3C-1

Ｌｉｎｕｘを適用した組み込み機器のブート時間短縮

安井 啓介† 上床 克樹† 島田 智文†

株式会社 東芝†

1. はじめに

近年、民生機器分野における Linux の活用が盛んである。デジタルテレビ、映像録画再生機器及び携帯電話などでは、ソフトウェアに対する機能要求は増大の一途にあり、その規模や複雑度も急速に増えている。そこで、これら要求に、より容易に対応可能なソフトウェアプラットホームが望まれている。ここに Linux を採用する例が増えてきている。

Linux を採用するにあたっての懸案事項の１つとして、一般の組み込み機器向けリアルタイムＯＳと比較してブート時間がより多くかかる問題がある。

一方、ＣＥ機器への Linux 適用を目的に活動を続けている、CE Linux Forum（ＣＥＬＦ）にてブート時間の短縮や、コードサイズ縮小の検討が行われている。２００４年６月にこれらの手法が実装されたカーネルのソースコード CELF 1.0 がＣＥＬＦから公開されている。

ここでは、ＣＥＬＦでまとめられている Linux システムにおけるブート時間短縮とサイズ縮小の手法を紹介する。また、実際に Linux を適用したＡＶ機器におけるブート時間短縮の効果を示し、ブート時間への影響を評価し、課題を整理する。

2. ブート時間の評価方法

Linux システムにおけるブート時間は次のように４つのセクションに分けることができる。

ａ）システムの初期化
ｂ）プログラムイメージのロード
ｃ）カーネル起動からユーザモードまで
ｄ）ユーザモードから操作可能まで

ａおよびｂは一般にブートローダに関わる処理であり、ブートローダの持つ測定コマンドなどを利用する。ＣＰＵ内蔵の高精度タイマを使い時間を測定する。ｃは Linux のカーネルの起動時間にあたり、**printk-times** や **KFI** を利用して時間を測定できる。ｄはユーザモードで動作するアプリケーションやシェルスクリプトなどの動作時間であり、ＬＴＴや **clock_gettime システムコール**や **gprof** を利用して時間を測定できる。

3. ブート時間短縮手法

ここでは、ＣＥＬＦでまとめられているブート時間短縮の手法を紹介する。

Preset-LPJ Linux は起動時に 1tick のループ回数を示す loops_per_jiffies という値を計算するために calibrate_dalay ルーチンで数 100ms 程度を費やす。この値は、組み込みシステムではほぼ一定になるために、固定値にしても問題ない。
Kernel-XIP NOR Flash のようにメモリ空間にリニアに割り当てられている領域に Linux カーネルのコード領域を置き、直接実行することで、イメージコピー時間が短縮される。逆に命令実行速度は犠牲になる。
Application-XIP NOR Flash のようにメモリ空間にリニアに割り当てられている領域にアプリケーションプログラムのコード領域を置き、直接実行する。逆に命令実行速度は犠牲になる。
PreLinking 共有ライブラリのリンクをアプリケーション実行時ではなく、前もって実行しておくことで、アプリケーションの起動時間を高速化する。
RTC No Sync リアルタイムクロックとシステムクロックの同期を起動時に行わない。
Disable Console デバッグコンソール（普通はシリアル）への出力を行わない。カーネル起動のコマンドラインで quiet を指定する。
Optimize RCScript RCScript の記述を簡略化する。Busybox(ルートファイルシステムに必要なコマンド群)のシェル内部コマンドを増やす。RCScript をバイナリプログラムとして実行する。RCScript の処理を並列実行するなど多様。
DMA Copy Kernel などのイメージのコピーに DMA を利用する。

Boot time reduction of Linux based embedded consumer electronics products.
† Embedded System Platform Development Department, CORE-TECHNOLOGY CENTER, TOSHIBA CORPORATION

