Linux のシステムコール削減とカーネルサイズの縮小

徐 振宇[1] 井上 純平[1] 須崎 有康[2]

1. はじめに

近年，インターネットに全てのものが繋がる技術 (IoT) が普及してきているが，セキュリティの問題が度々ニュースに現れている．IoT に使われているオペレティングシステム (OS) には組み込みで実績のある Linux カーネルをベースとしたものが多い[1)] が，汎用設定で使われることが多く，脆弱性に繋がる機能も多い．このため，IoT 技術を安全に利用するには，必要なカーネルレベルの機能だけを提供すべきである．まだ，IoT では物理的デバイスの要請からカーネルサイズの縮小も要求される．

本研究では，Linux カーネルの Tinyfication[2)] とシステムコールの削除をメインとして，取り組み，その現状を報告する．

2. 方針と問題点

カーネルの最適化を行うために三つの段階に分けて，研究を行った．①まずに最小限に動くカーネルを作り，②ユーザランドの全てのバイナリで使われるシステムコールを静的解析から調べ，③使わないシステムコールを削った Linux カーネルの作成を検討した．

2.1. Linux の Tinyfication

本研究では，Linux カーネル Tinyfication の機能を用いて，必要最小限なカーネルを作る．このため，カーネルをビルドする時に make tinyconfig 命令を用いて，最小限度なカーネルをビルドする．しかし，この段階のカーネルは起動できない．最低限動くカーネルを作るため，make menuconfig する時に手動で最低限なオプションを追加する必要がある．

1 電気通信大学
The University of Electro-Communications

2 産業技術総合研究所
Advanced Industrial Science and Technology

2.2. システムコール情報の収集

ユーザランドの全てのバイナリで使われるシステムコールを調べるためには動的解析が静的解析の手法を使う必要がある．

簡単な動的解析は ptrace を使って OS を起動させることで取得することができるが，バイナリ内の全ての実行パスを調べたことにならず，Code Coverage が 100%であることを保証できない．GCC では gcov オプションを使うとこで，どのパスが実行されたかを解析して Code Coverage を確認するとこはできるが，100%にすることは容易ではない．

静的解析ではコードそのものを扱うために Code Coverage が 100%にすることは可能であるが，システムコールの呼び出しにはレジスタに番号が入れられるため，全て網羅していることの保証が難しい．Tim Bird は ARM32 に限定してではあるが，objdump で逆アッセンプルして svc 命令を特定し，その命令に到達する前に設定されるレジスタの値からシステムコールを抽出する方法を提案している．残念ならこれはバイナリが static link されていなければならず，また，1 バス解析で完全でないことが明記されている．

2.3. システムコールの削除

Linux カーネルからシステムコールを削除する方法には幾つかのアプローチがある．例えば，直接

表1 Tinyfication の結果

	.config (行数)	**vmlinuz** (MB)	**initrd** (MB)	**合計** (MB)	**割合** (%)
通常カーネル	4392	6.7	35	41.7	100
モジュールなし	4278	36	7.7	43.7	104.8
tiny カーネル	817	3.3	7.7	11.0	26.4

システムコールのテーブルから特定なシステムコールを消すこともできる．しかし，このような方法ではカーネルにそのシステムコールのコードがまだ存在している．システムコールの削除としては，GCC の LTO(Link Time Optimzation) 対応のバッチを Linux カーネルソースに当てて削除する方法[3]を提案している．Daniel Sangorrin は kernel config に CONFIG_XXX_SYSCALL を指定して ifdef 文で削除行わせる方法[4] を提案されている．まだ，Nicolas Pitre は ld -gc-sections を使い，リンカーに code elmination を行わせる方法[5] を提案している．

3. 実験方法及び結果

本研究では，最適化を実現するため，それぞれの部分に分けて，実験を行った．

3.1. Tinyfication

本研究では，Linux-4.4.1 のカーネルを用いて，通常のビルド，カーネルモジュールなしのビルド，Tinyfication のビルド三つの場合を分けて，カーネルをビルドし，Tinyfication の評価を行った．

表1が示したように，カーネルのコンフィグファイルの中に有効な行数を比較し，カーネルのサイズが通常のカーネルのサイズの 26%になった．しかし，今回で作成したカーネルは，NIC のデバイスドライバを入れていてもインターネットの機能が使えなかった．インターネットを使うための機能として何が必要がは今後の調査対象である．

3.2. システムコール情報の収集

ユーザランド側で動作中のシステムコールの情報を取る実験では，static linux のディストリビューションである sta.li linux[6] で全てのバイナリ静的解析を試した．対象 CPU は ARM で libc は musl であり，システムコールの静的解析は Tim Bird[7] の方法を改良した．Tim Bird の方式は完全ではないが，Basic Block に対して適応すること，また解析不能な部分は手で解析することで少なくとも svn 命令に到達する 1 バスでのレジスタの値からシステムコールを特定した．

3.3. システムコールの削除

Nicolas Pitre の方法は，リンカーがリンクする時に，使わないシステムコールのコードを消す．つまり，特定したシステムコールをシステムコールテーブルから隠すことにより，使わないコードにさせ，最終的にリンク時に消す．

しかし，最終的にこの方法を用いて変更後のカーネルを削除したはずのシステムコールの呼び出しができるため，システムコールの削除が失敗しているようである．その理由はまだ調査中である．

4. 関連研究

動いているカーネルからシステムコールの情報を取るにおいて，ftrace を用いる方法もある．ftrace はカーネルの性能を測るや debug 分析するなどによく用いられるツールである．

5. まとめ及び今後の課題

本研究では，カーネルの最適化を実現するため，三つの段階に分けて，調査や実験を行った．その結果，カーネルの Tinyfication と システムコールの情報を取ることができた．今後の課題は，システムコール削除する方法を試し，この三つの段階の仕事を一つのツールとして提供することである．

我々は本研究とは別に Docker イメージに最適なカーネルをつけてリモートマシンで起動する技術である BMC: Bare Metal Container[8] を開発している．将来的は本研究で開発したシステムコールを静的解析で特定する技術とシステムコールを削除する技術を適用して，より安全，且つ高速に実行できるようにしたい．

参 考 文 献

1. ERIC BROWN, “*Embedded Linux Keeps Growing Amid IoT Disruption, Says Study”,* THE LINUX FOUNDATION
2. “*Linux Kernel Tinyfication*”, https://tiny.wiki.kernel.org
3. Tim Bird, *“Advanced Size Optimization of the Linux Kernel”*, LinuxCon Japan, 2013
4. Daniel Sangorrin, *“Kernel security hacking for the Internet of Things”*, LinuxCon Japan 2015
5. Nicolas Pitre, “*Reducing the ARM Linux kernel size without losing your mind*”, Linano connect 2015
6. “*static linux*”, http://sta.li
7. “*Find Syscalls*”, https://github.com/tbird20d/auto-reduce/blob/master/programs/find-syscalls.py
8. K.Suzaki, H.Koie and R.Takano, *“Bare-Metal Container”*, High Performance Computing and Communication (HPCC), 2015