社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－ARC－171 (6)
2007－EMB－ 3 (6)
2007／1／22

組込み機器向け仮想 Linux 環境の構築

稗田 諭士[†] 朝倉 義晴[†] 千嶋 博[‡] 佐藤 直樹[†]

[†] NEC システムプラットフォーム研究所
[‡] NEC 研究企画部

概要：組込み Linux に任意のプログラムをインストールすることで、組込み機器の利便性が向上するものの、その反面組込み機器の信頼性が低下するという問題がある。特に任意のプログラムにより、Linux カーネルのセキュリティホールを突かれてしまうと、ビルトインのプログラム実行環境も含めてシステム全体が停止してしまう可能性がある。そこで本論文では、UML(User Mode Linux)を用いた仮想 Linux 環境を構築し、任意のプログラムは仮想 Linux 環境上で実行させることで、ビルトインのプログラム実行環境を保護することを提案する。更に UML の小型化・高速化の改良を図ったので、その手法と評価結果を述べる。

Virtual Linux Environment for Embedded Systems

Satoshi Hieda[†] Yoshiharu Asakura[†] Hiroshi Chishima[‡] Naoki Sato[†]

[†] System Platforms Research Laboratories, NEC Corporation
[‡] Research Planning Division, NEC Corporation

Abstract: To install any Linux programs in embedded Linux improves extensibility of embedded systems. However, it possibly causes the decrease of reliability of the embedded systems. Especially, a certain installed program can stop the whole embedded system by attacking a security hall of Linux kernel. To resolve this problem, we propose virtual Linux environment with UML (User Mode Linux) in this paper. We protect the built-in program environment in the embedded system by making the installed programs run on virtual Linux environment. Moreover, we describe how we improve ROM/RAM usage and the performance of UML to run UML on the embedded systems.

1 はじめに

近年、情報家電を中心に Linux[1]ベースの組込み機器が増加している。これらの機器において、機器出荷時から組み込まれているビルトインプログラムの実行環境(ビルトインプログラム実行環境)を開放し、任意の人が作成したプログラム(任意プログラム)をインストールさせることにより、以下の利点が考えられる。

- ユーザのカスタマイズ性・ユーザビリティが向上する。
- Java VM 上で動作する Java アプリケーションと比較し、高速な実行が可能である。
- ビルトインプログラムと連携が容易である。

しかしその反面、以下のような問題も生じる。

P1) 機器内のファイルへ不正アクセスされる可能性がある(情報流出、ファイル改竄など)。
P2) 機器内のハードウェア資源を不正使用される可能性がある。
P3) 機器のセキュリティホールを突かれる可能性がある。
P4) 機器内部のディレクトリ構成が複雑で、プログラムを追加するのが難しい。

このうち P1)については、SELinux[1]、AppArmor[2]などのセキュア OS が提供するアクセス制御機能を用いることで解決できる。例えば任意プログラムについては、機器内の機密情報にアクセスできないようアクセスポリシを設定

[1] Linux は Linus Torvalds 氏の商標または登録商標である。その他会社名、製品名は各社の登録商標または商標である。

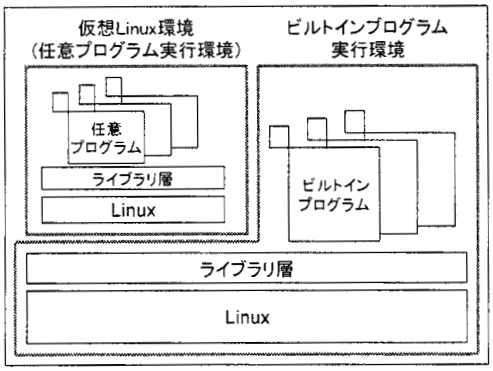

図 1 プログラム実行環境の分離

することができる。また P2)についても、リソース制御機能[3]を用いることで解決できる。

しかし P3)については、アクセス制御機能でもリソース制御機能でも防ぐことができない。例えば Linux カーネルにセキュリティホールがあり、任意プログラムによる 0-Day 攻撃を受けると、システム全体が停止してしまう可能性がある。

