linux カーネルの H8/300 アーキテクチャへの移植

佐藤嘉則

平成 15 年 10 月 2 日

1　はじめに

ごく小規模な組み込み Linux 環境を実現するために、ルネサステクノロジのオリジナルアーキテクチャ 16/32bit プロセッサである H8/300H CPU および H8S CPU へ linux カーネルの移植を行った。

この成果は 2.5 系列の 2.5.68 でマージされ、また 2.0/2.4 系列は uclinux.org にて維持されている uClinux の一部として配布されている。

H8/300 シリーズは組み込み用途で広く使われており、搭載したボードが低価格で容易に入手できるため、個人の趣味レベルの工作にも多数使用されている。

従来、このクラスの CPU では本格的な OS を使用することはなかったが、linux カーネルが動作することでファイルシステムやネットワーク接続などを利用した高度なアプリケーションを動作させることが可能になる。

本論文では、まずターゲットである H8/300 アーキテクチャの概要と、linux カーネルを動作させる時に制約を受ける部分について説明する。

その後、制約の回避方法などカーネルの実装と移植方法について詳細に説明する。また libc やユーザーランドの移植についてもふれる。

最後に linux-2.6 への対応など今後の展開について簡単に述べる。

2　H8/300 プロセッサ

内部構成が 32bit の CISC プロセッサであるが、組み込み用の小規模な構成であり、一般的な RISC プロセッサに近い命令セットになっている。

直接アクセス可能なメモリ空間は 16Mbyte であり、全てリニアアクセス可能である。レジスタ構成を図 1 に示す。

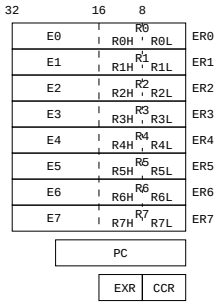

図 1: レジスタ構成

基本的なレジスタ構成は 16bit × 16 個であるが、二つのレジスタを組み合わせて 32bit のレジスタとして使用する事が可能になっており、gcc でもコンパイルオプションを指定することでそのようなコードを生成できる。

gcc のこの機能を使うことで、一部機能が不足しているものの 32bitCPU とみなして linux カーネルを移植することが出来た。

linux カーネルを移植する上で不足している機能は

- MMU がない

  普通の linux カーネルは MMU を使用するので単純に移植できない。

- カーネルモードとユーザーモードの区別がない

  linux カーネルの内部では、カーネルモードとユーザーモードが区別されていることが前提になっている。

- 割り込みベクタが固定されている

  通常は ROM に存在しているため、自分が起動してから書き換えることが出来ない。

などが挙げられる。

3 linux カーネルの移植

3.1 問題点への対応

上で挙げた各問題については、以下のように対応することでカーネルの移植が可能になった。

3.1.1 MMU とメモリ管理

http://www.uclinux.org/にて開発されている uClinux であれば MMU が不要になるので、これを移植することで MMU については必須でなくなる。

幸いにもメモリマップが酷似している m68k にて十分に開発されているため、この実装を流用することでまったく問題なく移植できた。

3.1.2 モードの区別

カーネルモード／ユーザーモードの区別が存在しないため、カーネルスタック／ユーザースタックの区別もない。

しかし、カーネルの内部ではモードが区別されカーネルスタックとユーザースタックが分離していることを前提にしているので、なんらかの方法でモードの切り替えと各スタック領域をを実装する必要がある。H8/300 への移植では、ステータスレジスタのユーザービットをモードの識別に割り当て、例外処理の起動・終了時にモード切替えが発生する場合はソフト的にスタックポインタを書き換えることで、カーネルスタックとユーザースタックを分離している。また、例外処理が起動した時にユーザーモードで実行されていた場合は、例外スタックフレームがユーザースタック側に構築されているので、それをカーネルスタックにコピーしている。

カーネルモードからユーザーモードに戻る場合も、同様にカーネルスタックからユーザースタックへ例外スタックフレームをコピーしている。なお、カーネルスタック→ユーザースタックへのコピーは不要な

場合も多い（通常は例外発生時に作成されたものが生きている）ので、効率を考えると必要な場合のみコピーを行うべきであるが、他のアーキテクチャと同じ動作になっていた方がわかりやすいので、現在の常時コピーする方法をとっている。このとき作成されるスタックフレームは、図 2 のような構造になる。

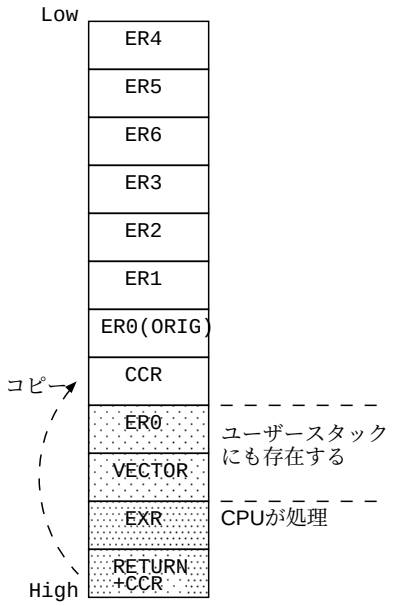

図 2: 例外発生時のスタックフレーム

ユーザースタックに残る部分は、本来存在していないものなので極力減らすようにした。

3.1.3 割り込みベクタ

ベクタテーブルの位置がハード的に固定されており、また通常 ROM に割り当てられる位置に存在するため、カーネルが自力でベクタテーブルを再設定することが出来ない.

