# Embedded Linux size reduction techniques

## 自己紹介

- Michael Opdenacker
- free electronsの創業者で組み込みLinuxのエンジニア
- 組み込みLinuxの起動時間に関する技術に興味がある

## カーネル/システムのサイズを減らす理由

- IoTデバイスで実行可
- 起動も早くなる
- 消費電力も減る
- システム全体をキャッシュとRAMで実行
- セキュリティとしても攻撃面が減る

## 今回話す理由

- ELCE 2015からサイズに関するセッションはなかった
- カーネルを小さくするためのプロジェクトの進展がない
- 自分が今までやってこなかった方法
- カーネルサイズに興味がある人をもう一度集めるいい機会

## Linux システムはどれくらい小規模なのか

- RAM
  - カーネルに2~6MB
  - ユーザ空間に8~16MB
  - より多くのRAMはパフォーマンスを向上
- Storage
  - カーネルに2~4MB
  - ユーザ空間は数百KBくらいに収まる
  - 複雑でないユーザ空間(アプリケーション)なら8~16MBあれば十分動作する

## コンパイラの最適化

- gccを使うときはオプション -Os をつけるとバイナリサイズを最小化できる

## 最近のコンパイラを使う

- 汎用ARMのLinux4.10をコンパイルする？/ARM上で動作するLinux4.10上でコンパイルする？
  - gcc 4.7 →407512 bytes (zImage)
  - **gcc 6.2 →405968 bytes (zImage) -0.4%**
    - zImage: カーネルイメージを圧縮したファイル,ブートイメージファイル

## gcc LTO の最適化を使う

- LTO: コードをリンクするときに不要なコードを排除する
- ソースファイルが一つしかないときにもやってくれる
- gcc 6.2 for x86_64
    - LTO未利用：2122624 bytes (unstripped), 1964432 bytes (stripped)
    - **LTO利用：2064480 bytes (unstripped, -2.7%), 1915016 bytes (stripped, -2.6%)**
- gcc 6.2 for armelhf
    - LTO未利用： 1157588 bytes (unstripped), 1018972 bytes (stripped)
    - **LTO利用：1118480 bytes (unstripped, -3.4%), 990248 bytes (stripped, -2.8%)**
        - stripped: LTO以外の最適化を有効
        - unstripped: LTO以外の最適化を無効
- x86_64とarmelhfとの比較ではない．64bitシステムは32bitシステムよりもサイズは大きくなる

## gcc vs clang

- gcc 6.2.0 on x86_64
    - 1964432 bytes
    - with LTO: 1964432 bytes (-2.7%)
- clang 3.8.1 on x86_64
    - 1865592 bytes (-5%)
- gccはhelloworld.cのような小さいプログラムの時に優良

## ARM: arm 命令セット vs thumb 命令セット

- ARM 32bitアーキテクチャでは，arm 命令セット(32bit長)とthumb 命令セット(16bit長)のどちらかを使うことができる
- arm modeでコンパル: 1323860 bytes
- **thumb modeでコンパイル: 1233716 bytes (-6.8%)**
- thumb命令はコンパクトだが，より多くの命令が必要
- 今回行ったテストArm modeでは，ArmとThumbのコードを組み合わせたバイナリが作成された

## 小さいカーネルを入手するには

- make tinyconfigを実行する
- カーネルに必要な最低限の設定
- ハードウェアとシステム独自の機能をサポートするためには，設定を追加する必要がある

## Linux kernelサイズ

- バージョンアップの度にサイズが増加するのが普通だが，tinyconfigを使うとあまり増えない
- 4.4~4.5の間では，ドライバのディレクトリが削除されたことが理由と予想される
- tinyconfigを使った実験ではvmlinuxファイルを減らすことができなかった
- カーネルは実行時にも割り当てを行うので，カーネルサイズの減少だけが重要ではない

### 起動するシステム上のカーネルサイズ

- QEMUでARM Versatile Platform Boardをエミュレートして Linux4.10を起動
  サイズは1MBちょっと
- このカーネルがブートできる最小限のRAMは4MB(3MBのRAMは低かった)

### カーネルサイズを小さくするプロジェクトの現状

- カーネルのパラメータの管理がすでに困難であるため，構成設定を変更するだけでのサイズ減少は不可能
- 近いうちに，使われていない部分は削除されるかもしれない
- メインライン(コミュニティ開発のカーネル)を開発するボランティアが不足してるのでなんとかしたい

## gcc LTO と Linux kernel

- Andi Kleenが2012年に提供したパッチ
- 必要ないコードを削除しサイズを減少，パフォーマンスの改善を可能にした ARMでは-6%もサイズ減少
- LTOは新しいバグが発生したり，バグを調べにくくする．Linusはその時点でツールチェイン(システムプログラムを政策するためのプログラムの集合体)を信用していなかった

## Kernel XIP

- XIP: execution in place
- カーネルテキストをflashし続けることを許可する
- かなり小さいRAMを使ったシステムを動かせる

### カーネルを小さくするための手助け

- ptraceや再起動なしでLinuxコンパイルできる
- コンパイルログを参考に必要かどうかをチェックする
- カーネルを小さくできる部分や大きい部分を見つける
- Bloat-O-Meterを使ってサイズの大きな関数を見つける

## LLVM Linux project

- clangを使ったコンパイルではパフォーマンスやサイズの最適化ができる
- gcc LTOよりもいい
- 2015年から進展がない

- Bernhadrd Rosenkranzerがパッチを更新，近いうちupstreamにpushされるはず

## ユーザ空間 BusyBox vs Toybox

ARMでgcc5.4を用いてコンパイル

- Toyboxはサイズやrootfsが小さいという点で良い
- BusyBoxは構成の自由性・精巧なニーズに対する機能性が良い
- ROB Lamdleyからのコメント：Toybox shellはexperimentalであるが，bashに取って代わる存在である
- 他の小さいシェルにmkshがある

## glibc vs uclibc vs musl

glibc，uclibc，muslは全て標準Cライブラリ 静的・動的・静的で小さいプログラムなBusyBoxでコンパイル ↓ **muslはコンパイルして作成されたバイナリファイルがかなり小さい**

## super stripを使う

sstripはプログラムの実行に不必要なELFを削除する
ELF:コンピューターにおける実行ファイルや共有ライブラリ及びコンパイラの出力するオブジェクトファイルの形式

- 数百バイトから数千バイトの削減
- sstripはアーキテクチャに依存せず，コンパイルが簡単

## ライブラリの最適化

- mklibs
  - 与えられた実行ファイルのセットに2つ分かれているライブラリをコピーするだけ
  - ライブラリから不必要な部分を削除する必要がある

## ファイルシステムのサイズを小さくするためには

- 小さいシステムのためには，initramfsで起動するのが最適解である．より早く起動することができる
- より大きなサイズの場合は，圧縮ファイルシステムを使うといい

## 結論

- 最近のコミュニティ開発のカーネルにはサイズを小さくする動きはなかったが，カーネルサイズを小さくすることは可能である．
- コンパイラは clang か gcc LTOを使う
- C ライブラリにはmuslを使うと良い
- シンプルなコマンドラインユーティリティで十分な時はToyboxを使って見ると良い
- 改善の余地はまだあるが，カーネルパラメータを増やしたり，テストしたりすることなく不必要な部分を削除するのは難しい．