また P4)に関しても、現状多くの組込み機器は、各機器でプログラム実行環境が作り込まれており、こうした環境を一般に公開するのは、セキュリティの観点で望ましくない。またプログラム開発者の開発効率を下げることにもなるため、任意プログラムを実行するための独自のプログラム実行環境を用意することが望ましい。

そこで本論文では、P3)、P4)を解決するために、組込み Linux 上に UML(User Mode Linux)[3]ベースの仮想 Linux 環境を導入する。そして任意プログラムを、仮想 Linux 環境上で実行することで、ビルトインプログラム実行環境と分離する(図 1 参照)。このような構成をとることで、P3)のセキュリティホール攻撃に関して、被害の範囲を仮想 Linux 環境内に局所化し、ビルトインプログラム実行環境を保護することができる。また P4)についても、仮想 Linux 環境上に、ビルトインプログラム実行環境よりシンプルな実行環境を用意することが可能である。

2 関連研究

仮想化関連の技術では、近年 x86 プロセッサにおいて、Intel VT や AMD Pacifica など、システムの完全仮想化をサポートする技術が導入されている。またソフトウェアでも、Xen[4]、VMware[5]、UML、QEMU[6]などの仮想化ソフトウェアが存在している。また組込み機器向けの仮想化技術ということでは、[7]、[8]などの研究がなされている。

一方、組込み機器向け仮想化環境に求められる機能要件は以下の通りである。

R1) ビルトインプログラム実行環境と共存できる。

R2) 組込み機器で多く使用されている ARM プロセッサで動作可能。

R3) 任意プログラムを安全に実行できる。

R4) 実行パフォーマンスが良い。

R5) 消費リソース量が少ない(ROM、RAM)。

R6) 将来性。

筆者らは、前述した仮想化ソフトウェアの利点・問題点を分析し、これらの機能要件を満たす最適な組込み機器向け仮想化環境を検討した。その結果、UML を用いた仮想化環境の構築を目指すこととした。

3 UML

UML は Jeff Dike 氏によって開発されたオープンソースの仮想化ソフトウェアであり、ホスト Linux 上で動作する 1 つのプログラムとして、Linux カーネルを実行することができる。また Linux2.6 からは kernel.org の Linux ソースツリーにマージされており、仮想 Linux 環境として注目を集めている。

UML におけるプロセス実行の基本的な考え方は、UML 上で動作するプロセス(以降、ゲストプロセスと表記する)がユーザ空間走行中はホストカーネル管理下で動作し、カーネル空間の処理を UML カーネルで代行するという考え方である。

この代行処理の実現方式は以下の 2 種類がある。

- tt(tracing thread)方式
- skas(separate kernel address space)方式

skas 方式はホストカーネルへのパッチ適用を

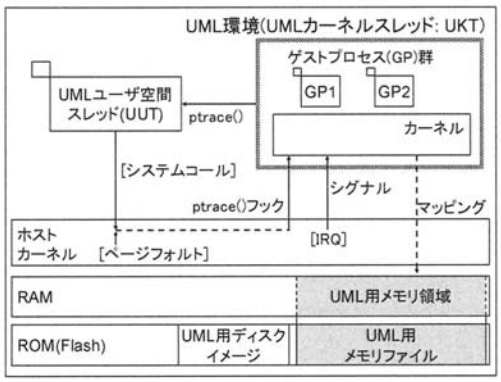

図 2 UML アーキテクチャ図

必要とするが、セキュリティ面、実行パフォーマンス面ともに tt 方式より優れている。そこで今回の組込み機器向け UML の実現においても、skas 方式を対象とする。

3.1 プロセス構成

UML は主に UML カーネルスレッド(UKT)と UML ユーザ空間スレッド(UUT)から構成される(図 2 参照)。UKT は、ゲストプロセスにカーネル機能を提供するスレッドであり、Linux カーネルの提供する機能の大部分を内包している。一方 UUT は、ゲストプロセスのユーザ空間処理を代行するスレッドで、UML 内でコンテキストスイッチが起こるたびに、新たに Active になったゲストプロセスが、UUT として設定される。

3.2 カーネル機能の提供

UKT は、標準の Linux カーネル同様、割込みや例外を受信すると起床して処理を行う。UKT は割込み・例外受信のために、ptrace()とシグナルハンドラを使用している。

