GNU/Linux on SuperH プロジェクト

g 新部 裕, 小島 一元, 杉岡 利信, 野澤 寿晴, 吉井 卓, 八重樫 剛史

2001 年 9 月 28 日

1 イントロダクション

GNU/Linux on SuperH プロジェクト [1] は、日立 SuperH プロセッサ (SH-3 または SH-4) 上で動作する GNU/Linux システムを実現するために、 1999 年 8 月に g 新部をはじめとするメンバーがスタートしたプロジェクトである。その沿革を 表 1 に示す。

時期		トピック
1998	夏	linux-kernel メーリングリストで話題になる。
1998	冬	杉岡による Linux の移植 (SH-3)。
1999	7	g 新部 GDB を CqREEK SH-3 で動かす
1999	8	g 新部による Linux の移植 (SH-3)。
1999	8	小島による Linux の移植 (SH-4)。
1999	8	吉井による Linux の移植 (SH-3)。
1999	9	Linux 2.3.15 標準 kernel へ。
1999	10	小島による GNU C Library の移植。
1999	11	小島 & g 新部による PIC ABI 定義。
1999	12	小島による PIC の実装。
2000	1	野澤による SH Linux/BSP のパッケージ企画。
2000	3	八重樫 HP680 で動かす。 X Server を移植。
2000	3	国際シンポジウム M17N2000 のスペシャルセッションに。
2000	4	SourceForge で LinuxSH[2] の CVS リポジトリの運用が始まる。
2000	春	g 新部がディストリビューション Chiyonofuji を発表。
2000	夏	八重樫 Dreamcast で動かす。 Debian を移植。
2000	夏	小島 & g 新部による GNU へのマージ。
2000	9	g 新部 Linux Kongress 2000 (ドイツ) で発表。
2000	10	DODES システムの研究開発始まる。
2000	11	Linux Conference 2000 Fall (京都国際会議場) にて発表。
2001	2	秋葉原オープンソースまつりにて発表。
2001	3	国際シンポジウム M17N2001 開催。
2001	4	Dreamcast サポートを取り込む。
2001	5	SH7751 サポートを取り込む。
2001	5	LinuxWorld Expo/Tokyo 2001 .Org Pavilion に出展。
2001	6	オープンソースの集い 2001 (名古屋大学) に参加。
2001	7	LinuxTag 2001(ドイツ) に参加。

表 1: GNU/Linux on SuperH プロジェクトの沿革

本プロジェクトは当初より国際的な協調体制のもとで進められ、カーネル、ユーザランド、ブートローダの開発から GNU Toolchain の拡張、ハードウェアプラットフォームの企画・設計・開発に至るまで、あらゆる分野で成果をあげてきた。

平成 12 年度 (2000 年度) には情報処理振興事業協会による未踏ソフトウェア創造事業に採択され [3]、ネットワーククラスタによって組み込みプラットフォームの効率的な開発・テスト環境の実現を目指す DODES プロジェクト [4] の研究開発を開始した。

このプロジェクトは今年度も継続して行われることが決定した [5]。今年度は、 DODES の公共サービスとしての利用を可能にするために必要なセキュリティ面の機能強化、 SuperH だけでなく MIPS や ARM など他の組み込みアーキテクチャをもサポートするマルチプラットフォーム化を重点的に行う計画である。また、 DODES の有効性を実証するためのクラスタ構築も予定している。

本稿では、まず前回の Linux Conference 2000 Fall から現在までに GNU/Linux on SuperH プロジェクトによって為された新たな成果をカーネル、プラットフォーム、ディストリビューションの各項について説明する (第 2 節)。

続いて、 GNU/Linux on SuperH プロジェクトの今後の主要な研究開発テーマである DODES の技術的解説 (第 3 節)、及び eCos/RedBoot for DODES の紹介 (第 4 節) を行う。

最後に、オープンソースプロジェクトとしての GNU/Linux on SuperH プロジェクトが為しえてきたことの意義について述べ、その未来を展望して結びとする (第 5 節)。

2 GNU/Linux on SuperH プロジェクトの進展

2.1 カーネル

性能改善

性能改善としては、 SH-4 のキャッシュの取り扱いに関する変更 (g 新部、小島) が大きな効果をあげている。キャッシュの取り扱いの API は現在の 2.4 のカーネルは移行期であり、新旧二つの API が混在している。

