

# PS2Linuxファン

有限会社メタロジック・福井利夫  
fukui@metallogic.co.jp

## PS2Linuxカーネル用DMAパッチ

待望のPlayStation2 Linux Kit (以後、PS2Linux) が発売され、みなさんさっそくさまざまな使い方をされているようですね。私は、PS2のハードウェアを直接操作してグラフィックで遊びたいと思いました。本稿ではこの目的に都合がいいようにカーネルに手を入れています。

### PS2のアーキテクチャ

最初に、PS2で3Dグラフィックを表示するためのアーキテクチャについて説明します。PS2は嬉しいとよく言われます。確かにプログラミングは単純とはいえませんが、アーキテクチャそのものはすっきりしていると思います。EE Core、VPU1、GS関係を簡単にまとめたブロック図を図1に示しておきます。

まず、PS2において、GSによってグラフィックを表示させるための方法としては、大きく分類すると2つの方法があります。

1つは、EE CoreでVPU0などを用いて透視変換処理まで行って2Dデータを求め、メモリ上に配置し、DMAを用いてGSへ転送する方法です。もう1つの方法は、EE Coreでは3Dの頂点データまたは頂点データを作成するためのヒントとなる情報のみを求めてメモリ上に配置し、後の作業はVPU1で処理をさせるものです。メ

モリからはVPU1へ直接データを転送し、VPU1上ではGSの描画データを生成して直接GSを動作させます。

VPU1を使用した描画、VPU1を使用しない描画、共に共通して使用することができます<sup>\*1</sup>。

EE Coreの実行と、DMAによるVPU1、GSへの転送も並列に動作します<sup>\*2</sup>。通常は、DMAへ転送するデータを格納するメモリをダブルバッファとし、1フレーム<sup>\*3</sup>の描画中にDMA転送を行い、EE Coreでは次のフレームのためのDMA用データを作成します<sup>\*4</sup>。

従って、1フレーム中にはDMA転送とVPU1、GSによる画面描画、EE Coreによる次のフレームのDMA転送のためのデータ作成が並行して行われることになります(図2)。

さて、1フレームの描画ごとにDMA転送されるデータ量は、実に膨大なものになります。PS2のV-RAMはあまり大きくないため、頻繁にテクスチャも入れ替わりますから、1フレーム中の描画でV-RAMの内容は何度も完全に入れ換えられます。すなわち、1フレームを描画するために、DMAは必要なすべてのデータを転送しなければなりません。

しかしながら、堅いモデルの頂点データ、テクスチャなどは各フレームで共通して使用するため、

毎回DMAデータ作成のためにDMAバッファにデータを設定するのは無駄な作業でしょう。

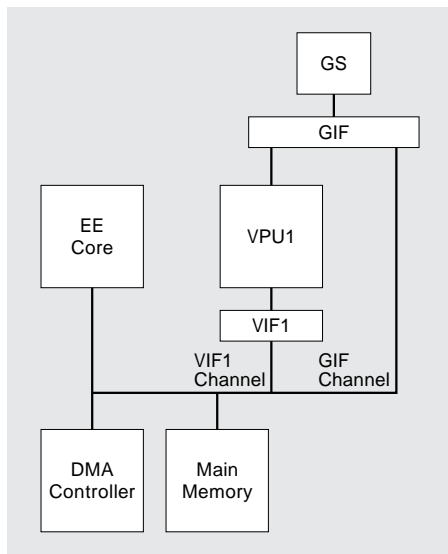
PS2では、このような問題の解決のために、DMAのモードとしてChain Modeが用意されています。中でもSource Chain Modeは、転送するデータの最初に必要なタグ(Tag)を置くことで、転送する情報を制御することができます。DMAのタグの種類には、表1のような種類がありますので、さまざまな要求をうまく解決できます。

PS2では、このDMAのタグの考え方は随所に盛り込まれており、GSやVPU1へのパケットにも必ずタグが付きまゝ。タグはそれぞれ、GIF、VIFが解釈し、データのデコードや転送先の選択といった、細かい扱いを制御します。

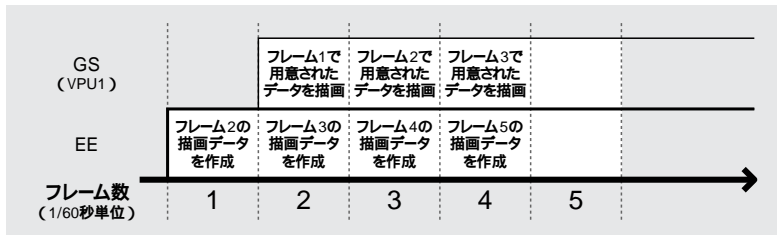
またチャンネル制御レジスタdn\_CHCRのビット6にはTTEビットというものがあり、このビットを立てるとDMAタグ自身も転送先プリフェラルへ転送されます。このTTEビットは、特にVPU1へのデータ転送において威力を発揮します。

例えば、VPU1へ頂点データを渡すときのことを考えてみます。メインメモリには頂点データの配列があるとします。通常、このような場合にはDMAのrefを使ってVPU1へ情報を転送することになります。

【図1】



【図2】

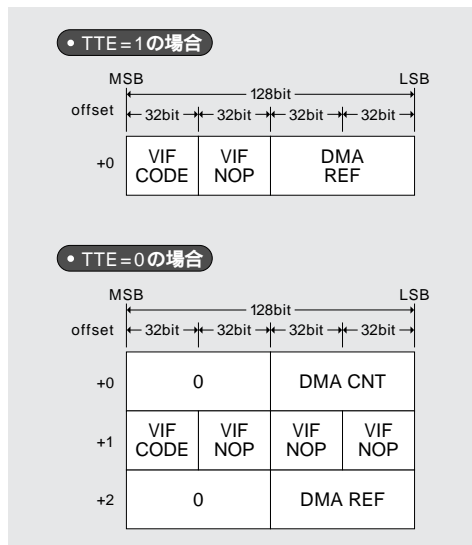


【表1】 DMA tagの種類

ID	動作
cnt	タグに続くデータを転送し、さらに次のデータへ進む
next	タグに続くデータを転送し、指定位置へジャンプ
ref	指定位置のデータを転送
refs	指定位置のデータをストール制御しながら転送
refe	指定位置のデータを転送し、転送を終了
call	タグに続くデータを転送し、次の番地を保存して指定位置へジャンプ
ret	タグに続くデータを転送し、保存されていた位置へジャンプ
end	タグに続くデータを転送し、転送を終了

<sup>\*1</sup> 2つのDMAは互いに向き調停されて転送されますが、PS2の効率のよい実行となると話は別で、別々のバスで同時にGSを描画させるのはあまりよくありません。GSは両者を並行して描画すると実力を発揮できないのです。しかしここからがプログラマの腕の見せどころで、これらは適材適所で使い分けることになり得ます。また1フレーム中において使い分けることもあるでしょう。PS2のチューニングにおいて最も効率よく動作させる必要があるのはGSです。どんなに効率のよいプログラムを書いたとしても、GSによる画面描画速度以上に速くすることはできません。 <sup>\*2</sup> 同様の通りバスは共通です。 <sup>\*3</sup> PCのV-RAMのイメージとは異なり、PS2などの3Dグラフィックシステムではディスプレイのリフレッシュごとに画面をすべて描画しなおします。普通、リフレッシュレートは60Hzですので、1画面の描画は1/60秒以内、終わらせなければなりません。 <sup>\*4</sup> 要のところ、1回のDMA転送のみですべて描画すると仮定しません。1度描画したデータをEEで加工して、再利用したい場合があります。そのときには、DMA転送による描画が終わった時点で読み込みをかけて、EE側にV-RAMを転送して処理を行った後、GSへ転送してDMA転送による描画を始めます。何度でも行うことができますから、アイデア次第でいろいろなことができます。後はプログラマの職人魂次第と言えます。

【図3】



この場合、VIFへは先にVIF CODEを渡す必要があります。しかし、TTEを用いると、DMAタグ自体の空き領域64bitにVIF CODEを埋め込めるので、DMA転送を効率よく扱うことができます(図3)。

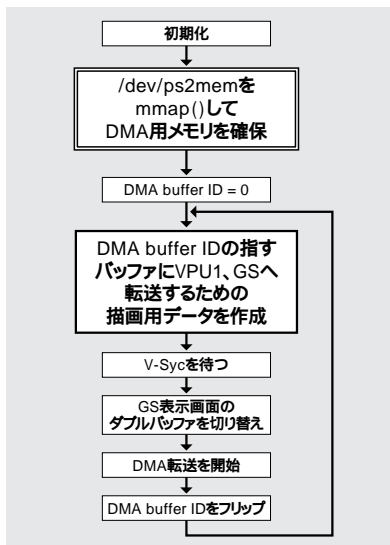
## PS2 LinuxでのプログラミングとDMAの実装

ここからは、PS2 Linuxの話に移ります。PS2 Linuxで実際にプログラミングを行う場合、だいたい図4a～図4cをベースとするような構造の構成になるでしょう。