ROM にカーネルを書き込む場合は同時にベクタテーブルを作成できるので特に問題にならないが、RAM 上に配置する場合はブートローダーがベクタテーブルを使用するため、なんらかの方法で割り込みベクタをカーネルに向けない限り割込みを受けることが出来ない。

この問題を回避するため、いったん RAM に用意したテーブルにジャンプさせ、そこから本来の割り込み処理を呼び出す方法を取った。

この方法は、RAM 上に用意されたテーブルを書き換えるためにその位置をなんらかの方法で知る必要があるが、ベクタテーブルの構造を解析することで先頭アドレスを求めることが出来る。

当初は静的（リンク時）にアドレスを決定していたが、ブートローダと同期がとれていないとテーブルの書き換えに失敗するので、現在は初期化時に動的に決定するようになっている。

また、カーネル内部の割り込み処理では割り込み要因の番号が必要になるが、この方法の場合 RAM のテーブルから割り込みエントリにサブルーチンコールでジャンプさせることで、スタックに積まれた戻りアドレスから割り込み番号を求めることができる、

この方法を使った場合、各割り込み毎に個別のエントリを設けた場合より若干効率が低下するが、割り込み要因がある程度多い場合はコードサイズをかなり抑えられるので、割り込み要因が 64 もしくは 128 個と比較的多い H8/300 では有効な方法である。

実際のコードで比較した結果

1. 個別のエントリを使用した場合

```
vec_n:
    jmp @interrupt_n

intterrupt_n:
    mov.l er0,@-sp
    sub.l er0,er0
    mov.b #n,r0l
    jmp @common_interrupt_entry
```

コードサイズ　12byte ×割り込み要因
実行時間　　26cycle

2. 共通のエントリを使用した場合

```
vec_n:
    jsr @interrupt_entry

interrupt_entry:
    mov.l er0,@-sp
    mov.l @(4:16,sp),er0
    sub.l #vector_top,er0
    shlr  er0
    shlr  er0
    subs  #1,er0
```

コードサイズ　20byte(共通部分) + 4byte ×割り込み要因
実行時間　　42cycle

この例では共通のエントリを使用した場合コードサイズが 1/3 になるが、現在の実装ではレジスタの退避処理がもっと複雑になっているため、さらに差が大きくなる。また実行時間の低下については、これ以降の処理時間も関係するため厳密に求めていないが、実用上無視できる程度と思われる。

3.2 システムコール

システムコールは他のアーキテクチャに合わせて、システムコール番号と引数を各レジスタにセットした後、trapa 命令でカーネルを呼び出す。具体的な仕様は

トラップ番号　　0
システムコール番号　ER0
システムコール引数　ER1 - ER6 (最大 6 個)

と定義した。

引数が 7 個以上になるとこの手順では対応出来なくなるが、i386 にも同様の問題があるので当面考慮する必要は無いと考える。

3.3 デバイスドライバ

極力既存のドライバを使用できる方向で進めた。

現在 NIC/IDE のドライバが動作しているが、I/O 部分と irq の番号を修正する程度で動作している。この時一部ドライバで割り込みコントローラの制御タイミングの違いにより問題が発生したが、i386 と同等の動作にすることで問題が解決した。まだ、実際に動作させているドライバが少ないため断言出来ないが、ほとんどのドライバは容易に移植できると思われる。

I/O アクセスについてもアドレス変換するためのテーブルを用意して i386 と同じアクセス方法を実現することにより、ドライバ側の修正を不要にすることも検討したが、テーブルのサイズが大きくなる・外部デバイスの標準的な接続方法が規定出来ない等の理由で断念した。なお、内蔵機能のドライバについては基本的に新規開発になるため、カーネル内部でのサポートは一切行っていない。

ただし、汎用 I/O ポートについては、その性格上複数のドライバが共有する事が予想されるが、ハードウエアではそういった状況がまったく考慮されていないと思われるので、利用状況を管理するための機構を H8/300 版固有の機能として追加している。

3.4 ブートローダー

現在、カーネルを RAM に読み込むためのブートローダーとして RedBoot を使用している。

これにより、シリアル・Ethernet を使用してホストマシンからバイナリをダウンロードするか、CompactFlash にあるバイナリを読み込んで起動する事が可能になる。

採用した理由は、ネットワークを利用してバイナリファイルをダウンロードできるためである。RS232C 経由でもダウンロード自体は可能であるが、ターゲット側のシリアルインタフェースが高速転送向きでないため、ダウンロード時間が非常に長くなり現実的ではない。