SH-4 のカーネルは、これまでの古いキャッシュの取り扱いの API (`flush_page_to_ram`) から、新しい 2.5 に向けた API (`flush_dcache_page`) へ対応した。あわせて、ページあたりのキャッシュの扱いをキャッシュラインの特性を考慮に入れて、アセンブラで記述しなおした。

チェックサムルーチンの改善 (g 新部, Siegel) は、 TCP/IP の通信の性能をあげた。

新機能の追加

新機能としては、新しい CPU のサポートが 2 種ある。ST40STB1 サポート (Menefy, McKay) と SH-7751 サポート (Silva, Siegel) である。これにともない、 PCI のサポートが共通化された (McIntire)。

また、不連続メモリサポート (g 新部、八重樫)、 SH7709A における ptrace 対応 (吉井)、 SH-3 における MMU の不具合の回避 (g 新部) が実装された。

新しいデバイスのサポート

SH-4 コンパクトフラッシュ直接接続がサポートされた (阿部)。また、割り込みレジスタ対応 (阿部)、 PC RTC エミュレーション (杉岡)、 SH-4 RTC の不具合の回避 (g 新部) が行なわれた。

SEGA Dreamcast 向けカーネルの開発においては、 2001 年に入ってから新たに Broadband Adapter(Ethernet) と GD-ROM ドライブにおける CD-R のサポートが追加された (BERO,

Comstedt)。これによって、 Dreamcast においても実用的な GNU/Linux システムの構築が可能となった。

2.2 プラットフォーム

2000 年 12 月から 2001 年 7 月の間に、表 2 に示す 10 のプラットフォームがカーネルで新たにサポートされるようになった。まだカーネルには取り込まれていないが、この他にも京都マイクロコンピュータの SH-7751 の評価基板、日立の Handheld PC ペルソナ、 HP の Pocket PC Jornada 548 の対応も行われている。

時期		プラットフォーム	開発者・貢献者
2000	12	EC3104(Compaq Aero 8000)	Philipp Rumpf
2000	12	DMIDA(Industrial PDA)	Mitch Davis, Greg Banks
2001	1	CAT68701[6]	海老原
2001	2	STB40STB1 HARP	Stuart Menefy, David McKay
2001	2	STB40STB1 Overdrive	Stuart Menefy, David McKay
2001	4	SEGA Dreamcast[7]	M. R. Brown, Paul Mundt, 八重樫
2001	5	SolutionEngine 7751	Ian da Silva, Jeremy Siegel
2001	5	BigSur	Dustin McIntire
2001	5	SH2000[8]	杉岡
2001	7	A&D ADX	阿部

表 2: 新しいプラットフォーム

SuperH Linux カーネルでは、 machine vector という仕組みを採用することによってプラットフォーム依存ルーチンを仮想化している (Stuart Menefy)。これにより、単一のバイナリで複数のプラットフォームに対応するジェネリック・カーネルが実現するだけでなく、新しいプラットフォーム (`new_machine`) のサポート追加も、表 3に示すファイルを変更または作成するのみで比較的簡単に行うことができる。

	ファイル	内容
変更	`include/asm-sh/io.h`	`io_new_machine.h` の追加
	`include/asm-sh/machvec.h`	machine vector 定義の追加
	`arch/sh/kernel/Makefile`	オブジェクトファイルの追加
	`arch/sh/config.in`	`CONFIG_SH_NEWMACHINE` の追加
作成	`include/asm-sh/io_new_machine.h`	I/O function の定義
	`arch/sh/kernel/io_new_machine.c`	I/O function の実装
	`arch/sh/kernel/mach_new_machine.c`	machine vector の実装
	`arch/sh/kernel/setup_new_machine.c`	固有の初期化処理
	`new_machine` 固有のデバイスドライバ	

表 3: 新プラットフォームのサポートに必要な作業

2.3 ディストリビューション

Debian GNU/Linux on SuperH

SuperH における Debian ディストリビューションの開発は 2000 年夏より八重樫により行われてきたが、 2001 年に入って Debian プロジェクトのオフィシャルメンテナによる作業が行われるようになった (Oliver M. Bolzer, 石川[9])。

現在では Debian パッケージの生成・アップロードの作業の自動化が実現しており、多くの “sh” アーキテクチャ用パッケージが Debian オフィシャルサイトより利用可能となっている。

Dreamcast Linux Distribution

SEGA Dreamcast 用カーネルの進展と eCos/RedBoot の開発を受け、 2001 年 5 月にはスタンドアロンシステムとして実用的な運用のできる初の Linux ディストリビューションが八重樫により作成された。

