86/Linux アプリケーション

ンを Java アプレットとして Web ブラウザ上に配信可能である。Web アプリケーションではユーザに対し動的な操作機能をを提供する、リッチな UI を実現するための処理系が普及しつつあるが、多くの処理系は Web アプリケーションのための独自のコーディングを要する。本エミュレータを用いることにより、既存の x86/Linux 用デスクトップアプリケーションを直接 Web ブラウザ上で実行することが可能となる。また、新規にアプリケーションを実装する場合においても、リッチな UI を持つ Web アプリケーションとネイティブコードで構成されるデスクトップアプリケーションを共通の言語、ライブラリによって実現することを可能とする。Web アプリケーションの開発者は本エミュレータを利用することで、図 2 に示すように xlib を基底とする GTK や Qt といった高度なウィジェットツールキットを組み合わせて、動的な UI を構成することが可能となる。

4. 本エミュレータの設計

4.1 本エミュレータがエミュレーションする機能

表 1 に本エミュレータが提供する機能を一覧で示す。本エミュレータ上で、x86/Linux 用のユーザーモードプログラムを可搬的に利用可能である。例として ls, wc といったコマンドや GUI やマルチスレッドを利用するプログラムが挙げられる。本エミュレータは、Linux 以外の POSIX 互換環境上でこうしたプログラムを利用可能とするために、Linux 独自のシステムコールをエミュレーションする。

一方、本エミュレータ上で利用できないプログラムとして、x86 の特権命令を利用する Linux のカーネルモードモジュールや、デーモンとして OS に登録されるプログラム、エミュレータの実行環境に存在しない I/O デバイスやプロトコルスタックへの入出力機能を利用するアプリケーションが挙げられる。デーモンは、動作中の JavaVM プロセスをデーモンプロセスに転化することが困難であるためサポートしない。エミュレータの動作環境に存在しないプロトコルスタックとして、Windows 環境における UNIX ドメインソケットのサポートが挙げられる。このような機能を利用するアプリケーションとして、Apache や KDE のようなシステムに深く組み込まれることを前提としたプログラムが挙げられる。

4.2 本エミュレータのシステムコールインターフェース

本エミュレータが提供するシステムコールのインターフェースは、内部割込みを契機として発動するレ

表 1 本エミュレータがサポートする機能
Table 1 Functions the emulator supporting

本エミュレータがサポートする機能	サポートしない機能・プログラム
x86 のユーザーモード命令	x86 の特権命令
Linux の独自システムコール	Linux のカーネルモードモジュール
GUI	Linux 用のデーモン
マルチスレッド	I/O デバイスのエミュレーション
アプリケーションの Java アプレット化	

イヤに属する。一方、NestedVM のように内部割込みでシステムコールを検出しつつも、アプリケーションの開発者に対し、NestedVM 独自の newlib をリンクすることを強制し、実質的に高級言語のレイヤでシステムコールを提供する立場もある。

本エミュレータは、既存の glibc をリンクした一般的な Linux アプリケーションをエミュレーション可能であり、より実用的なシステムコールインターフェースをアプリケーションに対し提供する。

4.3 命令列の動的解析によるエミュレーション

本エミュレータは、メモリ空間に展開した x86/Linux プログラムの命令列を 1 命令ずつ動的に解析しながら、エミュレーションを行う。一方、NestedVM のように、アプリケーションのバイナリ内に存在する命令列を事前に全て静的解析することにより、エミュレーションコードを Java クラスファイルとして出力する方式もある。本エミュレータでは命令列の動的解析によってエミュレーションする方式を採用した。この方式により、共有ライブラリの動的リンクを行うプログラムの実行をサポートできる。

5. 本エミュレータの実装

5.1 命令セット

本エミュレータは x86 の命令セットのうち、ユーザーモードのプログラムが直接利用するユーザーモード命令のほとんどをエミュレーションする。一方、Linux のカーネルモードで使用される IN, OUT などの特権命令はエミュレーションしない。