このうち ptrace()は、システムコールやページフォルトなど、UUT の動作に起因する割込み・例外を受信するために使用される。UKT は、UUT を ptrace()で監視しており、UUT がシステムコールを発行した時やページフォルトを発生した時に、フックすることができる。例として、UUT がシステムコールを発行した時の処理を以下に示す。

1. UUT がシステムコールを発行すると、UKT が起床し、UUT 停止時点でのレジスタ値を取得する。
2. 取得したレジスタ値のうち、システムコール番号を__NR_getpid(getpid()のシステムコール番号) に書き換え[2]、UUT に実行させる。
3. UUT がホストカーネル上で getpid()を実行する。その後再び UKT が起床する。
4. UUT の本来のレジスタ値をもとに、システムコールを UKT 内で実行する。
5. UKT が UUT のレジスタ値を書き戻した後、UKT は待ち状態に遷移し、UUT のユーザ空間処理に復帰する。

またシグナルハンドラは、UML における IRQ として利用されている。UKT は、あらかじめ IRQ 種別ごとにシグナルハンドラを登録しており、ホストカーネルから IRQ 相当のシグナルを受信すると、シグナルハンドラを介して IRQ の処理が行われる。

3.3 UML 用メモリファイル

UKT は、起動時にパラメータで指定されたサイズの UML 用メモリファイルを作成し、自らの仮想アドレス空間にマッピングする。この UML 用メモリファイルは、UML における物理メモリとして機能し、UML 環境内でメモリ割当てが必要な時は、UKT はマッピングした UML 用メモリファイルからメモリ領域を確保する。

4 組込み機器向け UML

UML は、2 章で挙げた組込み機器向け仮想化環境の機能要件のうち R1)、R3)、R6)の要件を満たしている。しかし R2)を満たすためは、UML が元々x86 プロセッサ向けに開発された仮想化ソフトウェアということもあり、ARM 適用のための改造が必要である。また R4)、R5)を満たすために、組込み機器向けに最適化する必要がある。本章では、これらの機能要件を満たすために行った改造内容について説明する。

[2] getpid()は、本来のシステムコールの代替システムコールとして、ホストカーネルに対して実行される。

4.1 基本ソフトウェア仕様

まず UML の動作環境として、PDA(SHARP Zaurus SL-C3100)の上で、ホストカーネルと同じ Linux カーネルを UML 化して動かすことを目標とした。これは、任意プログラムに対して、ビルトインプログラムと同一のカーネル機能を提供するためである。なお改造前のベースとして、UML カーネルとホストカーネルに以下のパッチを適用した。

UML カーネル

- ベースカーネル: Zaurus SL-C3100 用 Linux カーネル (2.4.20 ベース)
- パッチ(1): UML 化パッチ
- パッチ(2): SYSEMU パッチ

ホストカーネル

- ベースカーネル: Zaurus SL-C3100 用 Linux カーネル (2.4.20 ベース)
- パッチ(1): skas パッチ
- パッチ(2): SYSEMU パッチ

ここで UML 化パッチは通常の Linux カーネルを UML 化するためのパッチであり、skas パッチはホストカーネルを skas 方式で動かすために必要なパッチである。また SYSEMU パッチは ptrace()の機能拡張用パッチであり、ホストカーネルに対するシステムコールを無効化することができる。つまり、3.2 章で示したシステムコールの処理手順のうち、2 と 3 を実行する必要がなくなり、UUT が発行したシステムコールの処理を高速化することができる。

4.2 ARM 適用

R2)、すなわち ARM プロセッサ上で UML を実行するためには、以下の改造が必要である。

- システムコール番号取得ルーチンの改造
- UML で扱うレジスタ構成の ARM 対応
- シグナル受信処理の ARM 対応
- IP パケットの WORD アライメント修正

ここでは、「システムコール番号取得ルーチンの改造」について詳しく説明する。

UML では、ゲストプロセスがホストカーネルに対してシステムコールを発行した時に、その処理を代行するため、システムコール番号とパラメータ値を取得する必要がある。x86 プロセッサでは、システムコール番号とパラメータ値は、レジスタに格納されているため、ptrace() の PTRACE_GETREGS リクエストを用いることで、これらの値を取得できる。しかし ARM プロセッサでは、システムコール番号がレジスタに格納されないため、別の手法でシステムコール番号を取得する必要がある。そこで、以下のような手法でシステムコール番号を取得するようにした(図 3 参照)。