このディストリビューションはブートローダとして eCos/RedBoot を組み込み、ユーザランドは Debian をベースに作成されている。 X Window System、アーケードゲームエミュレータ XMAME、 DOOM クローン PrBoom などのソフトウェアを 1 枚の CD の中に収めている。

このディストリビューションは GNU/Linux on SEGA Dreamcast の Web サイト [7] にてダウンロード可能である。またこの CD は LinuxWorld Expo/Tokyo 2001、オープンソースの集い 2001 などの各イベントでデモンストレーションと配布が行われた。

3 DODES プロジェクト

3.1 DODES の紹介

組み込み機器の分野に GNU/Linux システムを利用するためには、研究レベルの現在のシステムを実際の製品に利用する段階のソフトウェアとしてまとめていく必要がある。具体的には、安定して稼働することが確認されたソフトウェアをパッケージとして配布し、稼働実績を広めていくことが必要となる。特に、このパッケージ作成に際して、テストスイートを一通り流して、動作の確認を行なう技術が重要なポイントとなる。

DODES は複数のターゲットマシンを、ホストマシンからネットワークを用いて利用するクロス開発環境のネットワークシステムを構築し、テストスイートをターゲットマシンに分散させてテストを実施する技術を実装するソフトウェアとして開発された。 これはまたクロス環境におけるテストシステムの構築を行ない、膨大なテストスイートを多くのターゲットマシンにばらまいて効率良くテストを実施する技術を実装し、ディストリビューション作成など大規模な作業の効率向上をも目的としている。

DODES のアイデア自体は非常にシンプルなものであるが、いくつかの場合に非常に強力な手段だと思われる。 ここで鍵となるのはターゲットとホストでほぼ同一の GNU/Linux という環境が用意できるという組み込み GNU/Linux システムではごくあたりまえの特性であるということに注意されたい。

なお、本節で解説する DODES の実装は、 DODES プロジェクトの Web ページ [4] よりそのソースコードなどの情報を入手することができる。

3.2 DODES の構成

DODES は

(1) ターゲット計算機の状態表示

(2) 待ち行列処理

(3) 遠隔実行

の 3 つの基本機能によって実現される。

DODES では、ホスト計算機からの遠隔実行要求を受け、ターゲット計算機上でプログラムが稼働する。この際、複数のプログラムの全体の実行時間を短縮させるためには、複数あるターゲット計算機において、負荷を分散させる必要がある。

DODES における負荷分散はプロセスの実行を単位とする、荒い粒度の負荷分散である。またプロセスのマイグレーション (実行途中のプロセスの実行を別のターゲット計算機に移動させること) は行なわず、負荷の分散は、プロセスの投入時にその時点での各ターゲット計算機の負荷の情報をもととして、投入するターゲット計算機を選択することで行なわれる。(1)、(2) の機能はこの選択のために使われる。

DODES の利用目的は、ディストリビューションの作成とそのテストにあるため、投入されるプロセスの数はターゲット計算機の数と比較して多数であり、プロセス間の関連が (特別の場合を除いては) 少ない、独立に稼働するプロセス群であるという性質がある。このため、プロセスの投入時にターゲット計算機を選択する、荒い粒度の負荷分散のサポートで、実用上十分と考えられる。

以下 (1)–(3) の各機能について述べる。

3.2.1 状態表示機能

状態表示機能は、 UDP のクライアントプログラムとして実装され、

(1) 現在のプロセスの負荷の情報を取得する部分

(2) ホスト計算機からのパケットを送る部分

から構成される。(1) の部分では、プロセスの負荷の状態の情報としては、`/proc` ファイルシステムの `/proc/loadavg` を用いる。(2) の部分では `DODES_UDP_PORT` として定義される UDP のポート (現在は、 3557、 DODES の 16 進表記 `0xd0de5` の下 4 桁) でパケットを「3.2.2 待ち行列管理機能」を有するサーバへと送信する。(2) の送信は一方的であり、ホスト計算機の機能のサーバプログラムが稼働していなくても送信は続ける。

状態表示機能は、以下のように動く。

(I) パケットを送信のため、通信の初期化を行なう。

(II) 一定時間おきに以下の動作を行なう:

  (a) プロセスの負荷の状態を `/proc/loadavg` から取得する。

  (b) ホスト計算機の UDP のポート `DODES_UDP_PORT` に向けて送信。

ここで、一定時間とは `DODES_NODE_INTERVAL` で定義される値で、現在は 30 秒としている。

この状態表示機能はプログラム dodes-node として実現され、システムの運用としては、各ターゲット計算機において常時 dodes-node を稼働させておき、常にホスト計算機へ情報を送るようにしておく。

3.2.2 待ち行列管理機能

待ち行列管理機能は、負荷分散のためにプロセスを管理する仕組みを提供する機能である。プロセス実行としてあたかもホスト計算機でプロセスが実行されるかのように見える透過の遠隔実行方

式を採用したため、遠隔実行するプロセス要求を待ち行列として、管理する方式は採用せず、遠隔実行を実行するターゲット計算機の待ち行列という形態を取ることにした。

設計の第一段階ではブロードキャストによって、各ノードの情報を収集するプロトコルであったが、実装にあたって非効率であることが判明し、各ノードが自律的に情報をサーバに送る構成に変更した。

ターゲット計算機の待ち行列は、負荷の軽い順に並んだ待ち行列であり、実行するプロセスは、先頭の、つまり、最も負荷の軽いターゲット計算機に割り当てられるという設計とした。

ホスト計算機では、ターゲット計算機の待ち行列の待ち行列を管理し、ターゲット計算機の状態表示機能によって定期的に送信される情報を受けとり、その待ち行列を更新する。

また遠隔実行機能の要求にしたがって、最も負荷の軽いターゲット計算機を選択し、遠隔実行をそのターゲット計算機で行なうように情報を渡す。負荷の制御として、最も負荷の軽いターゲット計算機の負荷が一定値より高い場合には、(一定時間毎の更新により) その状態から脱するまで返答を返さない。返答でブロック (停止) することとし、投入されるプロセスの最大値が制限される仕組みを持っている。

DODES における待ち行列管理機能は、

(1) UDP のサーバおよび TCP のサーバの両方の機能を有するプログラム

(2) TCP クライアントライブラリ

の二つの要素から構成され、(2) のライブラリは、遠隔実行機能を実装するプログラムで利用される。

待ち行列管理機能のうち (1) のサーバプログラムは、以下のように動く。(2) の TCP クライアントライブラリは、対応する要求を出すライブラリである。

(I) UDP パケットを受信のため、 TCP 接続の受信のために通信の初期化を行なう。

(II) 待ち行列を初期化する。

(III) UDP のパケット、 TCP の接続に応じて以下の動作を行なう:

  (a) UDP のパケットを受けた場合、ターゲット計算機の負荷の情報を更新する。

  (b) TCP の接続を (TCP クライアントライブラリから) 受けた場合、待ち行列の先頭となっている最も負荷の軽いターゲット計算機を選択し、その IP アドレスの情報を返し、負荷の情報を更新する。(ただし、最も負荷の軽いターゲット計算機の負荷がある一定値より大きい場合には返答は保留する。)

ここで、負荷の一定値とは `DODES_MAX_LOAD` で定義される値で、現在は 10 としている。また、UDP のパケットを受けるポートは `DODES_UDP_PORT` として定義され、現在は、3557 番 (DODES の 16 進表記 `0xd0de5` の下 4 桁) である。TCP の接続を受けるポートは `DODES_TCP_PORT` として定義され、現在は、UDP と同じく 3557 番である。

3.2.3 遠隔実行機能

DODES ではホスト計算機上であるプログラムを実行する場合にそれがターゲット計算機のためのものであることを検知し、複数あるターゲット計算機のどれかの上での遠隔実行を行う。 遠隔実行機能自体は UNIX 系オペレーティングシステムでは rexec や ssh で実現されているが、

DODES では遠隔実行結果やその失敗の原因の詳細をホスト計算機側で知る必要があり、これら既存の遠隔実行機能ではこの要求を充分に満たせない。そのため DODES での要求を満たす遠隔実行機能を新たに作成した。

UNIX 系オペレーティングシステムでの実行機能は execve と呼ばれるシステムコールに集約されるため、カーネルまたは共有ライブラリレベルでこの execve の中に遠隔実行機能を実現することで既存のプログラムを再コンパイルすることなく遠隔実行機能を付加することが可能になる。

現実装ではシステムライブラリである GNU libc の execve システムコールのラッパー関数において実行ファイルがターゲット計算機用のものであるかどうかを判定し負荷分散機能の下で選択されたターゲット計算機上で遠隔実行を行う。