また x86 では、ユーザーモード命令のうち浮動小数点演算命令として、FPU, MMX, SSE の各命令セットが定義されているが、本エミュレータでは fadd や fmul といった基本的な演算機能を提供する一部の FPU 命令のみを実装した。これは、gcc を含む今日の一般的なコンパイラが多用することによる。MMX や SSE はコンパイル時に設定を施したアプリケーション

のみが利用するため、本エミュレータではエミュレーションしない。表 2 に命令セットの実装状況を示す。

本エミュレータが実装するユーザーモード命令のうち、内部割込みで用いられる INT 命令以外の全命令は、後述するエミュレータ上のレジスタ、メモリ空間に対する操作として定義する。Java 言語の数値表現形式はビッグエンディアンであり、リトルエンディアンで動作する x86/Linux 環境とは異なる。このため、命令セットエミュレーションでは、演算の入出力先としてメモリ空間が指定された場合、エンディアンの変換を行う。入出力先がレジスタの場合は、エンディアン変換を行わない。

5.2 命令ディスパッチ

本エミュレータでは、x86 命令のデコード結果をもとにエミュレーションする命令を特定し呼び出す命令ディスパッチのために、Java 言語のリフレクション機能のほか、javassist⁸⁾、Java Compiler API⁹⁾を用いた動的コンパイルを用いている。ディスパッチ部分をハードコーディングしなかったのは、将来 threaded-code や super-instruction といった最適化手法を適用することを目指したためである。ただし、これらの最適化手法はまだ組み込んでいない。javassist を用いた命令ディスパッチ性能は、リフレクションに比べ 10%程度、Java Compiler API で 20%程度改善している。動的コンパイルは立ち上がりでのオーバーヘッドが大きいので、頻繁に利用される命令についてのみ用いている。

表 2 本エミュレータがサポートする命令セット
Table 2 Instruction-set the emulator supporting

命令種別	例	備考
整数演算	ADD,SUB, MUL など	実装済み
論理演算	AND,OR,NOT など	実装済み
比較	CMP など	実装済み
シフト/ローテート	SAR,ROL など	実装済み
10 進算術演算	AAA,AAS など	実装済み
制御転送	CALL,RET, JMP など	セグメント間 ジャンプ等は不要
システム	LIDT,STR, ARPL など	ユーザーモードでは 不要
ストリング	MOVS,CMPS など	実装済み
FPU	FLD,FCOMP, FADD など	FPU 制御命令 などが未実装
MMX	MOVQ,PADDB, PAND など	未実装
SSE	MOVAPS,CMPPS など	未実装

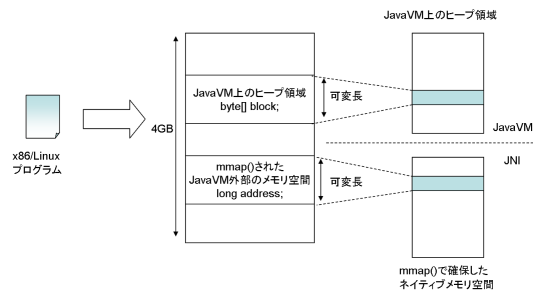


図 3 エミュレータ上のメモリ空間
Fig. 3 Memory space of the emulator

5.3 レジスタ

本エミュレータでは、x86 のレジスタのうち、汎用レジスタのほか、セグメントレジスタや FPU レジスタをエミュレーションする。一方、MMX や SSE レジスタは先に述べた理由により省略した。各レジスタは、Java 言語の long 型変数として確保した。

5.4 メモリ空間

本エミュレータがエミュレーションするメモリ空間は、x86/Linux 環境におけるユーザーモードの仮想アドレス空間に相当する。このメモリ空間は Java 言語の可変長の byte 型配列の集合と、JavaVM 外部のネイティブなメモリ空間からなる。byte 型配列を可変長としたのは、エミュレーションするアプリケーションが mmap システムコールを用いて、任意のファイルディスクリプタとメモリ空間を関連付けた際に、byte 型配列の一部を切り離して、OS が管理するネイティブなメモリ空間に対応付けるためである。