多くの人が同じことをやると思うのですが、私の場合も例に漏れず、四角いキューブを表示させるプログラムを書いてみることにしました。ところが、実際に始めてみるとlibps2devのDMAの機能が妙に少ないことに気付きました。特にTTE=1の有無は重要で、これがあるのとないのでは大きくプログラミングの方針が変わってしまいます。我慢して使おうかとも考えたのですが、Linuxなんだし、ソースは全部揃っているのですから、改造することにしました。

最初はTTE=1だけならば簡単に追加できるだろうと考えていたのですが、実際に解析を始めると、いきなり驚かされたことがありました。DMAスタート関数であるps2\_dma\_start()の実装です。なんと、Source Chain ModeのDMA Tagを全部ソフトウェアで解釈しています。確かに実際にDMAを開始するioctl()にはChain ModeのDMA転送がありません。カーネルのドライバ部分のソースを読んでやっと気付いたのですが、実に当り前の本質的な制限でした。PS2 Linuxの開発担当者はこのような実装を行うにあたり、か

【図4a】



なり悩んだのではないかと想像します。

Linuxは仮想記憶を持つ普通のOSです。DMAコントローラにはメモリアドレスとして、仮想アドレスではなく、物理アドレスを渡す必要があります。仮想空間上では連続しているメモリ空間ですが、物理的にはページサイズ単位に分割されており、配置もバラバラです。

前述の通り、PS2 Linuxは、ドライバレベルではノーマルモード以外のDMAは無視し、連続したデータの転送のみを扱っています。ioctl()が発行されてDMA転送の要求を受け取ると、ドライバではデータ転送をページ単位に区切って分割します。DMA転送のNormal Modeはリニアな転送要求しか扱いませんからこれは簡単です。

続いて、ページ単位で対応する物理アドレスを求め、最後にページの数だけDMAのrefタグを並べて、1回のSource Chain ModeのDMAを実行します。以上のような流れで、仮想空間上のDMA転送が実現されています。

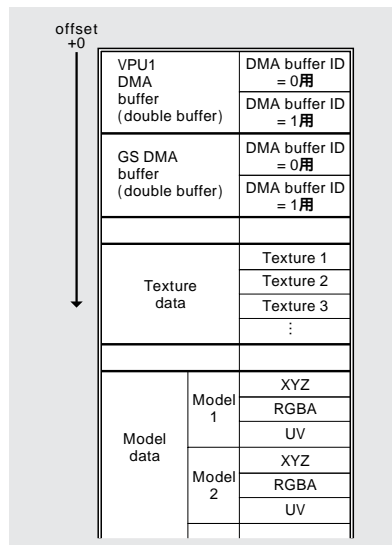
Chain Modeはソフトウェア上ですべてを解釈するために、ユーザーはわずらわしいアドレス空間の違いとかわれず、すべてを仮想アドレスで処理することができます。

### ●優れたDMA実装。しかし？

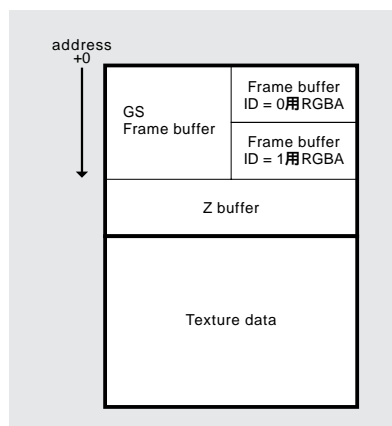
ここまでで述べたような実装ならば、PS2 Linuxでは普通にmalloc()したデータなどでも問題なく転送できそうですが、実際には、/dev/ps2memをmmap()してメモリを確保しなくてはなりません。また、malloc()したデータなどは、ドライバ側で確保した領域へ転送されてからDMA転送が行われるため、極端に速度が遅くなってしまいます。

何故このようになっているのでしょうか？ 実

【図4b】



【図4c】



は、ページ単位で対応する物理アドレスを求めるところがミソで、/dev/ps2memで確保したページはfile構造体メンバのprivate\_dataに物理アドレスが入っており、DMA転送時にはここを参照しているのです。PS2 LinuxはDMAアーキテクチャを以上のように実装していますが、これは実に上手な解決方法です。

理由は、/dev/ps2memで確保するメモリ領域は、一体どのくらい必要であるかを考えると明らかです。ここに置くべきデータは膨大な静的なテクスチャデータ、モデルのデータなどで、またEEの生成するDMAで参照されることのある動的なデータもすべて含まれることになるでしょう。PS2のノワーを最大限生かすためには10Mbytesくらいあれば十分なのでしょうか？ いや、ゲームによってはデータの9割以上が表示用データだと思われるから、もっと必要でしょう。私はPS2のゲーム開発経験がありますが、その時は20Mbytesは楽に超えていたように記憶しています。

では、Xで使うときはどうでしょうか。GSをどう

扱っているのかソースは全く見ていないので根拠があるわけではありませんが、ざっと見積もったところ、1Mbytesも使っていないように見えます。

以上のように、おおよその予測だけでも1Mbytes～20Mbytesと幅広くメモリの使用量は変化しますし、PS2の搭載メモリ量が32Mbytesですから、余裕はありません。動的なメモリ確保が必須となるでしょう。

さて、DMAで使用するメモリは物理的に連続している必要があります。Linuxで連続した物理メモリを確保する関数としては、`get_free_pages()` などがありますが、最大128Kbytesですから、到底こういう目的に使用できるものではありません。順当にいくなら、こちらを修正するのがよさそうですが、これは容易ではありません。そもそも、仮想空間にマップされた物理メモリというのは起動直後ならばともかく、時間が経つにつれて激しく対応が乱れていきます。カーネルが起動した後、ログインできるようになる時点でさえ、連続した領域はほとんどなくなっていると思われます。32Mbytes程度のメモリでは、例え1Mbytesの領域さえも、連続した領域を確保するのは難しいでしょう。

これを解決するとしたら、1度OSのすべてのプロセスを止めて、SWAP領域へ追い出せるプロセスをみんな追い出して、メモリの中身をソートして詰めることでしょうか。しかし、この作業の間OSは固まるしかありませんし、実際に実装することを考えると現実的な案ではなさそうです。

こうなると、PS2 LinuxのDMAの実装がいかにかえ抜かれたものであったかが見えてきます。PS2 Linuxはあくまで、PS2上へのLinuxの実装であり、その上でXが動作してさまざまなアプリケーションが動作します。32Mbytesのメモリは決して多いわけではありませんから、効率よく使わなければなりません。この前提を満たす上ではSCEによる実装は実にすばらしいものです。

しかし、`ps2_dma_start()` ではタグの数に比例してCPUを無駄に占有しますから、DMAのタグをチェーンすれば、その間は固まります。DMAのタグは実際、実用的に使うと

とばかりできない結構な量になります。このままではせっかくのDMAアーキテクチャが活いていません。

## カーネルを自分用に改造する

以上、実際に考え抜かれたPS2 LinuxのDMA実装ですが、前提条件を変えれば話は違います。

私がしたいのは単にグラフィックで遊ぶことだけですから、Xは必要ありませんし、その上で動作するアプリケーションのことも考える必要がありません。コンソール、あるいはネットワークからのログインができてGCCが動けば十分です。これならば、32Mbytesもメモリは必要ないでしょうし、16Mbytesでも十分と思われます。そして余った16MbytesをDMA専用にするれば、問題は解決です。

さっそく、自分専用のカーネルを作成し、そのための方針を決定します(表2A)。

DMA内のrefタグなどで指定するアドレスは、仮想アドレスではなく、物理アドレスを指定しなくてはなりません。しかし、この計算も自分でメモリ予約してしまっている以上、実に簡単で、単純に`/dev/ps2mem`で`mmap()`したメモリ領域のオフセットを予約開始アドレスに加えるだけです。ついでに、I/Oも直接制御したくなることは分かっていますので、さらに追加します(表2B)。これで当初の目的は達成できます。他にも機能がいろいろと必要になるかもしれませんが、それはまた欲しくなったときに追加すればいいのです。自分専用ですから。

実際にパッチ作成で行った作業は以下の通りです。

### ●起動時のメモリ領域を予約

これは、ソースツリーの`$(LINUXSRC)/arch/mips/mm/init.c`を変更します。i386のコードには、起動時のメモリ容量表示にreservedな領域サイズの表示があるのですが、MIPSのコードにはないので、ついでに付けます。本来ならば、リザーブ容量の設定はカーネルオプションあたりに追加する方法が綺麗で良いと思いますが、困ったときにリコンパイルすればいいだけのことで、`$(LINUXSRC)`