遠隔実行機能は、ホスト側スタブプロセス、ターゲット側遠隔実行サーバ、ターゲット側スタブプロセス、ターゲットプロセス、の 4 つのプロセス要素とその間の通信で構成されている (図 1 参照)。

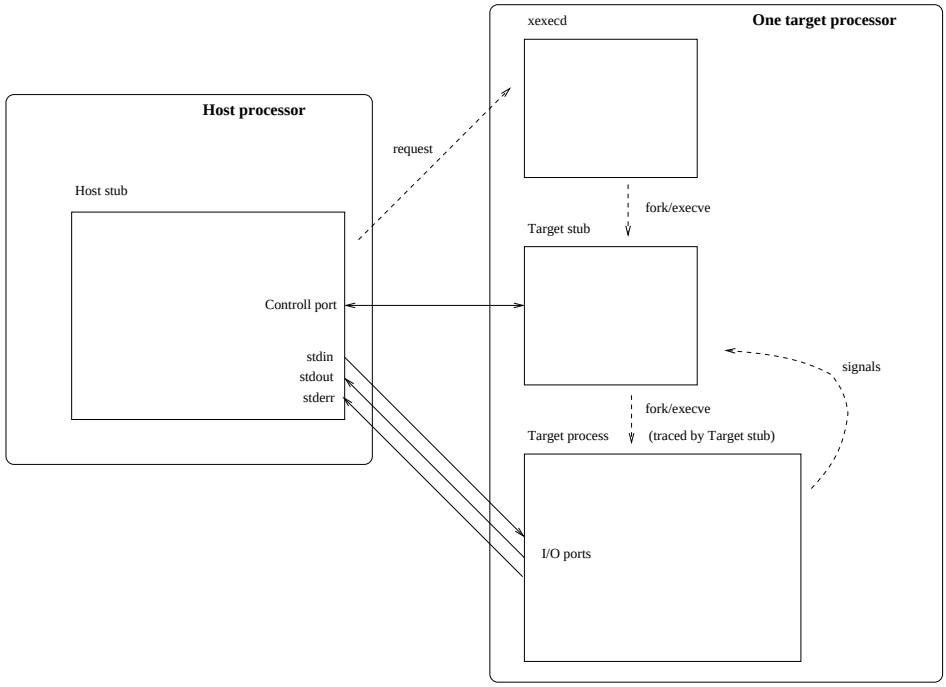

図 1: DODES の遠隔実行

遠隔実行は以下のような順序で行われる。

(I) ホスト側計算機で execve が実行されるとき、その引数として実行可能ファイル名、コマンド引数リスト、環境変数リストが渡される。

(II) 実行可能ファイルがターゲット計算機上で実行すべきものの場合 execve を実行しようとしているホスト側プロセスがホスト側スタブプロセスとなる。このホスト側スタブプロセスはターゲット側遠隔実行サーバに execve の引数として渡されている実行可能ファイルの遠隔実行要求を行う。

(III) ターゲット側遠隔実行サーバはホスト側スタブプロセスからその他の引数およびホスト側スタブプロセスへの標準入出力ポートを取得し、ターゲット側スタブプロセスを生成する。

(IV) ターゲット側スタブプロセスはターゲットプロセスを被トレース状態の子プロセスとして生成する。

(V) はターゲット計算機上で execve を行い実行可能ファイルを実行する。この execve に対する引数は (III) で取得されたものを使う。

(VI) ターゲット側スタブプロセスはターゲットプロセスの終了コード等を取得してホスト側スタブプロセスに渡し、ホスト側スタブプロセスはそれを自分の終了コードとして終了する。

DODES ではそれが可能な限りターゲット用のプログラムがあたかもホスト上で実行されているようにみせかける。 そのためホスト側スタブプロセスの標準入力へのデータを通信を用いてターゲット用プロセスの標準入力に到達させ、逆にターゲット用プロセスの標準出力へのデータをホスト側スタブプロセスの標準出力先へ送り込むようにしている。 この通信はホスト - ターゲット間の TCP/IP 通信で行われる。 (III) でのポートとはこの TCP ポートを表している。

ホスト - ターゲット間の入出力の結合と (VI) によってホスト側ではその実行ファイルが遠隔実行されることをほとんど意識せずにテストなどを行うことができる。 ただし DODES は、 NFS を用いたネットワーク環境によって実現されるためファイルの存在などをチェックするようなテストスクリプトについては、ホスト側とターゲット側でのファイルの存在情報の伝達遅延など NFS 特有の問題に注意する必要がある。