図 3 に示すように本エミュレータは JavaVM 上の byte 型配列として確保した領域と、JavaVM の外部に存在する mmap によって確保したネイティブなメモリ空間からなる 2 種類の異なる領域を集約し、アプリケーションに対し、単一のメモリイメージとして提供する。2 種類のメモリ空間に対するアクセス機能をアプリケーションに対し提供することで、本エミュレータは共有ライブラリの動的リンクをサポートし、また静的リンクされたプログラムに対しては、メモリアクセスに伴う JNI のオーバーヘッドを低減している。

5.5 システムコール

表 3 に本エミュレータ上で利用可能なシステムコールの一覧を示す。アプリケーションが利用するシステムコールのうち、POSIX 互換システムコールは、本エミュレータの外部で動作する OS のシステムコールライブラリを利用する。なお、Windows 環境では Cygwin のライブラリを用いている。set_thread_area などの Linux 独自のシステムコールをアプリケーション

ンが利用する場合、本エミュレータは POSIX 互換システムコールを組み合わせて代替機能を提供する。

Linux 独自のシステムコールは、アプリケーションの作成者が明示的に利用しなくても、glibc がその初期化コード内で多用する。NestedVM はこれを回避するため、エミュレーションする対象のアプリケーションを、glibc ではなく NestedVM 専用に改変した newlib をリンクしたプログラムに限定している。したがって、本エミュレータは glibc をリンクした既存の x86/Linux 用アプリケーションのバイナリを直接ロード可能であり、エミュレータの有用性を高めている。

5.6 マルチスレッド

図 4 にスレッドのエミュレーション方式を示す。本エミュレータではエミュレーションするスレッドを Java のスレッドではなく、OS のネイティブスレッドでエミュレーションする。ネイティブの pthread_create を呼び出しネイティブスレッドを生成する。そのネイティブスレッドをエミュレーションするプログラムのスレッドとする。この仕組みは JNI が提供する AttachCurrentThread によって実現している。したがって、ネイティブスレッドとエミュレーションするスレッドは対応関係にある。

JavaVM がマルチスレッド動作するかは実装依存であり、グリーンスレッドを用いて実装しているものもあるが、今日一般的に利用されている Sun の JavaVM はマルチスレッド動作をサポートしている。

本エミュレータはエミュレータ自身がマルチスレッド動作を行うことで、マルチスレッドアプリケーション

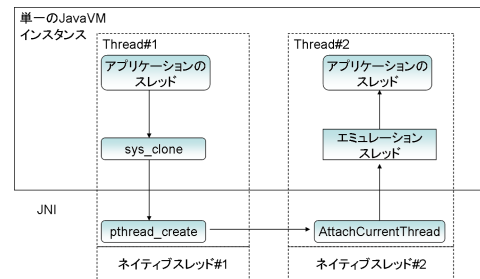


図 4 スレッドのエミュレーション方式

Fig. 4 clone system-call emulation

ンの並列性を再現している。

5.7 共有ライブラリを動的リンクするプログラムのロード

本エミュレータは、Linux の共有ライブラリを利用可能とし、多くの実用アプリケーションがロードできる。Linux 環境における標準的なロードプログラムである ld.so は、単体で実行が可能な静的リンクされた ELF バイナリである。ld.so はアプリケーションの ELF バイナリと、依存する共有ライブラリをロードすることができる。

本エミュレータが共有ライブラリを利用するプログラムをロードする場合、この ld.so を本エミュレータ上で実行することによって対応している。本エミュレータはこの x86/Linux 用のロードプログラムを実行することにより、共有ライブラリのロード部分の実装を大幅に簡略化している。