`/include/linux/ps2/dev.h`に値を定義するだけになります。

### ●/dev/ps2memの処理を変更

オリジナルのものでは、必要に応じてカーネルからメモリを取得して割り当てているのですが、今回は予約したページをそのままremapするようにします。DMAのためにfile構造体の`private_data`には物理アドレスを計算して設定することを忘れてはいけません。以上で、`/dev/ps2mem`は物理空間上に連続メモリが確保されます。

### ●Source ChainかつTTE=1であるDMA転送用に`ioctl()`を拡張

各デバイスを処理するコードは、`$(LINUXSRC)/drivers/video/ps2dev.c`にあります。DMAの`ioctl()`の処理は各デバイスから共通して、このソースファイル内の`ps2dev_send_ioctl()`が処理していますので、ここに追加します。PS2IOC\_SEND、PS2IOC\_SENDAに見習って、新しい定義は、PS2IOC\_SEND\_TTE1、PS2IOC\_SENDA\_TTE1とします。この定義も`$(LINUXSRC)/include/linux/ps2/dev.h`に追加します。

### ●実際に`ioctl()`で呼び出されたDMA転送を処理するコードを変更

DMA転送を処理するコードは、`$(LINUXSRC)/drivers/video/ps2dma.c`にありますので、こちらを変更します。Source ChainでTTE=1としてDMA転送を発行する関数を入れ込みます。PS2IOC\_SEND(A)のときのルーチンをうまく利用して辻褄を合わせ、必要ところは識別フラグを持たせるなどして、もとのルーチンとぶつからないように混在してDMA転送を発行できるように実装します。

### ●I/Oを直接制御できるように新しいデバイスを作成

名前は勝手に`/dev/ps2genio`と名付けますが、ドライバのソースでは使用していないminor番号の処理を追加して、`/dev/ps2mem`のように行ったremapを今度はI/O空間0x10000000-0x20000000に対して行います(`$(LINUXSRC)/drivers/video/ps2dev.c`と`$(LINUXSRC)/drivers/video/ps2mem.c`)。

### ●\$(LINUXSRC)/drivers/video/ps2mem.cを変更

`/dev/ps2genio`の`ioctl()`処理も追

【表2A】 自分専用カーネルの方針その1

- ・起動時にDMA用のメモリ空間を予約してしまい、Linuxからは使いません。
- ・`/dev/ps2mem`からの`mmap()`は予約した物理メモリに対して行います。
- ・`ioctl()`を拡張して、自分に必要なTTE=1かつSource Chain ModeのDMAを追加します。

【表2B】 自分専用カーネルの方針その2

- ・メモリ空間0x10000000-0x20000000を自由にアクセスできるデバイス`/dev/ps2genio`も用意します。
- ・DMAを自前で発行するときのことも考えて、`/dev/ps2genio`の`ioctl()`でデータキャッシュをフラッシュできるようにする。

加するために、\$(LINUXSRC)/drivers/video/ps2mem.cを変更します。さらに、\$(LINUXSRC)/include/linux/ps2/dev.hに新しく、2つのioctl()、PS2IOC\_FLUSH\_CACHE\_ALLとPS2IOC\_DMA\_CACHE\_WBACK\_INVを追加し、それぞれ、キャッシュの全範囲、指定範囲を処理させることにします。PS2IOC\_DMA\_CACHE\_WBACK\_INVは開始番地と長さの2つの引数が必要なので、構造体で渡します。

## ●バージョン定義を追加

いくらなんでも、勝手に作ったパッチですから、オリジナルのカーネルとの識別方法を用意しなくてはなりません。ioctl()で未定義のコードを指定するとEINVALを返しますから、実行時のチェックは可能です。コンパイル時のチェックのために、\$(LINUXSRC)/include/linux/ps2/dev.hにバージョン定義を追加しておきます。

以上で完成となりますが、実際にはここまで綺麗にことが進むわけではなく、ある程度の試行錯誤が必要でした。

完成したパッチは純正のカーネルとは互換性を保っていますので、サンプルのblowなども実行できます(blowの動作時、リザーブ領域として16Mbytesも取ってしまうと、普通ではスワップが発生してしまいます)。

## 使い方

ps2linux-patch-0.2-20010909.gz(今月号の付録CD-ROMに収録)を実行例1のようにしてカーネルにパッチを当てます。後は普通にカーネルコンパイルを行って、新しいカーネルで再起動します。カーネルのConfigurationとして、私は\$(LINUXSRC)/config\_ps2をそのまま使用しています。

実際の作業に当たっては、Boot menuを追加して純正カーネルも残して置かないと後悔することになりそうです。

無事にブートできたら、rootになってI/Oをマッピングするための/dev/ps2genioデバイスを作成します(実行例2)。以上でセットアップは終了です。

使い方ですが、libps2devではGSに対するSource Chain転送を、リスト1のように記述していましたが、この代わりにリスト2のように記述できるようになります。同じく、VPU1へは次のようにしてデータを転送できます(リスト3)。

lenはデータに関係なく16とします。これは、\$(LINUXSRC)/drivers/video/ps2dma.cのDMA処理を強引に拡張した結果で

### 【実行例1】

```
$ cd /usr/src/linux
$ gzip -dc ps2linux-patch-0.2-20010909.gz | patch -p1
```

### 【実行例2】

```
$ mknod /dev/ps2genio c 240 96
```

### 【リスト1】

```
ps2_dma_start( fdgs, NULL, datapointer );
```

### 【リスト2】

```
struct ps2_packet dma_gs;
dma_gs.ptr = datapointer;
dma_gs.len = 16;
ioctl( fdgs, PS2IOC_SEND_TTE1, &dma_gs );
```

### 【リスト3】

```
struct ps2_packet dma_vpul;
dma_vpul.ptr = datapointer;
dma_vpul.len = 16;
ioctl( ps2_vpu_fd(vpul),
        PS2IOC_SEND_TTE1, &dma_vpul );
```

す。lenの値そのものは使用しませんが、lenのエラーチェックはそのままなので、16の倍数を与えないとエラーになります。

DMA転送データのタグで記述するメモリアドレスは全部物理アドレスでなければなりません。仮想アドレスから物理アドレスへの変換方法としては、/dev/ps2memをmmap()した後のアドレスを覚えておき、引き算してオフセットを求めた後、本パッチの適用後に/usr/include/linux/ps2/dev.hで定義されるPS2\_RESERVE\_MEMORY\_STARTを加算することをお勧めします。

使用上の注意点をまとめると、以下のようになります。

- /dev/ps2memをmmap()する領域は起動時に予約します。本パッチ適用後は、/dev/ps2memは予約領域からしかメモリを取得できないのですが、Linuxからは予約領域を使用することはできません。予約領域のデフォルトは16Mbytesですが、用途に応じて変更した方がよいと思われます。その際、頂点情報やテクスチャなどもDMA転送する場合にはこの領域に置かなければならないことに注意してください。/usr/src/linux/include/linux/ps2/dev.hのPS2\_RESERVE\_MEMORY\_STARTとPS2\_RESERVE\_MEMORY\_SIZE。

- /dev/ps2memをmmap()するときにはoffsetを使ってはなりません。

- /dev/ps2genioはoffsetを指定してもよいのですが、ページ単位である必要があります。すべてのI/Oエリア0x10000000-0x20000000をマップすることはできませんが、EE関連の0x10000000-0x14000000程度ならば1度にマップすることができます(通常は必要ないはずですが、どうしても後半にアクセスしたいときは、offsetを使ってmmap()してください)。

- 上記2つのoffsetの制限は、少々修正することで緩和できますが、現在のところその必要性を感じたことはありません。

## DMAを自前でキック

せっかく/dev/ps2genioを追加したのですから、DMAを自分でキックしてみます。まず準備として、プログラムの初期化時にはリスト4のように/dev/ps2genioと/dev/ps2memをmmap()しておきます。実際にDMA転送を行う前に、CPUのデータキャッシュ内のデータをメモリに書き出しておく必要がありますので、先に/dev/ps2genioに対してioctl()を発行します。その後、I/Oポートに対してデータを書き込み、DMA転送をスタート

させます。これを実際にコーディングしたものは、リスト5のようになります(VPU1、GSに対してDMAを発行しています)。

ただし、本格的に使用するためには、このようなDMA発行はカーネルの関知するところではありませんので、カーネルのDMA発行(libps2devが暗黙で発行するものも含みます)とは問題を起こさないように調整しながら実装する必要があります。

ところで、DMA転送をスタートしたので、次に転送終了を待ってみましょう。これは、単にDMAのCHCRのビットが立ったままかどうかを確認するだけで良いのですが、Linuxのようなマルチタスクオペレーティングシステムではループで待つのは問題があります。