また遠隔実行の現実装では標準入力がコンソールであることを判定したりするプログラムはターゲットでの直接実行と DODES による実行では差がでる場合もある。

3.3 DODES の実行例

もっとも簡単な例として x86 ホストにおいて SH ターゲットのクロス gcc によってターゲット用のプログラムを作成し、それをシェルから実行する場合の例を上げる。

```
dodes:~$ sh4-unknown-linux-gnu-gcc -o helloSH hello.c
dodes:~$ file helloSH
helloSH: ELF 32-bit LSB executable, Hitachi SH, version 1, dynamically linked
 (uses shared libs), not stripped
dodes:~$ ./helloSH
Hello, world
```

ここでホスト dodes は Intel Celeron を CPU とする計算機である。

```
dodes:~$ cat /proc/cpuinfo | grep "model name"
model name      : Pentium III (Coppermine)
```

実用的な例として GNU Compiler Collection (GCC) における C コンパイラのリグレッションテストスクリプトの実行に要した時間を上げる。このテストスクリプトではターゲット用コンパイラにより多数の小さなソースプログラムのコンパイルを行いその約半数についてはコンパイルされた実行可能プログラムのターゲットでの実行を伴う。

GCC のバージョンは 2.97 でテストの総数 13154 個に対しテストの所要時間はホスト Celeron 600MHz 主メモリ 256M byte と ターゲット SH-4 200MHz 主メモリ 64M byte からなる最も単純なシステムで 28 分 20 秒であったのにターゲット計算機のみでのテストには 185 分 13 秒を必要とした。このテストスクリプトの実行時間のほとんどはコンパイル時間に費されるがターゲットが並列化されない DODES でもこのような場合には非常に有利であるといえる。

なお GCC のバージョンは 3.x では C コンパイラのテストの総数が 16000 弱に達しテストの所要時間も 2.97 の 2 倍程度が必要になっている。

他の実行例として面白いものに GNU libc, emacs, Perl, Ruby などのクロス環境での構築がある。 これらターゲットでの実行プログラムを実際に走らせて構築を行うものは通常のクロス環境では完全な構築は非常に困難になる。 DODES ではターゲット上での構築とほとんど変わらない操作で、高速な構築ができる。

3.4 DODES クラスタ

本プロジェクトで現在計画中の DODES クラスタは、図 2 に示すような Ethernet で結ばれた複数の異なるアーキテクチャの計算機群によって構成されるネットワーク環境である。ホストとな

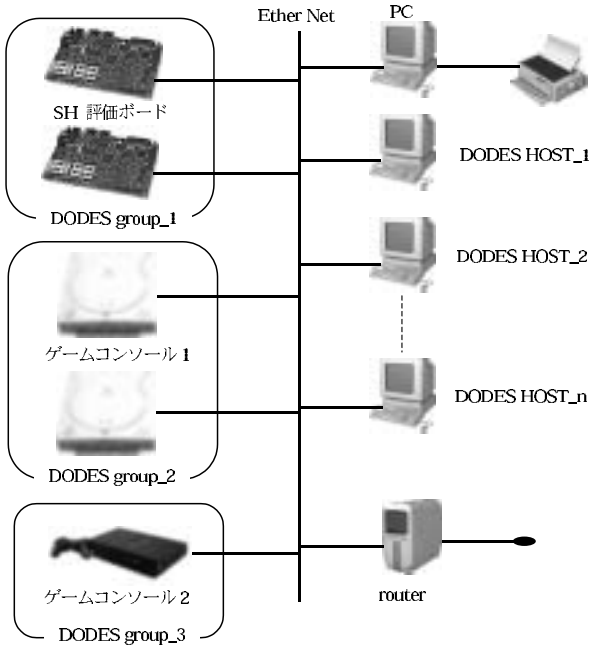

図 2: DODES クラスタ

る計算機では DHCP や TFTP、 NFS などの各種サーバを稼働させ、ターゲットとなる計算機ではネットワークブートかつディスクレスの運用を行う。

この環境においては、クラスタ内に存在するアーキテクチャのプログラムであればどの計算機でも区別なく実行することができるようになる。この環境は、低速なターゲット計算機向けにディストリビューションのような大規模ソフトウェア開発を行う場合などに特に有用であると考えられる。