図 5 に示すように本エミュレータは、アプリケーションが指定したメモリアドレスにあたかも共有ライブラリがロードされているかのように振舞う。本エミュレータは JNI を利用してネイティブな mmap システムコールを呼び出して共有ライブラリをロードするため、アドレス指定が有効なまま mmap を呼び出すと、JavaVM が利用しているメモリ空間を破壊する恐れがある。これを避けるため、本エミュレータは mmap のアドレス指定を解除し、空いているヒープ領域に共有ライブラリをロードする。ネイティブメモリ空間とエミュレーションするメモリ空間の間にはアドレス変換を行うための機能を備えた。こうして、本エミュレータは独自の共有ライブラリローダを実装することなく、既存の ld.so を直接実行することを可能とした。

5.8 GUI とアプレット

Linux 用の GUI プログラムは、基本的に xlib をリンクした X クライアントとして実装されている。X クライアントは実際に描画を行う X サーバーに対しソ

表 3 本エミュレータ上で利用可能なシステムコール
Table 3 The emulator provides follow system-calls

exit	sync	stat	poll
read	kill	fstat	waitpid
write	rename	lstat	select
open	mkdir	ioctl	socket
close	rmdir	gettimeofday	bind
creat	dup	fcntl	connect
link	pipe	set_tid_address	listen
unlink	times	sigaction	accept
chdir	setgid	sigprocmask	send
time	getgid	ugetrlimit	recv
mknod	geteuid	clone	sendto
chmod	getegid	futex	recvfrom
lseek	writew	set_robust_list	getsockname
getpid	readv	nanosleep	getpeername
stime	uname	sched_yield	setsockopt
alarm	mmap	getcwd	
pause	munmap	statfs	
utime	mprotect	fstatfs	
access	brk	getdents	
nice	set_thread_area	lseek	

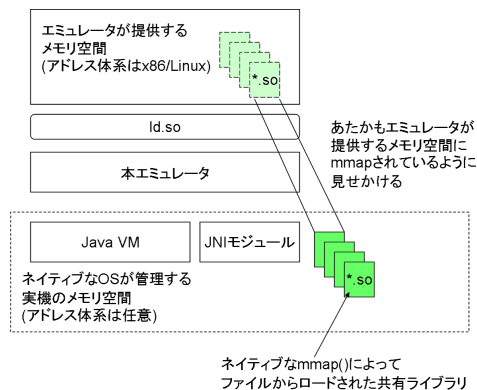


図 5 共有ライブラリのロード機能

Fig. 5 dynamic linking of shared libraries

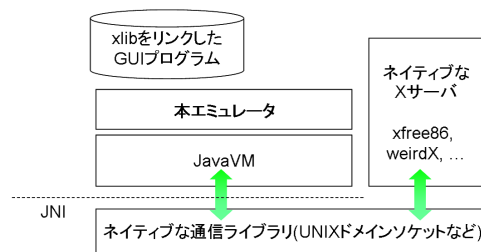


図 6 GUI 機能の構成

Fig. 6 Contribution of GUI

ケット API を用いて描画要求を行う。本エミュレータは、xlib を利用する X クライアントプログラムのエミュレーションのみを行い、X サーバーはエミュレータの外部で動作するものとした。図 6 に本エミュレータ上で GUI を利用する場合の構成を示す。

また、本エミュレータを X サーバーが用意されていない Windows 環境や、Java アプレット上で実行する場合、Java で書かれた X サーバーである WeirdX⁽⁶⁾ と組み合わせることで X 環境を再現している。

6. 本エミュレータの評価

6.1 基本性能

本エミュレータ上での性能が、実機上でプログラムが動作した場合に比べ、どの程度の比率になるかを表 4~8 に示す。対象としたプログラムの処理内容はフィボナッチ、空の for ループ、ADD 命令を線形に配置した繰り返しの無いコード、及び /dev/zero に対する read、/dev/null に対する write である。プログラムの実行に要した CPU のサイクル数をもとに本エミュレータと実機の性能を比較する。