本来ならば、1度スレッドをスリープさせて、DMA終了の割り込みでスレッドをウェイクアップさせる必要があるのですが、今回は話を簡単にするために、usleep()で少し待ちながらループさせました。実際のプログラムコードは、リスト6のようになります。

## ●プログラム例

今月号の付録CD-ROMには、サンプルプログラムを4つ用意しています。

## ●dmamemtest-20010722.tgz

/dev/ps2memのテストと使用方法のサン

プルです。1度mmap()してメモリの内容を変更した後、もう1度mmap()しなおして、同じ内容がmmap()されることを確認します。オリジナルのPS2 Linuxとは動作が異なり、何度実行して2回11からは同じ物理メモリをアクセスすることが分かります。

## ●geniotest-20010722.tgz

/dev/ps2genioのテストと使用方法のサンプルです。I/Oポートの読み込みと書き込みを行います。本プログラムを実行すると、Returnを押すたびに、4つのタイマT0、T1、T2、T3の値が表示され、背景色がランダムに変化します。

## ●cube200-20010909.tgz

約200個のCUBEの表示を行います。パッドで適当にカメラを操作することができ、SELECTボタンで加算表示ができます。PS2らしいこともしておりませんし、PS2の性能を発揮したものでもありません。このパッチ日のデモという以上の意味はありません。

## ●cube200dma-20010909.tgz

cube200のdma発行をioctl()を使わずに直接I/Oポートをキックするように変更したものです。

## 最後に

PS2とLinux、実に面白い組み合わせです。しかしこの2つは、最適な組み合わせというわけではないと私は思っています。

本稿ではメモリを扱いましたが、ゲームの世界において重要な要素の1つであるリアルタイム性についても、今のLinuxでは脆弱な部分と言えるでしょう。

製品そのままのPS2 Linuxで楽しむのもいいでしょうし、ソフトウェアのソースは公開されているのですから、両者の特性がぴったり合うようにLinuxを改造する楽しみもあるでしょう。SCEは実に面白い商品を発売してくれたと思います。

### [リスト4] 初期化

```
int fdgenio = -1;
char *genio_p = MAP_FAILED;
int fddmamem;
char *dmamem_p;

void init( void )
{
    fdgenio = open( PS2_DEV_GENIO, O_RDWR );
    if ( fdgenio < 0 ) {
        /* error */
    }
    genio_p = mmap( 0, 0x4000000,
        PROT_READ | PROT_WRITE, MAP_PRIVATE, fdgenio, 0 );
    if ( genio_p == MAP_FAILED ) {
        /* error */
    }

    fddmamem = open( PS2_DEV_MEM, O_RDWR );
    if ( fddmamem < 0 ) {
        /* error */
    }
    dmamem_p = mmap( 0, PS2_RESERVE_MEMORY_SIZE,
        PROT_READ | PROT_WRITE, MAP_PRIVATE, fddmamem, 0 );
    if ( dmamem_p == MAP_FAILED ) {
        /* error */
    }
}
```

## 【リスト5】DMA初期化

```
/* DMAに先だって、キャッシュをフラッシュ */
struct ps2_genio_ioctl_wback_inv flushparam;

flushparam.addr = PS2_RESERVE_MEMORY_START;
flushparam.len = PS2_RESERVE_MEMORY_SIZE;
if (ioctl( fdgenio, PS2IOC_DMA_CACHE_WBACK_INV, &flushparam ) < 0) {
    /* おそらく/dev/ps2genioのioctl()がサポートされていないカーネル */
}
//ioctl( fdgenio, PS2IOC_FLUSH_CACHE_ALL, 0 );  <- もちろんこちらでもよい

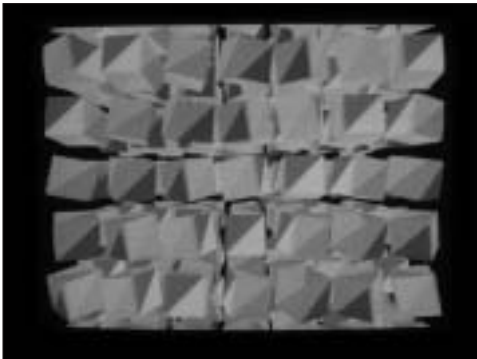
/* VPU1へDMA転送 */
*(unsigned int *) (genio_p+0x10009030 /* TADR */ -PS2_DEV_GENIO_OFFSET) =
    vpu_ptr_addr - (unsigned int) dmamem_p + PS2_RESERVE_MEMORY_START;
*(unsigned int *) (genio_p+0x10009020 /* QWC */ -PS2_DEV_GENIO_OFFSET) = 0;
*(unsigned int *) (genio_p+0x10009000 /* CHCR */ -PS2_DEV_GENIO_OFFSET) = 0x0145;
    (tte = 0であればCHCRには0x0105を発行)

/* GSへDMA転送 */
*(unsigned int *) (genio_p+0x1000a030 /* TADR */ -PS2_DEV_GENIO_OFFSET) =
    gs_ptr_addr - (unsigned int) dmamem_p + PS2_RESERVE_MEMORY_START;
*(unsigned int *) (genio_p+0x1000a020 /* QWC */ -PS2_DEV_GENIO_OFFSET) = 0;
*(unsigned int *) (genio_p+0x1000a000 /* CHCR */ -PS2_DEV_GENIO_OFFSET) = 0x0105;
```

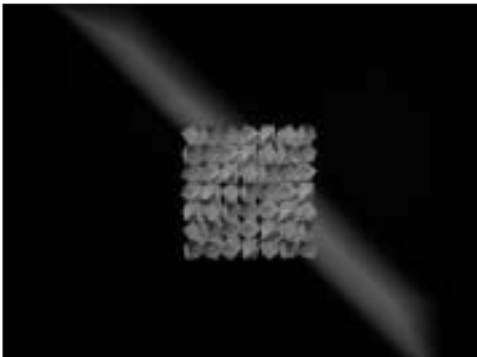
## 【リスト6】DMA転送終了待ち

```
while (*(unsigned int *) (genio_p+0x10009000-PS2_DEV_GENIO_OFFSET) & 0x0100)
    usleep( 1000 );
while (*(unsigned int *) (genio_p+0x1000a000-PS2_DEV_GENIO_OFFSET) & 0x0100)
    usleep( 1000 );
```

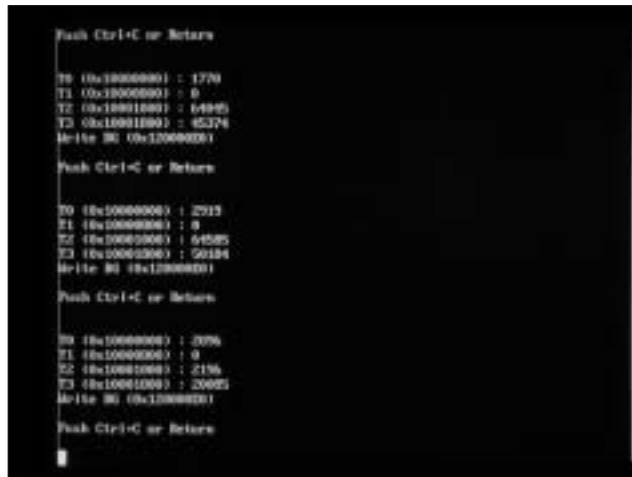
【画面1】 cube200d ma-20010909.tgzの実行画面1



【画面2】 cube200d ma-20010909.tgzの実行画面2



【画面3】 geniotest-20010722.tgzの実行画面



## Resource

### [1] PlayStation2 Linux DMAおよび直接I/Oポート制御のためのカーネルパッチ

<http://homepage1.nifty.com/fukui/ps2linux/dmaio.html>

### [2] PlayStation 2 Linux 約200個のキューブを表示

<http://homepage1.nifty.com/fukui/ps2linux/cube200.html>

### [3] 有限会社メタロジック

<http://www.metalogic.co.jp/>

# PS2Linux ファン

たむらけんいち  
<http://rubysdl.s1.xrea.com/>

## Playstation2 LinuxKit with Ruby/SDL

さて、今回のネタは、PlayStation2 Linux Kit (以降PS2 LinuxKit)だ。内容的に Kondara MNU/Linux 1.2 をベースに移植されているとあって、パッケージの中にruby-1.4.4-4.mipsel.rpmが用意されていたし、SDLだって問題なくビルドできた。その上、SDL-1.2.1では公式に対応されたのである。この辺から紹介しよう。