この他、遠隔ネットワークからの DODES クラスタの利用を可能とするために、利用者の認証を行う機能、遠隔指令によりターゲット計算機の電源の制御やリセットを行う機能を別途開発する予定である。

4 eCos/RedBoot for DODES

当初 SuperH アーキテクチャには標準のカーネルブートローダが存在しなかったため、GNU/Linux on SuperH プロジェクトではカーネル開発後まもなく IPL+g という小さなフリー

ソフトウェアを独自に開発した。

この IPL+g は、カーネルブートローダ、GDB stub によるリモートデバッグ支援機能、カーネルが利用する BIOS ルーチンなどを統合したブートストラップソフトウェアであり、システムのブート ROM などに書き込んで、そのままファームウェアとして用いることができる。

また、IPL+g と組み合わせて用いる ethboot と呼ばれるソフトウェアを etherboot[10] をベースに開発し、SuperH 用 Linux カーネルのネットワークブートを実現した。これにより、カーネルの開発・デバッグの効率が大きく向上した。

しかしながら GNU/Linux on SuperH はそのサポートプラットフォームを増やし続けており、単純な IPL+g がこれらすべてに対応することが次第に困難になってきていることが問題となっている。

このことから本プロジェクトではブートストラップソフトウェアには以下のような要件が必要であると認識するようになった。

- 高度な移植性
- コンフィギュレーションや機能拡張が容易な構造
- 高水準ライブラリサポート (特に診断・デバッグ出力用として)

そこで本プロジェクトでは、このような要件を満たす組み込み用 OS として実績があり、かつオープンソースソフトウェアである Red Hat(旧 Cygnus) の eCos[11] と、eCos 上で動作するブートストラップ・デバッグ環境の RedBoot[12] を採用することにした。

eCos は Hardware Abstruction Layer(HAL) による層構造と徹底したモジュール化構造、独自のコンフィギュレーションツールを採用することによって高度な移植性を実現し、また新しい機能やプラットフォームサポートの追加を容易にしている。

これにより eCos/RedBoot は、異なるアーキテクチャ・プラットーム間で統一された機能とインタフェースを持つブートストラップ・デバッグ環境を実現しており、日々拡張を続ける SuperH Linux カーネルだけでなく、マルチアーキテクチャ・マルチプラットフォーム化を目指した研究開発を行っている DODES をもサポートすることができる。

これまでの本プロジェクトの主な実績として、表 4 に示すプラットフォームに対する HAL やデバイスドライバなどを開発してきた。今後も、新しいプラットフォーム HAL やデバイスドライバ

プラットフォーム	アーキテクチャ	主なサポートデバイス
SEGA Dreamcast	SH/SH-4	Ethernet(BbA)、GD-ROM
HP Jornada 680/690	SH/SH-7709A(SH-3)	
SCEI PlayStation 2	MIPS/EE(R5900)	Ethernet

表 4: eCos/RedBoot for DODES の対応状況

の開発や移植など必要な拡張をこの eCos/RedBoot に行っていく予定である。

拡張された eCos/RedBoot のソースコードは本プロジェクトの CVS リポジトリによって管理・公開されている。さらに、アップストリーム (Red Hat) へのフィードバックを積極的に行い、コードの統合を進めていく方針である。現在 Dreamcast HAL 関連のコードの統合作業が Red Hat において行われており、まもなく正式にリリースされることであろう。

5　まとめ

5.1　プロジェクトの名前

本プロジェクトの名前は、“**GNU/Linux**”と始まる。これは、カーネルだけでなくトータルの OS を目指したということもあるが、カーネルの開発からはじまった後、GCC を始めとする GNU Toolchain の研究開発に多くの努力を集積してきたということが大きい。「カーネルだけではない」ということを示すために、“**GNU**”をつけている。

活動内容が組み込み OS からクラスタ環境までに広がり、SuperH 以外のプロセッサ対応などにも伸びているので名が全く体を表さなくなってきた。なにか良い名前を考えねばなるまい。

5.2　この 2 年の所感

本プロジェクトの 2 年を振り返えると、最初に、「そこに CPU があれば GNU/Linux を動かすのだ。誰もやらないのなら自分自身で。」という気概がとても強かったことを思い出す。その時点では、必要な仕事の量やその効果などはほとんど考えてなく、偏狭な思い込み (良い言葉で言えば信念) で取り組んでいた。この最初の時点で集まったメンバは強い。どんどん手を動かす。