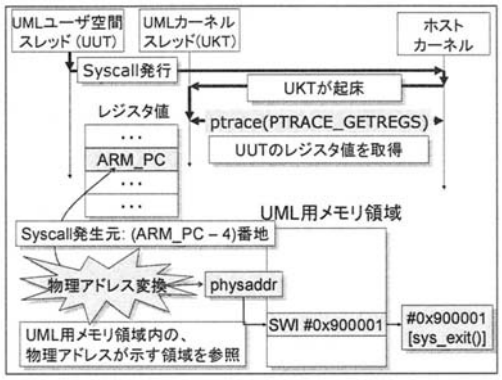

図 3　システムコール番号取得ルーチン

1. UUT がシステムコールを発行すると、UKT が起床し、UUT 停止時点でのレジスタ値を取得する。
2. 取得した PC(プログラムカウンタ)レジスタの値をもとに、システムコール発生元の仮想アドレスを求める。
3. 仮想アドレスを物理アドレスに変換する。
4. 求まった物理アドレスが示す領域を、UML 用メモリ領域内で探索し、4 バイトのデータを取得する。これによりシステムコール発生元の SWI 命令を取得できる。
5. SWI 命令データをマスクし、システムコール番号を取得する。

4.3 実行パフォーマンスの性能改善

R3)、すなわち実行パフォーマンスの性能改善のために、以下に示す改造を行った。

- システムコール番号取得ルーチンの最適化
- 起動時間・シャットダウン時間の短縮

このうち「システムコール番号取得ルーチンの最適化」について詳しく説明する。

4.2 章で述べたように、UML を ARM プロセッサ上で実行するためには、独自のシステムコール番号取得ルーチンが必要となる。このルーチンは、既存の Linux カーネルが提供する機能を組み合わせることで、実現することができる。しかし用いる Linux カーネルの機能には、SWI 命令を取得するためには不要な処理も含まれていることが分かった。そこで組み合わせた Linux カーネル機能を見直し、UML が SWI 命令を取得するために最適化した。これにより、システムコール番号取得ルーチンの性能改善、更には UML 全体の性能改善を図った。

4.4 消費リソースの削減

組込み機器は、ROM/RAM の搭載量が制限されているものが多い。つまり組込み機器の少ない ROM/RAM 量でも UML が動作するように、ROM/RAM 量を削減する必要がある。そこで以下に示す改造を行った。

- UML カーネルのコンフィグ最適化
- ディスクイメージ軽量化
- UML 用メモリファイルの削除
- UML 用メモリ領域を 3MB まで削減

ここでは、「UML 用メモリファイルの削除」について詳しく説明する。

3.3 章で述べたように、UKT は起動時にパラメータで指定された値に基づき、フラッシュ ROM 上に UML 用メモリファイルを作成していた。しかし調査の結果、UKT から UML 用メモリファイルに書き込んでも、実際にはホストカーネルのページキャッシュに書き込まれているだけで、ファイルアクセスは発生していないことが分かった[3]。そこで UML 用メモリファイルを作成しないよう、以下のような改造を行った。

1. 0 バイトの UML 用メモリファイルを作成。
2. 作成した UML 用メモリファイルを UKT の仮想アドレス空間に mmap()する。
3. UML 用メモリファイルを close()せずに削除する。なお削除しても、UML 用メモリファイルのファイルディスクリプタは有効のままである。
4. UML 用メモリファイルのファイルディスクリプタを指定して、ファイルの先頭からパラメータで指定されたサイズ分離れた領域に"0"を書き込む。

このような手順を踏むことで、UML 用メモリファイルを作成することなく、UML カーネルが起動・動作することを確認した。

5 評価

以上述べてきた組込み機器向け UML を、PDA 上で動作させ、性能評価を行った。

5.1 評価環境

評価環境は以下の通りである。

ハードウェア環境

- PDA: SHARP 製 Zaurus SL-C3100
- CPU: Intel XScale PXA270 416MHz
- RAM: 64MB

ソフトウェア環境

- ホストカーネル: SL-C3100 用 Linux 2.4.20 (4.1 章のパッチ適用済)
- UML カーネル: 同上(4.1 章のパッチ適用済)
- X 環境: ホスト OS 上で Xfbdev(XFree86 4.3.0 ベース)を実行。UML 上の X クライアントも、ホスト OS 上の Xfbdev で描画。

