特定プログラムのための汎用OSのサイズ縮小*

2L−3

森若 和雄　九州大学 システム情報科学府†
中西 恒夫　奈良先端科学技術大学院大学 情報科学研究科‡
福田 晃　九州大学 システム情報科学研究院§

1 概要

工期の短縮や実績のあるミドルウェアを利用するために組み込みシステムに汎用OSを使うことが増えている[1]。組み込みシステムでは事前にそのシステムで動作するユーザアプリケーションを限定することができる場合が多い。

本研究では、このような場合に汎用OSのうちアプリケーションから利用されないAPIの実装を削除することでサイズ縮小をはかる。

本稿では、サイズ縮小化の単位として、文または関数の2つの場合について考察する。その結果、関数を単位としてサイズ縮小化する手法を提案する。さらに、提案手法をMINIXに適用し、その効果について考察する。

2 OSのサイズ縮小

以下の部分でのサイズ縮小を考える。

- ソースコードの削除
  ソースコード中の一部分、たとえば `i=i+1;` という一文や、関数を削除する。削除できるのは、その命令文が実行されない場合か、実行されても入出力、制御、削除されていない命令で参照されている変数の全てに影響を与えない場合に限られる。そのほかにデバイスドライバなどではタイミングに関する制約などもある。
- 自動変数の削除
  ある自動変数を参照している命令文が全て削除できる場合、その変数の宣言と、その変数の変更だけをしている命令文を削除できる。
- グローバル変数の削除
  あるグローバル変数について、それを参照する全ての命令文が削除できる場合、その変数の宣言と、その変数だけの変更を行っている命令を削除することができる。また、配列の一部しか参照しない場合には、参照されないことをユーザに示してOS改造時の情報とすることもできるだろう。

削除の単位を文にした場合と関数にした場合を考える。

2.1 ソフトウェアスライシングを利用した文単位のコード削減

ソフトウェアスライシングを利用した専用化について考える。ソフトウェアスライシングとはソフトウェアのデータ依存関係と、制御依存関係を利用して、プログラム中のある変数や入出力に関係する部分をとりだす技術である[2]。

プログラムスライシングでは、プログラムを静的に解析することで、プログラム内のある命令の実行に影響を与える可能性のあるすべての命令を抽出したり、ある命令を変更した場合に影響を受ける可能性がある命令の集合を求めることができる。

また、プログラムを動的に解析して、プログラム内のある命令の実行に影響を与えた可能性のあるすべての命令を抽出することができる。

プログラムスライシングの特徴として、スライシングによって求められたスライスは、ある基準(変数や入出力)について、元のプログラムと等価であり、実行可能である点がある[3]。

2.2 関数呼び出しグラフを利用した関数単位のコード削減

関数呼び出しグラフを利用した専用化について考える。関数の呼び出し関係グラフを作る。

---

*Size reduction of Multi-Purpose Operating System for Specific Programs

†Kazuo Moriwaka, Graduate School of Information Science and Electrical Engineering, Kyushu University
‡Tsuneo Nakanishi, raduate School of Information Science, Nara Institute of Science and Technology
§Graduate School of Information Science and Electrical Engineering, Kyushu University

表 1: 各プログラムを動作させるために必要なコード量

モジュール	全体	APIなし	init	wc	kermit
カーネル (デバイスドライバ以外)	3150	1031	2228	2224	2345
メモリマネージャ(MM)	1788	296	1611	1351	1743
ファイルサーバ (FS)	4437	1389	3536	3299	3797
計	9375	2716	7375	6874	7885
(コード全体に対する割合)	(100%)	(29.0%)	(78.7%)	(73.3%)	(84.1%)

このグラフ上で、ユーザプログラムから利用されるAPIに対応する関数を起点として、到達可能な関数だけを残し、到達不可能なものは削除する。これによって、不要な関数を削除できる。

3 試行

グローバル変数の更新がソースコードの全体に散在している場合や、システムコールによって実行される命令のほとんどが他のシステムコールと共用されている場合には、提案手法はほとんど役に立たないことが予想される。そこでMINIXを利用して、どの程度のサイズ縮小が見込めるか、関数を単位としたコード削除の試行を行ない、確認した。

3.1 MINIX

MINIXは、PC上で動作するマイクロカーネル構成でデザインされた、UNIX cloneである [4]。本評価には最新版である2.0.2のソースコードを利用した。

MINIX上で、init,wc,kermitを動作させるために必要なシステムコールを実装した場合に必要となるコードサイズを、kernel、mm、fsについてみた。また、比較の為にAPIを一つも提供しなくても、起動や初期化、割り込み処理のために必要となるコード量も調べた。

3.2 試行の手順・結果

関数単位のコード削除でどの程度コードが削減できるかをためすには、以下の情報が必要となる。

- OSのモジュール中の関数の呼び出し関係グラフ
- アプリケーションが利用するAPI情報
- APIと、OS内の関数対応表
- OS内の各関数のサイズ (行数)

以下の手順で作業を行った。

1. 関数の呼び出し関係グラフの生成には、cflow* を利用し、メッセージパッシング、関数ポインタを利用した呼び出しを一部人手で追加する。
2. Python† を利用して、呼び出し関係グラフの操作、マーク付けなどを行うプログラムを作った。
3. アプリケーションが利用するAPIの抜き出しは、上の2つを利用して、関数 `sys_call` を呼び出す関数群を取得する。
4. グラフ上の (アプリケーションから利用されている) APIに対応する関数にマークを付け、そこから到達可能な関数にマークをつけることで、利用される関数の一覧を作った。

結果は表1のようになった。

この対象アプリケーションに限れば、行数にして約15%〜20%程度のサイズ縮小ができている。実際にUNIXとして利用する時には、initとユーザアプリケーションが最低限必要なので、MINIXに関して、関数単位でのコード削除を行う場合は20%程度が限界といえる。

4 おわりに

今後の予定としては、Linux、NetBSDなどのよく利用されているOSを対象にどの程度サイズの縮小が見込めるかの試行を行う。文を単位とした手法と関数を単位とした手法を組み合わせて、さらに小さなサイズを目指す。

謝辞

本研究の一部は科学技術振興事業団戦略的基礎研究推進事業 (CREST) の支援のもとに行われたものである。

参考文献

[1] 堀内岳人, 加納護: 組み込みOSとしてのBSD ,BSDマガジン, Vol.6, pp.17–23 (2000).

[2] M. Weiser.: Program slicing., In Proceeding of the Fifth International Conference on Software Engineering, pp. 439–449 (1981).

[3] 下村隆夫: プログラムスライシング技術と応用, 共立出版株式会社 (1995).

[4] Tanenbaum, A.: The MINIX Home Page (1996), `http://www.cs.vu.nl/~ast/minix.html`.

*`ftp://sunsite.unc.edu:/pub/Linux/devel/lang/C/cflow-2.0.tar.gz`

†`http://www.python.org`