また、新規に開発するより短時間で用意できるだろうとの目論見もあったが、結果的にはあまり変わらなかった。なお、カーネル自体は特定のブートローダーには依存していないので、容易に独自のブートローダーを作成することも可能である。

現在、gzip で圧縮したバイナリを RAM に展開して起動するブートプログラムが実験的に作成されている。これを利用することで CPU 内蔵の ROM から linux カーネルを起動する事が可能になった。現状では展開に時間がかかり過ぎるので、今後アルゴリズムの見直しなど高速化を行う必要がある。

3.5 libc

通常使われている glibc はサイズなどの問題があったので、uClibc を移植した。

uClibc は MMU-less のカーネルにも正式に対応しておりサイズも小さいため、特に問題もなく移植できた。

この libc は CPU に依存する部分が非常に限定されており、移植性は非常に高い。

4 開発環境

4.1 ターゲットボード

いくつかのターゲットをサポートしており、それぞれの構成は以下のようになっている。

基本的な構成はほぼ同じである。

ホストとのインタフェースは RS232C(CPU 内蔵の UART) と Ethernet を使用することになる。

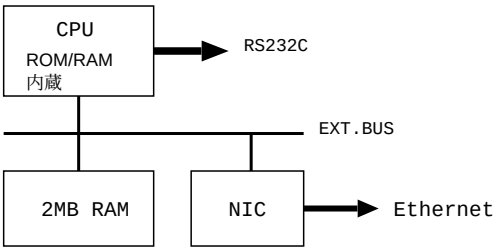

図 3: aki3068net

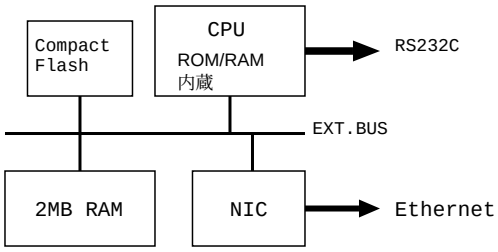

図 4: h8max

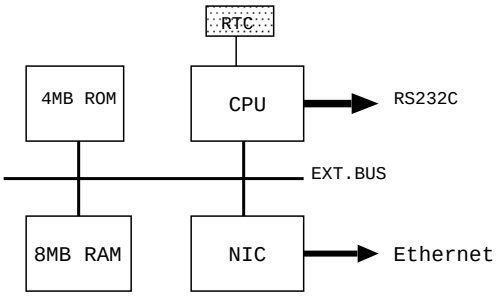

図 5: edosk2674

大量のデータ転送の場合も Ethernet により高速に転送する事が可能なため、カーネルや RAMDISK イメージの転送も短時間で完了し、ICE などの高級な機器を使用しなくても効率的にデバッグ作業を行うことが出来る。

4.2 ホスト環境

GNU/Linux 環境に h8300-elf のクロス開発環境を構築して開発を行っている。

なお、現在使用している toolchain のバージョンは以下のようになっている。

gcc	3.3 + パッチ
binutils	2.13.1
gdb	5.2.1 + パッチ

gcc は他のアーキテクチャと一部互換性がない部分 (long long 型が 32bit になる) を修正するため、gdb は linux カーネルを動かすために不足する機能を追加するためのパッチをあてて使用している。

また、開発用のディレクトリをターゲットから NFS 経由でマウントさせることで、作成したプログラムをすぐにターゲット上で評価することが可能になっている。

4.3 ディストリビューション

最近までまとまったものは存在していなかったが、uclinux.org で配布されている uClinux-dist パッケージにてサポートされたことにより、必要な環境を容易に構築出来るようになった。

このパッケージは、メニュー形式でコンフィグレーションを行うと、指定された構成のカーネルとルートイメージが作成されるという非常に便利な物である。

現在、sash,busybox,vi,boa(web サーバー),telnet などが動作している。

コンフィグレーション中の画面を図 6 に示す。

図 6: コンフィグレーション画面

5　今後の予定

2.4 カーネルについてはバグフィックスとコードの最適化を主に行っている。この成果は随時開発版へマージしている。

開発版については最新版への追従と各デバイスドライバの移植を行っている。デバイスドライバについてはマージが完了していないので、マージ作業を進めて行く必要がある。

このほか、カーネルとは直接関係がないが、ユーザーランドをコンパイルするための toolchain サポートが十分ではないため、この部分についても改良を進めて、より扱いやすい物にしていきたいと考えている。

6　おわりに

2.4 カーネルの移植は IPA の未踏ソフトウエア創造事業の支援を受けて行われた。本年度も継続して支援を受けており、カーネルの開発と toolchain の改良を行っている。

それ以外にも，ターゲットボードの提供や技術的な支援などを多くの人から受けた。

これら支援に感謝して終わります。

参考文献

[1] uClinux-h8 porting project http://uclinux-h8,sourceforge.jp