### Ruby/SDL 近況

大林 一平氏の手によるRuby/SDLとは、オブジェクト指向スクリプト言語Rubyから、クロスプラットフォームなマルチメディアライブラリ SDLを利用するための拡張モジュールである(記事木のResource [1]、[2]、[3]を参照)。Rubyはずいぶん有名になったし、SDLは、LinuxJapanでは Adas氏の『SDLバカ一代』の連載記事があるので、詳細は必要ないだろう([4])。Ruby/SDLに関しては、2001年7月頃に記事を執筆させていただいた([5])。実際に原稿を書き上げたのは、3月末だったのだが、当時と、現在(2001年9月末)のバージョンの対比を挙げておく(表1)。

Ruby/SDLに関する変更点をリストアップすると、表2のようになる。この辺の経過は、随時筆者のWebサイト([6])でも取り上げているのでオンラインで確認してほしい。

もっとも、今回の記事にRuby/SDLは登場しない。Ruby/SDLは、結局のところRubyとSDLが動く環境ならば、利用することができる。

【表1】 Ruby、SDL、Ruby/SDLのバージョン比較

	2001年7月号	2001年9月末
Ruby	1.6.3	1.6.5
SDL	1.2.0	1.2.2
Ruby/SDL	0.3	0.6 (0.7 dev)

【表2】 Ruby/SDLの変更点

- require 'rubysdl' から、require 'sdl'に変更
- Ruby/OpenGLがRuby/SDLから利用できるようになった
- FreeBSD 4-Stubleで動作確認が取れ、Portsに含まれるようになった
- 他バグフィックスやドキュメント追加など

### 【リスト1】

```
--enable-video-ps2gs    use PlayStation 2 GS video driver [default=yes]
```

一つの環境で動くスクリプトは他でも動くはずである。今回の内容はそのマルチプラットフォームな実装とクロス環境の構築についてである。

### SDLの PS2対応

#### ●何をサポートしたのか

SDLのPlayStation2(以下、PS2) 公式対応とは一体何をして対応と呼ぶのだろうか? PS2 Linux Kitでは X Window System(以下、X)が動くので、特に何もしなくてもXのプログラムやライブラリは動作する。もちろん、SDLをX上で動かすことに何の問題もない。SDL-1.2.1のPS2対応とはズバリGSのサポートを意味する。

GS(Graphics Synthesizer)とは、PS2の描画、ローカルメモリ、ビデオ出力処理を担当するデバイスで、通常のPCでいえばビデオカードに位置する。描画は、ローカルメモリ内のフレームバッファが対象となり、フルカラー内部演算や、シェーディング、テクスチャマッピング、アルファブレンディングなどのレンダリング処理が可能である。詳細は、PS2Linux Kitに付属しているDVD-ROMの\$DVD/SCSI/Document/PDF/gs/Gsuser.pdfと、「GS User's Manual」を参照してほしい。

#### ●マルチプラットフォームのために

最近のプログラムは、例えばRubyでもSDL(そして関連するライブラリ群たち)でもインストールするのは簡単で、

```
# ./configure && make
# [make test]
# sudo make install
```

などとコンソール上で打ち込むのが一般的である。普段何々なく、./configureと打っているが、この「configure」というシェルスクリプトは、OSなどの環境に依存しているプログ

ラムの場所や有無を検出して適切なMakefileを吐いてくれる大したやつなのである。このconfigureを生成するツールが、automake/autoconfなどのAutotoolである。そして、configureを利用することで、利用できるsubsystemの検出が可能なのだ。

例えばSDL-1.2.1以降で、./configure --helpとすると、リスト1に挙げたオプションが見つかる。このオプションが、configureによってどのように検出されるのか追ってみよう。

リスト2(52ページ)に挙げた部分では、オプション引数を処理して、正常にテストプログラムがビルドできるかで存在チェックを行っている。存在していれば、「ENABLE\_PS2GS」というシンボルを定義している。

コンパイル時、video/SDL\_video.cでは、bootstrapの中で各subsystemの有効、無効のシンボルが定義されているかを順に見ていく。`ENABLE\_PS2GS'が有効な場合、video/ps2gs/SDL\_gsvideo.cで定義されてるリスト3がコンパイルされていく。Available()/CreateDevice()をX11のソースコードと比較してみよう(53ページのリスト4、54ページのリスト5参照)。

お分かりのように、\*\_Available()関数は、描画前の初期化、\*\_CreateDevice()は、SDL\_VideoDevice 構造体へのセットを行っている。\*\_VideoInit()や、

\*\_ListModes()もsubsystemでデバイス依存な処理をそれぞれ担当している。

つまり、SDLでは、

- autotoolを利用した実装の検出
- 実装依存部を階層化したソース構造

の2つで、マルチプラットフォームを実現しているのである。

なお、他のデバイスに関しても同様な仕組みになっている。Audioデバイスに関しては、SDL SourceTourで詳しく解説してあるので、ぜひ併せて読んでほしい ([7])。

## ソースの追いつ

他人のソースを追うという行為は、あるときは単なる苦痛であり、またあるときはワクワクするような胸躍る体験である。通常、前者は仕事としてのデバッグで、後者はオープンなソースをホゲっているときが多い(ハハ)。プログラムのソースも規模が大きくなると、構成を理解するだけでも大変だし、宣言部や関数の実装は当然分割されていることになる。リスト6のようにするのは常套手段だが、以下に挙げていく専用ツールを利用することで、効率が上がる場合も多い。

### ●GLOBAL

GLOBAL ([8]) は、ソースコードタグシステムと呼ばれるものだ。ソース群のトップで、gtagsを実行しておくだけでタグを生成してくれる。例えば、`global -x SDL_SetVideoMode` などとするだけで、リスト7のような情報を教えてくれる。もっとも、SDLの関数なら、`man SDL_SetVideoMode` とした方が良い(笑)。

またGLOBALを利用すると、less、vi、Emacsなどの画面からハイパーリンクをたどって欲しい情報を入手することも可能だ。そして何より、「`gtags; htags`」とするだけで、HTMLディレクトリ以下にコードハイライトされたソースのHTMLファイルを作ってくれ、関数や定義はすべてリンクが張られる。この例としては、多摩通信 ([9]) の「UNIXのためのフリーソフト」というコーナーに、「UNIXカーネルソー

スツアー」と銘打った各種UNIXのカーネルソースのHTMLページがある。非常に便利なのがすぐに分かる好例なので、ぜひ見てほしい。global、gtagsは、BCC-5.5などのWindows用コンパイラでもコンパイル可能だし、h tagは\$GLOBAL/htags/htags.plをActivePerlなどのWin32なPerlで実行すれば利用できる。インストールも、Linuxならmake installでOKだ。

### ●The Source-Navigator IDE

Red Hatが2000年7月にオープンソース化。プロジェクトエディタ、ソースエディタのみならず、GREPやXref(クロスリファレンス)などの機能から、make、rcs、cvsの呼び出しなども持つIDE環境である。ソースコードは、globalの生成したHTML同様カラーリングされる。Tclで記述されたと思われるエディタからは、左クリックのメニューから関数実装部や宣言部へジャンプが可能だ。Red Hat社のページからソース、各UNIX、またWin32バイナリがダウンロードできる ([10])。

### ●Namazu

日本の誇る全文検索システム、Namazu ([11])。漠然としたキーワードで検索したいときは早くて便利。Namazuならば、正規表現使ってマニュアルも対象にできる。w3mと組み合わせ、CGIライクに使うのがお勧めである。

## PS2 Linux用のクロスコンパイル

### ●クロス開発環境の構築

GCCはクロス開発環境をサポートしているコ

ンパイラである。このクロス開発環境とは、実際にプログラムを動かす環境以外で開発を行い、出来上がったバイナリを転送して実行するスタイルを指す。組み込み機器や、MobileGear、最近ではWindows CEマシンなどのUNIX環境の構築などにも用いられている。また構築方法は、基本的には実際に動かす環境(以降Targetと呼ぶ。開発を行う環境はBuild環境)用のGCC、GNU C Library (glibc)、GNU Binutilsを用意するだけである。

塩崎さんという方が、NetBSD/i386にてPS2Linux Kit用クロス環境構築について発表されている ([12]) ので、これを参考に(というかそのままパクらせて)いただいた。ただし、そのままだと本当に単なるパクリ(笑)なので少し手を加えてある。シェルスクリプト化して、環境依存部を変数化したり、パッチを自動的に当てたりしてある(55ページのリスト8)。たぶんこれで、GCCが動かささまざまな環境で構築が楽になるはずである。以下、必要なものを解説しよう。

### ●PS2 Linux Kit付属のバイナリとソース

表3に挙げたファイルを、PS2 Linux上から、ネットワーク経由でFTPでもSambaでもscpでも構わないので、Build環境1に持ってきてほしい。

binutilsとgccのソースは、Target環境用のバイナリを作るため。gccのmipselバイナリと各種ライブラリ群はコンパイラが利用するライブラリファイルだ。