それから、GNU の環境の SuperH に関する未成熟と CPU/OS の ABI の問題が明らかになった。そして本プロジェクトとして ABI の定義を拡張し、GCC の進歩に合わせて開発環境を整備してきた。この経験を通じて、GNU Project の提供するフリーソフトウェアが、まさにソフトウェア開発の土台であり、非常に重要であることがわかり、特に注力してきた。また、OS 全体といった大規模かつ多様なソフトウェアを構築することにより、開発環境の問題が明らかになり鍛えられるという関係があり、一旦回り出すと大きな成果につながる。この循環をつくり出せる人間はとても数が少ない。人の宝。

その後、ある程度の成果が出てから集まってきた人間には、いろいろな人間がいる。「もっと面白いことを!」と展開する人、応用を着実に進める人と広がってきたことは非常に嬉しい。しかし、残念ながら勢力争いをするのみで手は動かさない人なども出てくる。フリーソフトウェアの活動で重要なのは、ソフトウェアを増やしたり、品質を高める方向の活動を継続することである。人が手を動かすことを批判し止めたり、アイデアを進めることにブレーキをかけることなどは、残念。当初のメンバの熱意をもう一度。

5.3　本プロジェクトの意義と未来

本プロジェクトは、フリーソフトウェアのプロジェクトとして発展し、多くのソフトウェアの成果を出してきた。最も重要な点は、フリーソフトウェアを**開発**する実例として、日本においてわかりやすい成果が出しつつ、発展する場を実現したことである。世界のハッカーが集まるプロジェクトを日本から発信し、世界の一極として注目を保ち続けてきている。

Linux に対しては、カーネルをより小さい環境でも動かすという方向性を示し、(小さい方の) スケーラビリティの基軸となっている。携帯情報端末や組み込み機器への Linux の可能性を示し、小さなコンピュータへ Linux を、ハードウェアに関するさまざまな選択肢を持って、「組み込む」ことが夢ではなくなった。

GNU 開発環境に対しては、実際の大きなソフトウェアを対象として使うことで、鍛えに鍛え、バグフィックスと安定性をもたらした。このことは、本プロジェクト以外のソフトウェア開発にも大きく役立ち、GNU 開発環境の組み込み機器のソフトウェアへの適用をさらに広げるであろう。

フリーソフトウェアのハードウェアとのつながりができてきたことも大きな意義である。ブートローダからカーネル、アプリケーションまですべてのソフトウェアがフリーソフトウェアで構成される機器構成が可能となり、いくつか実現されている。これは、情報としての価値、再現性という価値が非常に大きく、ソフトウェアを科学とすることができる一歩かもしれない。その開発環境もすべてフリーソフトウェアで提供されるのである。

今後は、このようなフリーソフトウェアの環境を前提として、それを十分に活用するハードウェアの設計と製品開発ができるようになるだろう。一つの実例が、別のアイデアを生み、フリーソフトウェアを有効に適用できる分野がさらに増えていく。さまざまな開発が互いに互いを触発し、プラスの循環をもたらすように祈りたい。フリーソフトウェアを中心としてその渦ができんことを。

参考文献

[1] GNU/Linux on SuperH Project. http://www.m17n.org/linux-sh/.

[2] LinuxSH Project. http://linuxsh.sourceforge.net/.

[3] 情報処理振興事業協会. 平成 12 年度「未踏ソフトウェア創造事業」の公募について. http://www.ipa.go.jp/NBP/12nendo/12mito/.

[4] DODES Project. http://www.m17n.org/dodes/.

[5] 情報処理振興事業協会. 平成 13 年度「未踏ソフトウェア創造事業」の公募について. http://www.ipa.go.jp/NBP/13nendo/13mito/koubo13.htm.

[6] 有限会社りぬくす工房. CAT68701. http://www.si-linux.com/cat/index.html.

[7] GNU/Linux on SEGA Dreamcast. http://www.m17n.org/linux-sh/dreamcast/.

[8] (株) アイ・ティー・オー. SH2000 HomePage. http://sh2000.sh-linux.org/.

[9] Debian GNU/Linux for SH. http://hanzubon.jp/Linux/Debian/SH/.

[10] Etherboot Project. http://etherboot.sourceforge.net/.

[11] eCos^{TM} Home Page. http://sources.redhat.com/ecos/.

[12] RedBoot^{TM}. http://sources.redhat.com/redboot/.