エミュレータ上での繰り返しの無いコードの実行時性能が特に低下する理由として、本エミュレータでは

命令デコード結果をキャッシュして再使用することが挙げられる。

また、read/write システムコールの結果は、本エミュレータの有用性が純粋な演算処理よりも、I/O 処理において発揮されることを示している。

6.2 Linux コマンドのエミュレーション

Linux に標準で添付されている 4 つのコマンドプログラムの実行時性能を測定した。ここでは実機の time コマンドを用いて本エミュレータ上での実行に要した時間と実機上での実行時間を比較した。表 9 に結果を示す。ls は 15 個のファイルをオプションなしで表示した。wc は 112 個のファイルに分散した Java 言語のソースファイル 6 万行程度の行数を取得した。gzip は約 7KB のファイルを圧縮した。dd は /dev/zero から /dev/null に 10MB のデータを読み書きした。

性能低下が見られるが、これらの実際的なプログラムではシステムコールを多用するため、先に示した基本性能と比べ概ね良い傾向を示した。したがって、本エミュレータが提供する性能はより実際的なプログラムの利用において有効だと考えられる。

6.3 GUI とエミュレータのアプレット化

本エミュレータと Java 言語で書かれた X サーバーである WeirdX を組み合わせることにより、x86/Linux 用の GUI プログラムが Java アプレットとして Web ブラウザ上で動作することを可能とした。以下では x86/Linux 用の GUI プログラムを Windows 上で実行したものを示す。

図 7 にはウィンドウを表示するだけの簡易な GUI プログラムの動作画面を示し、図 8,9 にはこれをアプレット化したものを示す。ブラウザの画面内でウィンドウを描画した場合、ウィンドウマネージャの不備により、ウィンドウの拡張といった基本操作は実現できていない。一方、ブラウザの画面外で描画した場合、動作環境のウィンドウ管理システムと連携するため、マルチウィンドウのプログラム等も支障なく操作可能である。

図 10 には簡易なペイント機能を提供する GUI プログラムを、同じくアプレットとして実行したものを示す。実際にアプレット上において、ユーザからのマウス入力をもとに線分を描画可能であることを確認した。

本エミュレータはここに示したように、xlib を直接呼び出す GUI プログラムの実行をサポートしている。また、高機能なウィジェットとして FLTK をリンクしたプログラムがウィンドウの作成までを行えることを確認した。一方、FLTK でウィンドウ内に文字列を描画する場合や、GTK、Qt といったより高度な GUI

表 4 フィボナッチの実行サイクル数
Table 4 Cycle counts of executing fibonacci

n	10	15	20	25	30	35
本エミュレータのサイクル数	4.47 $\times 10^9$	9.982 $\times 10^9$	6.439 $\times 10^{10}$	6.664 $\times 10^{11}$	7.385 $\times 10^{12}$	8.087 $\times 10^{13}$
ネイティブのサイクル数	2.38 $\times 10^5$	3.311 $\times 10^5$	1.112 $\times 10^6$	9.615 $\times 10^6$	1.58 $\times 10^8$	2.088 $\times 10^9$
サイクル比	1.878 $\times 10^4$	3.015 $\times 10^4$	5.792 $\times 10^4$	6.931 $\times 10^4$	4.673 $\times 10^4$	3.874 $\times 10^4$

表 5 for ループの実行サイクル数
Table 5 Cycle counts of executing for-loop

ループ回数	1	10	100	1000	1 万	10 万	100 万	1000 万	1 億
本エミュレータ (サイクル数)	1.645 $\times 10^8$	1.972 $\times 10^8$	3.465 $\times 10^8$	1.854 $\times 10^9$	1.689 $\times 10^{10}$	1.651 $\times 10^{11}$	1.632 $\times 10^{12}$	1.617 $\times 10^{13}$	1.623 $\times 10^{14}$
ネイティブ (サイクル数)	1.656 $\times 10^3$	1.818 $\times 10^3$	3.114 $\times 10^3$	1.562 $\times 10^4$	1.408 $\times 10^5$	1.392 $\times 10^6$	1.204 $\times 10^7$	1.524 $\times 10^8$	1.474 $\times 10^9$
サイクル比	9.931 $\times 10^4$	1.085 $\times 10^5$	1.113 $\times 10^5$	1.188 $\times 10^5$	1.2 $\times 10^5$	1.186 $\times 10^5$	1.355 $\times 10^5$	1.061 $\times 10^5$	1.1 $\times 10^5$