Linuxの場合、システムコール利用するヘッダはカーネルのヘッダをインクルードする必要があるため、Kernel-headersも展開するしなければならない。libps2devはPS2 Linux固有のライブラリだ。

### ●Build環境用のネイティブなrpmパッケージ(rpm2cpio)、gmake、bison、cpio

rpmパッケージは、Red Hat系以外のLinuxや、\*BSDではcontrib、ports、pkgsrcなどに用意されているので、そちらを使ってほしい。あまりバージョンが高いとマズイかも知れない。ちなみに、PS2 Linux Kitのバージョンは、3.0.4である。その他は特に問題ないはずだ。

シェルスクリプトの中で変更する必要がある変数は、RPMSとPREFIX、MAKEくらいだろう。RPMSは、PS2 LinuxKitから持って来たrpmファイルを置いた場所を指す。PREFIXは、クロス環境のインストール先である。

TARGET 変数で指定されている

[リスト3]

```
PS2GS_bootstrap
GS_Available
GS_CreateDevice
GS_VideoInit
GS_ListModes
GS_SetVideoMode
...
```

[表3] 入手するPS2 Linux Kit付属のバイナリとソース

```
DVD.SRPMsbinutils-2.9EE-3.src.rpm
DVD.SRPMsgcc-2.95.2-3.src.rpm
DVD.SCEVRPMS/gcc-2.95.2-3.mipsel.rpm
DVD.SCEVRPMS/glibc-2.2.2-3.mipsel.rpm
DVD.SCEVRPMS/glibc-devel-2.2.2-3.mipsel.rpm
DVD.SCEVRPMS/gcc-c++-2.95.2-3.mipsel.rpm
DVD.SCEVRPMS/kernel-headers-2.2.1_ps2-6.mipsel.rpm
DVD.SCEVRPMS/libps2dev-0.9-1.mipsel.rpm
```

[リスト6]

```
find ./ -name \*.h|xargs grep SDL_SetVideo
```

[リスト7]

```
SDL_SetVideoMode 522 src/video/SDL_video.c SDL_Surface * SDL_SetVideoMode (int width, int height, int bpp, Uint32 flags)
```

#### 【リスト9】hello.c

```
#include <stdio.h>

int main()
{
    printf("Hello,PS2.\n");
    return 0;
}
```

「mipsEEel-linux」が、Targetとなるクロスホストタイプとなる。この「mipsEEel-linux」は、PS2 Linux Kit用パッケージのパッチに使うホスト名だ。アーカイブの中のディレクトリ名もこれだったりするので、変更せずにそのままの指定にしておこう（もちろん、Hackして、書き換えるのはあなたの自由だ）。ただし、この名前をそのままクロスコンパイル時のTargetに指定するとマズイことになってしまう。この辺は後述する。

PREFIXの初期値は、/usr/local/cross\_ps2とした。これは独立したディレクトリならどこでも構わないだろう。シェルスクリプトの中でmkdirしたりしているので、権限のあるユーザーか、あるいは前もって「chown -R foo:bar」しておくこと。ちなみに筆者は、わたなべひろふみ氏（[13]）の真似をして/usr/local以下は、

```
chown -R tamura:tamura
```

としている。自宅でも職場でも、基本的に私個人が占有しているLinuxマシンはこんなヤクザな設定で使っている（もちろんサーバ用途のマシンはちゃんとしている）。MAKEに関しては、\*BSDな環境ではgmakeを指定してほしい（常識か?）。後は、「sh mk\_cross4ps2.sh」だ。

すると、マシンや環境によってはかなり時間がかかるが、必要なモノさえ揃っていればスクリプトを実行後クロス環境ができて上がるはずである。とりあえず、お約束で以下のコードをhello.cとして作り（リスト9）、コンパイルしてみよう（実行例1）。

後は、ftpなどでPS2Linuxに持っていく。実行属性(x)が付いているのを確認して（付いてなければ、chmod +x mipsEEel-linux-hello）、実行例2のようにする。

無事「Hello,PS2.」と表示されたらどうか? fileコマンドの実行結果をi586マシンでコンパイルしたものと一緒に紹介しておこう（リスト10）。

## Rubyをクロスで作ってみよう

ここでは、ruby-1.6.5.tar.gzで試し

#### 【実行例1】

```
$ export PATH=/usr/local/cross_ps2/bin:$PATH
$ mipsEEel-linux-gcc hello.c -o mipsEEel-linux-hello
```

#### 【実行例2】

```
$ ./mipsEEel-linux-hello
```

#### 【リスト10】

```
i586-cygwin-hello.exe: MS Windows PE Intel 80386 console executable not relocatable
mipsEEel-linux-hello: ELF 32-bit LSB mips-3 executable, MIPS R3000 LE [bfd bug],
version 1, dynamically linked (uses shared libs), not stripped
```

#### 【リスト11】

```
\--- config.sub-org      Tue Jul 16 13:01:42 2001
+++ config.sub      Tue Sep 18 23:11:56 2001
@@ -227,7 +227,7 @@
        | we32k | ns16k | clipper | i370 | sh | sh[34] \
        | powerpc | powerpcle \
        | 1750a | dsp16xx | pdp10 | pdp11 \
-       | mips16 | mips64 | mipsel | mips64el \
+       | mips16 | mips64 | mipsel | mipsEEel | mips64el \
        | mips64orion | mips64orionel | mipstx39 | mipstx39el \
        | mips64vr4300 | mips64vr4300el | mips64vr4100 | mips64vr4100el \
        | mips64vr5000 | mips64vr5000el | mcore | s390 | s390x \
@@ -273,7 +273,7 @@
        | clipper-* | orion-* \
        | sparclite-* | pdp10-* | pdp11-* | sh-* | powerpc-* | powerpcle-* \
        | sparc64-* | sparcv9-* | sparcv9b-* | sparc86x-* \
-       | mips16-* | mips64-* | mipsel-* \
+       | mips16-* | mips64-* | mipsel-* | mipsEEel-* \
        | mips64el-* | mips64orion-* | mips64orionel-* \
        | mips64vr4100-* | mips64vr4100el-* | mips64vr4300-* | mips64vr4300el-* \
        | mipstx39-* | mipstx39el-* | mcore-* \
```

てみよう。Rubyは、ビルド時にRuby自体が必要なので、ネイティブなバイナリをまず作ってインストールしてほしい。ネイティブなbuildでは、minirubyというスタティックなバイナリを作ってそれを利用する。Perlなども同様だ。

それでは早速、configureの実行である。まず、setpggrp/getpggrpの検出はクロス環境では対応していないので指定する必要がある。同様に、ac\_cv\_c\_bigendianの指定もしておく。PS2のEE Core (R5900)はMips系だが、エンディアンはインテル同様リトルエンディアンである。i386用LinuxやCygwinではコンパイルできたが、環境を考えるとincludedirも明示的に指定すべきだろう（実行例3）。

--targetは、mipsEEel-linuxのような指定だとエラーになる。mipsel-\*にすると、config.subで判定してもらえるが、そうするとGCCなどのprefixが判定できずに、ネイティブなGCCを使ってしまう。

実行例4のようにするとbuildできるのだが、長過ぎである。しかもconfigure実行画

面を見ればお分かりのように、ar、ranlibなどがネイティブなものを検出してしまふ。

通常のクロス環境ならば、--hostにこのプレフィックスを指定できるが、このままでは、やはりマシン検出でエラーになってしまう。そこで、config.subに手を加えることにする。リスト11のパッチだ。

クロス構築時の、\$WORK/gcc/gcc-2.95.2.EE-linux.patchをちょっと眺めて作った。要は「mipsel-なんとか」がOKで、「mipsEEel-」でダメなら、判定部に追加すればいいちゃんってことである。`case \$basic\_machine`以下の、「the basic CPU types without company name.」と、「the basic CPU types with company name.」がそれだ。このパッチを当ててもなく、エディタで追加する方がいい。他のプログラムでも同様に対応できると思うが、もちろんAt Your Own Riskである。

結局、configureに与えるパラメータは実行例5の通りとなった。`--target=

mipsEEl-ps2-linux' は、単に configure の check で、`mipsEEl-unknown-linux' と表示されるのが嫌なだけである。

configure が終わったら、make してみよう。ext ディレクトリ以下の拡張ライブラリ群は、必要なライブラリファイルなどが存在しなければスキップしてくれる。socket.so は問題なく作られるはずである。

とりあえず、Ruby、socket.so を、mipsEEl-linux-strip -s としてから PS2Linux に持っていく、**実行例6**のようにコマンドを実行してほしい。