5.2 評価項目

ROM/RAM 使用量

UML は以下の用途で ROM を使用する。

- UML カーネルのバイナリ(UKB)
- UML 用ディスクイメージ(UDI)

そこで、上記 ROM 使用量を計測した。また

[3] 本来 mmap()されたファイルへの実際の更新は、msync()もしくは munmap()といった API が呼ばれた時であるが、UKT はそれらの関数を呼んでいない。

表 1 ROM/RAM 使用量

ROM			RAM
UKB	UDI	合計	
1,358KB	512KB	1,870KB	3.0MB

表 2 xengine の性能評価

	ホスト	UML	ホスト比
回転数(RPM)	754.4	667.9	0.89

表 3 ブラウザの性能評価

	ホスト	UML	ホスト比
BT(ms)	659	3415	5.2
CT(ms)	720	2510	3.5
ST(ms)	1741	1737	1.0

RAM 使用量として、UML を起動させるために最低限必要な UML 用メモリ領域のサイズを計測した(表 1 参照)。

評価結果として、ROM 使用量については、ホスト OS における ROM 使用量(約 37MB)と比較して十分小さい値にまで削減できた。しかし RAM 使用量については、UML はホスト OS 上の 1 プログラムであるものの、一般的な組込み機器向けプログラムと比べて大きい。そのため、今後更なる小型化が必要と思われる。

UML 上プログラムの性能評価

まず描画性能を比較するため、X 描画のベンチマークツールである xengine をホスト OS 上と UML 上で実行し、性能比較を行った(表 2 参照)。

また一般的なプログラムの性能評価として、ブラウザの性能評価を行った。ここでは、

- ブラウザの起動時間(BT)
- コンテンツ取得・描画時間(CT)
- スクロール時間(ST)

の各項目について、ブラウザをホスト OS 上と UML 上で実行し、性能比較を行った(表 3 参照)。

なおこれらの性能評価時に、UML 用メモリ領域(RAM 使用量)は 6MB とした。

評価結果として、描画性能に関しては、UML で約 1 割のオーバーヘッドがあるものの、十分実用に耐えうることが分かった。しかしブラウザに関しては、CT と BT に大きなオーバーヘッドが発生することが分かった。

こうした性能オーバーヘッドの原因として、UML 上のプログラムがシステムコールやページフォルトを発生した時に、UML 内で行う複雑な処理(3.2 章参照)などが考えられる。

6 結論

本論文では、組込み機器向け仮想 Linux 環境ということで、UML を用いた仮想 Linux 環境を提案し、性能評価を行った。

この評価結果をもとに、今後は UML 上プログラムの性能改善を中心に、更なる改良を図っていく予定である。現在のところ、UML におけるシステムコール・ページフォルト処理ルーチンの見直しなどを検討している。

参考文献

[1] Peter Loscocco et al., Integrating Flexible Support for Security Policies into the Linux Operating System, In Proc of the FREENIX Track: 2001 USENIX Annual Technical Conference, pp.29-42, June 2001.

[2] AppArmor, http://www.novell.com/linux/security/apparmor/.

[3] Jeff Dike, A user-mode port of the Linux kernel, In Proc. of the 4th Annual Linux Showcase and Conference, pp.63-72, Oct 2000.

[4] Paul Barham et al., Xen and the art of virtualization, In Proc. of the nineteenth ACM symposium on Operating systems principles, pp.164-177, 2003.

[5] VMware, http://www.vmware.com/.

[6] QEMU, http://fabrice.bellard.free.fr/qemu/.

[7] Yuki Kinebuchi et al., Virtualization Techniques for Embedded Systems, RTCSA 2006, Aug 2006.

[8] Hideaki Koshimae et al., Using a Processor Emulator on a Microkernel-based Operating System, RTCSA 2006, Aug 2006.