表 6 繰り返しの無いコードの実行サイクル数
Table 6 Cycle counts of executing non-loop code

命令数	50	100	150	200	250	300
本エミュレータ (サイクル数)	2.062×10^8	3.482×10^8	4.940×10^8	6.340×10^8	7.811×10^8	9.291×10^8
ネイティブ (サイクル数)	1.485×10^3	1.629×10^3	1.683×10^3	1.764×10^3	1.845×10^3	2.286×10^3
サイクル比	1.388×10^5	2.138×10^5	2.935×10^5	3.594×10^5	4.234×10^5	4.064×10^5

表 7 ファイル読み込みの実行サイクル数
Table 7 Cycle counts of executing read file

	1024	2048	4096	8192	16384	32768	65536	131072	262144	524288
本エミュレータ (サイクル数)	1.942 $\times 10^{10}$	2.082 $\times 10^{10}$	2.261 $\times 10^{10}$	2.648 $\times 10^{10}$	3.386 $\times 10^{10}$	4.875 $\times 10^{10}$	7.311 $\times 10^{10}$	1.151 $\times 10^{11}$	1.951 $\times 10^{11}$	3.501 $\times 10^{11}$
ネイティブ (サイクル数)	7.659 $\times 10^6$	1.173 $\times 10^7$	1.039 $\times 10^7$	9.302 $\times 10^6$	1.215 $\times 10^7$	1.751 $\times 10^7$	2.354 $\times 10^7$	4.088 $\times 10^7$	6.755 $\times 10^7$	1.3 $\times 10^8$
サイクル比	2.535 $\times 10^3$	1.774 $\times 10^3$	2.175 $\times 10^3$	2.846 $\times 10^3$	2.786 $\times 10^3$	2.784 $\times 10^3$	3.104 $\times 10^3$	2.815 $\times 10^3$	2.89 $\times 10^3$	2.691 $\times 10^3$

機能を利用する場合、本エミュレータは FPU 命令を用いた浮動小数演算を完全にサポートしていないことから、また実現に至っていない。しかし、こうした例外を除けば、本エミュレータはユーザーからのインタラクティブな GUI 処理を Web ブラウザ上においても可能としており、本方式により xlib を基底とする様々な Linux 用の GUI プログラムを Web ブラウザ上で利用可能となることを示唆している。

7. ま と め

筆者らは x86/Linux 用ユーザーモードアプリケーションの異種 OS・アーキテクチャ間における可搬的な利用を目的とした、JavaVM 上で動作するユーザーモードエミュレータを実装し評価を行った。

結果、GUI、共有ライブラリ、マルチスレッド等を使用する x86/Linux 用プログラムを、異種 OS 間に

おいて可搬的に利用可能な機能を実際に有している。また GUI プログラムを Web ブラウザ上でアプレットとして実行することも可能であり、既存の X ウィンドウライブラリを使用する GUI プログラムに変更を加えることなく Web ブラウザ上に配信することを可能とした。

性能面では、x86/Linux の実機の環境に対し、演算性能で 2 万倍から 40 万倍程度、I/O 性能で 1600 倍から 2.8 万倍程度の性能低下が見られた。これは類似研究である NestedVM が MIPS 用プログラムを実機に対し演算性能で 8 倍程度、I/O 性能で 2 倍程度の性能低下で実現しているため、本エミュレータはまだ改善の余地がある。これは、本エミュレータ上で命令ディスパッチのためにリフレクションや Java 言語ソースコードの動的コンパイルを利用することに起因している。