"-rssocket" をはずしたものと実行結果を比較して、間違いなく Socket クラスが require されたことが確認できる。ちなみにこのクロスコンパイルした Ruby を file コマンドで見るとリスト12のようになり、「あれっ」と思うのだが、PS2Linux の GCC でコンパイルしても、元からインストールした RPM なのでも同じであった。file コマンドがおかしいのか、GCC の設定がぬるいのか……。

#### 【実行例3】

```
$ mkdir cross_ps2; cd cross_ps2
$ ac_cv_func_getpgrp_void=yes \
  ac_cv_func_setpgrp_void=yes \
  ac_cv_c_bigendian=no \
  ../configure --target=mipsEEl-linux \
  --includedir=/usr/local/cross_ps2/include
```

#### 【実行例4】

```
% CC=mipsEEl-linux-gcc ac_cv_func_getpgrp_void=yes \
  ac_cv_func_setpgrp_void=yes ac_cv_c_bigendian=no \
  ../configure --target=mipsel-linux \
  --host=mipsel-linux --build=i586-pc-linux \
  --includedir=/usr/local/cross_ps2/include
```

#### 【実行例5】

```
$ ac_cv_func_getpgrp_void=yes \
  ac_cv_func_setpgrp_void=yes ac_cv_c_bigendian=no \
  ../configure --target=mipsEEl-ps2-linux-gnu \
  --host=mipsEEl-linux --build=i586-pc-linux-gnu \
  --includedir=/usr/local/cross_ps2/include
```

#### 【実行例6】

```
$ ruby -v -rssocket -e 'p Class.constants.grep(/TCP/)'
ruby 1.6.5 (2001-09-19) [mipsEEl-linux-gnu]
["TCPsocket", "TCPserver", "TCPsocket", "TCPserver"]
```

#### 【リスト12】

```
ruby: ELF 32-bit LSB executable, MIPS R3000_BE - invalid byte order,
version 1, dynamically linked (uses shared libs), stripped (実際には1行)
```

## Column

Windows 9x、Me、NT、2000、XP でクロス環境を作るには、Cygwin を勧める([14])。必要な rpm2cpio、cpio は、Project HeavyMoon([15]) にバイナリが用意されている。Cygwin とは、Win32 上で UNIX エミュレーションを行う環境であり、GCC なども含めた一通りの開発環境まで手に入る。Version 1.x 以降は結構安定しているし、sshd とか、perl-5.6.0 とか、Python とか、PostgreSQL とか含まれていたりする。ただし、以前に比べるとだいぶ安定してきたといっても、Cygwin のバージョンのマイナーナンバーは、基本的に偶数になることはないで、完全な初心者入門用にはお勧めできない。詳しくは Project HeavyMoon の Cygwin

文書図書館を参照してほしい。

Cygwin の場合、Windows 環境で実行するために、拡張子として「.exe」をつける必要がある。これは、configure.in の AC\_EXEEXT マクロで判定している。これを避けるには、ac\_cv\_exeext=no をセットするだけで良いはずだが、Cygwin では GCC が勝手に .exe を追加してしまう。今回のシェルスクリプトにはこの対応のパッチもすでに含めてある。

ただし、何故か実行ファイルの prefix に mipsEEl-linux- なんて付くので後で名前を変更すること。また、configure 実行時の環境変数の設定で「=」の前後に空白があるとエラーになるので注意が必要である。

(たむらけんいち)

## Windows 環境でもクロスコンパイル

#### 【画面】Project HeavyMoon



## Resource

### [1] Web page of ohai

<http://www.kmc.kyoto-u.ac.jp/~ohai/>

### [2] Ruby公式ページ

<http://www.ruby-lang.org/ja/>

### [3] Simple DirectMedia Layer

<http://www.libsdl.org/>

### [4] Adas' Linuxゲームプログラム

<http://www.geocities.co.jp/Berkeley/2093/>

### [5] RubySDLへの招待

<http://www6.tok2.com/home/tamura/RubySDL/>

### [6] Just another RubySDL porterなページ

<http://www6.tok2.com/home/tamura/rubysdl/>

### [7] SDL Source Tour

<http://www2.jan.ne.jp/~zinnia/sdl/sourcetour/>  
SDL Watch (<http://www2.jan.ne.jp/~zinnia/sdl/watch.html>)は、SDLを利用したいと思う人間には必須ページだ。

### [8] GLOBAL

<http://www.tamacom.com/global/>

### [9] 多摩通信

<http://www.tamacom.com/index-j.html>

### [10] The Source-Navigator IDE

<http://sources.redhat.com/sourcnav/>

### [11] 全文検索システムNamazu

<http://www.namazu.org/>

### [12] PS2Linuxクロス環境 on NetBSD-current

<http://ps2.imou.to/cross.html>

### [13] Just another Ruby porter,

<http://www.ruby-lang.org/~eban/>

### [14] Cygwin Official Home

<http://cygwin.com/>

### [15] Project HeavyMoon

[http://www.sixnine.net/cygwin/index\\_ja.html](http://www.sixnine.net/cygwin/index_ja.html)

CygwinなバイナリをRPMで配布している野心的なサイト。最近のCygwinに関する文書が充実している。

### [リスト2]

```
#lined(5279,<<)
CheckPS2GS()
{
    # Check whether --enable-video-ps2gs or --disable-video-ps2gs was given.
    if test "${enable_video_ps2gs+set}" = set; then
        enableval="${enable_video_ps2gs}"
        :
        :(snip)
        :
    else
        enable_video_ps2gs=yes
    fi

    if test x$enable_video = xyes -a x$enable_video_ps2gs = xyes; then
        echo $ac_n "checking for PlayStation 2 GS support"... $ac_c" 1">&6
        echo "configure:5292: checking for PlayStation 2 GS support" >&5
        video_ps2gs=no
        cat > conftest.$ac_ext <<EOF
#line 5295 "configure"
#include "confdefs.h"

        #include <linux/ps2/dev.h>
        #include <linux/ps2/gs.h>

int main() {

; return 0; }
EOF
if { (eval echo configure:5306: \"\$ac_compile\") 1>&5; (eval $ac_compile) 2>&5; }; then
    rm -rf conftest*

    video_ps2gs=yes

else
    echo "configure: failed program was:" >&5
    cat conftest.$ac_ext >&5
fi
```

```

rm -f conftest*
    echo "$ac_t""$video_ps2gs" 1>&6
    if test x$video_ps2gs = xyes; then
        CFLAGS="$CFLAGS -DENABLE_PS2GS"
        VIDEO_SUBDIRS="$VIDEO_SUBDIRS ps2gs"
        VIDEO_DRIVERS="$VIDEO_DRIVERS ps2gs/libvideo_ps2gs.la"
    fi
fi
}

```

#### 【リスト4】SDL\_gsvideo.c

```

/* GS driver bootstrap functions */

static int GS_Available(void)
{
    int console, memory;

    console = open(PS2_DEV_GS, O_RDWR, 0);
    if ( console >= 0 ) {
        close(console);
    }
    memory = open(PS2_DEV_MEM, O_RDWR, 0);
    if ( memory >= 0 ) {
        close(memory);
    }
    return((console >= 0) && (memory >= 0));
}

/* : (snip)
:
: */

static SDL_VideoDevice *GS_CreateDevice(int devindex)
{
    SDL_VideoDevice *this;

    /* Initialize all variables that we clean on shutdown */
    this = (SDL_VideoDevice *)malloc(sizeof(SDL_VideoDevice));
    if ( this ) {
        memset(this, 0, (sizeof *this));
        this->hidden = (struct SDL_PrivateVideoData *)
            malloc((sizeof *this->hidden));
    }
    if ( (this == NULL) || (this->hidden == NULL) ) {
        SDL_OutOfMemory();
        if ( this ) {
            free(this);
        }
        return(0);
    }
    memset(this->hidden, 0, (sizeof *this->hidden));
    mouse_fd = -1;
    keyboard_fd = -1;

    /* Set the function pointers */
    this->VideoInit = GS_VideoInit;
    this->ListModes = GS_ListModes;
    this->SetVideoMode = GS_SetVideoMode;
    this->CreateYUVOverlay = GS_CreateYUVOverlay;
    this->SetColors = GS_SetColors;
    this->UpdateRects = NULL;
    this->VideoQuit = GS_VideoQuit;
    this->AllocHWSurface = GS_AllocHWSurface;

```



```

    this->CheckHWBlit = NULL;
    this->FillHWRect = NULL;
    this->SetHWColorKey = NULL;
    this->SetHWAAlpha = NULL;
    this->LockHWSurface = GS_LockHWSurface;
    this->UnlockHWSurface = GS_UnlockHWSurface;
    this->FlipHWSurface = NULL;
    this->FreeHWSurface = GS_FreeHWSurface;
    this->SetIcon = NULL;
    this->SetCaption = NULL;
    this->GetWMInfo = NULL;
    this->FreeWMCursor = GS_FreeWMCursor;
    this->CreateWMCursor = GS_CreateWMCursor;
    this->ShowWMCursor = GS_ShowWMCursor;
    this->MoveWMCursor = GS_MoveWMCursor;
    this->InitOSKeymap = GS_InitOSKeymap;
    this->PumpEvents = GS_PumpEvents;

    this->free = GS_DeleteDevice;

    return this;
}