4. サイズの削減

Linux ではアプリケーションプログラムの他にカーネルとファイルシステムが必要であり、イメージサイズが大きくなる。イメージの不揮発メモリから、RAM へのコピーなどに時間がかかるため、ブート時間にはイメージのサイズも大きく影響してくる。ＣＥＬＦで検討されているサイズ縮小手法を簡単に紹介する。

Linux Tiny 軽量化カーネル
squashfs 圧縮率の高いファイルシステム
XIP NOR 上のコードを実行し RAM 使用量を削減
uClibc glibc よりも軽量なライブラリ
ARM Thumb ARM ならば 16bit 命令を使える
gcc の-0s オプション サイズ優先の最適化

5. ブート時間短縮事例

本章では、AV 機器に適用したブート時間短縮手法とその効果を紹介する。

この機器は図１のようなハードウェアシステム構成を持つ。CPU と RAM とバスは２つずつあるが不揮発メモリは片方のＣＰＵのバスにのみ接続されている。ブート時は、ブートローダが NAND から RAM1、RAM2 へイメージデータをコピーし、Kernel とアプリケーションが起動する。

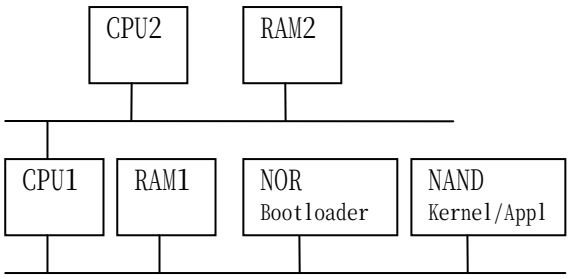

図 1 ハードウェアシステム構成

Preset-LPJ、*Disable Console* はブート時間削減の定石として採用する。

機器としては NAND の容量を減らしコストを下げる意味もあり、カーネルイメージ、アプリケーションイメージは圧縮する。これはコピーサイズを減らすことができるが、解凍時間は逆に増える。全体としては時間短縮になる。

Optimize RCScript の処理は、Busybox で test コマンドなどを内部コマンドとして実行するように修正することで、数 100ms を削減できる。

glibc はサイズが大きいため、ライブラリから未使用のオブジェクトを削除するツール **libopt** を使用することで、ルートファイルシステムのサイズは 2/3 まで削減された。

表 5-1 ルートファイルシステムの縮小

glibc	optmized	rate
1940KB	1300KB	67%(-640KB)

ブートローダのレベルで NAND から RAM へのコピーに DMA を使用することで、DMA の裏で平行して、コピー完了済みの圧縮データの解凍が行える。この並列実行の効果は、24%の削減とかなり大きなものとなった。

表 5-2 コピーと解凍の並列化の効果

直列実行	100%
NAND から RAM へのコピー	45.8%
解凍	54.2%
並列実行	76.2% (-23.8%)

本システムにおける電源投入からアプリケーションプログラムの main 関数までの時間分布を見ると、ほとんどがイメージコピーの時間となった。

表 5-3 起動からアプリの main までの時間分布

システム初期化	4.0%
イメージコピーNAND→RAM	83.1%
Kernel 起動からユーザまで	1.7%
ユーザアプリの main まで	11.3%

6. まとめ

Linux システムであってもブート時間短縮手法を使うことで、カーネルの起動時間は十分に高速であるといえる。Linux 採用により搭載できる機能が増え、アプリケーションプログラムのサイズが増加しブートデバイスから RAM へコピーするイメージサイズが大きくなることが、ブート時間に最も大きな影響を与える。ブートデバイス I/O の高速化（デバイス自身及びソフトウェア処理）、圧縮アルゴリズムの改良による容量削減または解凍時間高速化により、より高速なブートが実現可能と考える。また、アプリケーションがユーザ操作を受け付けるまでの時間も重要なポイントとなってくる。

7. 参考文献

[1] CE Linux Forum HP
http://www.celinuxforum.org/