表 8 ファイル書き込みの実行サイクル数
Table 8 Cycle counts of executing write file

	1024	2048	4096	8192	16384	32768	65536	131072	262144	524288
本エミュレータ (サイクル数)	1.642×10^{10}	1.721×10^{10}	1.907×10^{10}	2.353×10^{10}	3.138×10^{10}	4.617×10^{10}	7.341×10^{10}	1.122×10^{11}	1.91×10^{11}	3.421×10^{11}
実機 (サイクル数)	7.901×10^6	1.092×10^7	9.951×10^7	9.996×10^7	9.753×10^7	8.188×10^6	1.19×10^7	1.177×10^7	8.629×10^6	8.046×10^6
サイクル比率	2.078×10^3	1.575×10^3	1.916×10^3	2.354×10^3	3.218×10^3	5.639×10^3	6.165×10^3	9.534×10^3	2.213×10^4	4.252×10^4

表 9 Linux コマンドの実行時間
Table 9 Seconds of executing Linux commands

	ls	wc	gzip	dd
本エミュレータ (s)	5.460×10^1	3.262×10^4	7.736×10^1	5.145×10^1
実機 (s)	5.0×10^{-3}	5.920×10^{-1}	6.000×10^{-3}	2.100×10^{-2}
時間比	1.092×10^4	5.511×10^4	1.289×10^4	2.450×10^3



図 7 元の GUI
プログラム
Fig. 7 desktop

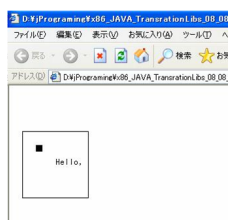


図 8 アプレット化
(ブラウザ内描画)
Fig. 8 In browser

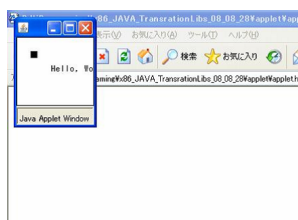


図 9 アプレット化
(ブラウザ外描画)
Fig. 9 Out side of browser

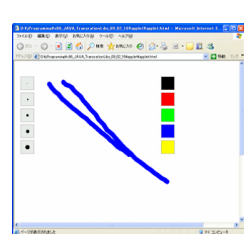


図 10 アプレット化した GUI プログラム
Fig. 10 GUI program runs with applet

今後の課題として、性能面では threaded-code や super-instruction⁵⁾ といった命令ディスパッチの最適化手法をユーザーモードエミュレータの実装に取り入れることや、Java バイトコードの直接生成が挙げられる。機能面では、FPU 命令を含む x86 のユーザーモード命令の実装と、利用可能なシステムコールを追加により、より高度な機能を利用するアプリケーションをサポートすることが挙げられる。

参 考 文 献

- 1) QEMU Project, <http://bellard.org/qemu/>.
- 2) ALLIET, B., AND MEGACZ, "A. Complete translation of unsafe native code to safe byte-code.", In Proceedings of the 2004 Workshop on Interpreters, Virtual Machines and Emulators, pp. 32-41, June.2004.
- 3) A. Yermolovich et al, "Portable Execution of Legacy Binaries on the Java Virtual Machine", ACM International Conference Proceeding Series; Volume 347, pp.63-72, Sep.2008.
- 4) Linux/Alpha project, "EM86: How To Run Linux/x86 Apps on Linux/Alpha", <http://www.alphalinux.org/faq/FAQ-16.html>
- 5) M. Anton Ertl, David Gregg. "Optimizing

Indirect Branch Prediction Accuracy in Virtual Machine Interpreters", ACM SIGPLAN Notices Volume 38, pp.278-288, May.2003.

- 6) JCraft, Inc, "WeirdX - Pure Java X Window System Server under GPL", <http://www.jcraft.com/weirdx/>.
- 7) Vikram Adve, Chris Lattner, Michael Brukman, Anand Shukla, and Brian Gaeke, "LLVA: A Low-level Virtual Instruction Set Architecture", Proc. of the 36th annual ACM/IEEE International Symposium on Microarchitecture, Dec.2003.
- 8) 千葉滋, 他, "Java バイトコード変換による構造リフレクションの実現", 情報処理学会論文誌 42(11), pp.2752-2760, Nov. 2001.
- 9) John Zukowski, "Java 6 platform Revealed", pp155-169, SpringerLink, November. 2006.