```



[リスト5] SDL\_x11video.c

```
static int X11_Available(void)
{
    Display *display;

    display = XOpenDisplay(NULL);
    if ( display != NULL ) {
        XCloseDisplay(display);
    }
    return(display != NULL);
}

/* : (snip)
:
: */

static SDL_VideoDevice *X11_CreateDevice(int devindex)
{
    SDL_VideoDevice *device;

    /* Initialize all variables that we clean on shutdown */
    device = (SDL_VideoDevice *)malloc(sizeof(SDL_VideoDevice));
    if ( device ) {
        memset(device, 0, (sizeof *device));
        device->hidden = (struct SDL_PrivateVideoData *)
            malloc((sizeof *device->hidden));
        device->gl_data = (struct SDL_PrivateGLData *)
            malloc((sizeof *device->gl_data));
    }
    if ( (device == NULL) || (device->hidden == NULL) ||
        (device->gl_data == NULL) ) {
        SDL_OutOfMemory();
        X11_DeleteDevice(device);
        return(0);
    }
    memset(device->hidden, 0, (sizeof *device->hidden));
    memset(device->gl_data, 0, (sizeof *device->gl_data));

    /* Set the driver flags */
    device->handles_any_size = 1;

    /* Set the function pointers */
    device->VideoInit = X11_VideoInit;
    device->ListModes = X11_ListModes;
    device->SetVideoMode = X11_SetVideoMode;
    device->ToggleFullScreen = X11_ToggleFullScreen;
    device->UpdateMouse = X11_UpdateMouse;
#ifdef XFREE86_XV
    device->CreateYUVOverlay = X11_CreateYUVOverlay;
#endif
    device->SetColors = X11_SetColors;
    device->UpdateRects = NULL;
    device->VideoQuit = X11_VideoQuit;
    device->AllocHWSurface = X11_AllocHWSurface;
    device->CheckHWBlit = NULL;
    device->FillHWRect = NULL;
    device->SetHWColorKey = NULL;
    device->SetHWAlpha = NULL;
    device->LockHWSurface = X11_LockHWSurface;
    device->UnlockHWSurface = X11_UnlockHWSurface;
    device->FlipHWSurface = X11_FlipHWSurface;
    device->FreeHWSurface = X11_FreeHWSurface;
    device->SetGamma = X11_SetVidModeGamma;
    device->GetGamma = X11_GetVidModeGamma;
    device->SetGammaRamp = X11_SetGammaRamp;
    device->GetGammaRamp = NULL;
#ifdef HAVE_OPENGL
    device->GL_LoadLibrary = X11_GL_LoadLibrary;

```

```

        device->GL_GetProcAddress = X11_GL_GetProcAddress;
        device->GL_GetAttribute = X11_GL_GetAttribute;
        device->GL_MakeCurrent = X11_GL_MakeCurrent;
        device->GL_SwapBuffers = X11_GL_SwapBuffers;

#ifdef
        device->SetCaption = X11_SetCaption;
        device->SetIcon = X11_SetIcon;
        device->IconifyWindow = X11_IconifyWindow;
        device->GrabInput = X11_GrabInput;
        device->GetWMInfo = X11_GetWMInfo;
        device->FreeWMCursor = X11_FreeWMCursor;
        device->CreateWMCursor = X11_CreateWMCursor;
        device->ShowWMCursor = X11_ShowWMCursor;
        device->WarpWMCursor = X11_WarpWMCursor;
        device->CheckMouseMode = X11_CheckMouseMode;
        device->InitOSKeymap = X11_InitOSKeymap;
        device->PumpEvents = X11_PumpEvents;

        device->free = X11_DeleteDevice;

        return device;
}

```

#### 【リスト8】mk\_cross4ps2.sh

```

#
# mk_cross4ps2.sh
#

RPMS=/work/files/dev/ps2/cross4ps2
PREFIX=/usr/local/cross_ps2

MAKE=make
TARGET=mipsEEel-linux

CURRENT=`pwd`

echo %%%%%%%%%%%%%%%%%%%%%%%%%%
echo % seting binutils ... %
echo %%%%%%%%%%%%%%%%%%%%%%%%%%

mkdir binutils; cd binutils

echo unpacking...
rpm2cpio $RPMS/binutils-2.9EE-3.src.rpm | cpio -iv
tar xzf binutils-2.9EE.tar.gz
cd binutils-2.9EE
patch -p1 -N -E -s < ../binutils-2.9EE.linux.patch
patch -p1 -N -E -s < ../binutils-ps2linux-0.9.patch

echo build...
mkdir obj; cd obj
../configure --prefix=$PREFIX --target=$TARGET

$MAKE tooldir=/usr
$MAKE tooldir=/usr info

echo install...
$MAKE install install-info

echo %%%%%%%%%%%%%%%%%%%%%%%%%%
echo % seting gcc ... %
echo %%%%%%%%%%%%%%%%%%%%%%%%%%
cd $CURRENT
mkdir gcc; cd gcc

echo unpacking...

```

(次ページに続く)

【リスト8】 前 ページの続き

```

rpm2cpio $RPMS/gcc-2.95.2-3.src.rpm | cpio -iv
tar xzf gcc-2.95.2.tar.gz
cd gcc-2.95.2

echo patched...
patch -p1 -N -E -s < ../gcc-2.95.2.EE-linux.patch
patch -p1 -N -E -s < ../gcc-ps2linux-0.9.1.patch

cd gcc/config/i386
cat <<< EOM |patch
--- libc.so.orig      Sun May  6 18:47:38 2001
+++ libc.so           Sun Sep 16 00:02:22 2001
@@ -1,4 +1,4 @@
/* GNU ld script
   Use the shared library, but some functions are only in
   the static library, so try that secondarily.  */
-GROUP ( /lib/libc.so.6 /usr/lib/libc_nonshared.a )
+GROUP ( libc.so.6 libc_nonshared.a )
EOM

mv gcc-lib ..
cd ..
tar cf - include lib | \
    ( cd $PREFIX/$TARGET && tar xf - )
mv $PREFIX/$TARGET/include/g++-3
mv $PREFIX/include/g++-3
tar cf - gcc-lib/$TARGET/2.95.2/include | \
    (cd $PREFIX/lib && tar xf -)
ln -s ../../../../../../$TARGET/lib/libstdc++-3-libc6.2-2-2.10.0.a \
    $PREFIX/lib/gcc-lib/$TARGET/2.95.2/libstdc++.a
ln -s ../../../../../../$TARGET/lib/libstdc++-3-libc6.2-2-2.10.0.so \
    $PREFIX/lib/gcc-lib/$TARGET/2.95.2/libstdc++.so
ln -s ../../../../../../$TARGET/lib/soft-float/libstdc++-3-libc6.2-2-2.10.0.a \
    $PREFIX/lib/gcc-lib/$TARGET/2.95.2/soft-float/libstdc++.a
ln -s ../../../../../../$TARGET/lib/soft-float/libstdc++-3-libc6.2-2-2.10.0.so \
    $PREFIX/lib/gcc-lib/$TARGET/2.95.2/soft-float/libstdc++.so

echo %%%%%%%%%%%%%%%%%%%%%%%%%%
echo % seting kernel-header%
echo %%%%%%%%%%%%%%%%%%%%%%%%%%
cd $CURRENT
mkdir kernelheader; cd kernelheader

echo unpacking...
rpm2cpio $RPMS/kernel-headers-2.2.1_ps2-6.mipsel.rpm | \
    cpio -iv --make-directories

cp -fR usr/src/linux-2.2.1_ps2/include/linux \
    $PREFIX/$TARGET/include
cp -fR usr/src/linux-2.2.1_ps2/include/asm* \
    $PREFIX/$TARGET/include

echo %%%%%%%%%%%%%%%%%%%%%%%%%%
echo % seting libps2dev ...%
echo %%%%%%%%%%%%%%%%%%%%%%%%%%
cd $CURRENT
mkdir libps2dev; cd libps2dev

echo unpacking...
rpm2cpio $RPMS/libps2dev-0.9-1.mipsel.rpm | \
    cpio -iv --make-directories

cp -fR usr/include/* $PREFIX/$TARGET/include
cp -fR usr/lib/*     $PREFIX/$TARGET/lib

echo %%%%%%%%%%%%%%%%%%%%%%%%%%
echo % seting done      ...%
echo %%%%%%%%%%%%%%%%%%%%%%%%%%

```