Linux/Alpha 活用講座

清水 尚彦　nshimizu@keyaki.cc.u-tokai.ac.jp

第22回
Alphaアーキテクチャに最適化したLinuxカーネルパッチの作成(1)

今月の動向

いやぁ、参りました。モノクロ画面の「Mobile Gear II」に加えて、画面の大きなシャープの「Telios(テリオス)」というWindows CEマシンを購入していたのですが、今回の原稿執筆に使ってみようと思って、使い始めてみました。ところが、ファミリーレストランでモーニングを食べながら8割ぐらい書いたところで、画面がブラックアウト。小まめにファイルをセーブしているにもかかわらず、不吉な予感が的中して、フルリセットがかかり、原稿はすべて消え去ってしまいました。

Microsoft社のOSは信頼していないので、いつもは原稿はフラッシュメモリに保存するように気をつけているのですが、たまたま今回だけ、フラッシュメモリを持ち歩いていなかったので(いつもの鞄ではなかったため)、Windows CEの本体メモリに記憶させていたのが災いしました。

悪いことは最悪の状況で発生する、肝に命じておきましょう。電池が危ういのなら警告を出してサスペンドすればいいのに。こんなことなら、とんでもなく使いにくいキーボードと、タッチパネルがない操作性の悪さを我慢して使ったりしなければ良かったと、深く反省。

こういった出来の悪い機械を使うと、開発者は自分で使うことなんて考えていないのだろうとよく思うのですが、不思議ですね。作った人間に「この機械で原稿用紙30枚分の原稿を2日で打ち込んでみろ」と強要したくなります。なんて文句を言っても消えた原稿が戻ってくるわけでないので、気を取り直して書き直しましょう。

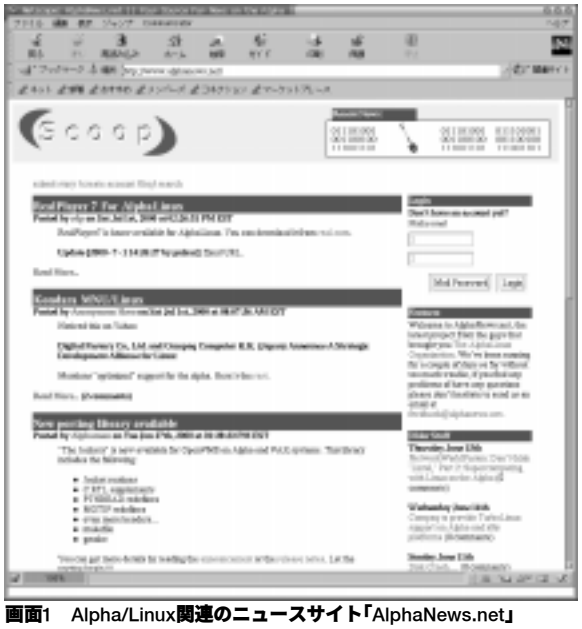
●Alpha関連のニュースサイトがオープン

まずは、Alpha関係の新しいサイトの報告です。ニュース速報のようにAlphaの記事をまとめたサイト「AlphaNews.net」(記事末RESOURCE[1]を参照、画面1)がオープンしました。

●スーパーコンピュータTop 500

さて、例年、6月と11月に、スーパーコンピュータの処理性能上位500サイトのリストが発表されるのですが([2]、画面2)、今回も、やはりAlpha Linuxのサイトがいくつかランクインしています。

Alpha Linuxではないのですが、CompaqのAlphaServer SCもリストに登場しています。プロセッサ台数あたりの性能で



画面1　Alpha/Linux関連のニュースサイト「AlphaNews.net」

みると、IBMのSP Power3の性能とほぼ同じくらいになっていますね。そのほかに、日立のSR8000も、今回初登場でトップ10に入るなど、なかなか検討しています。

Alpha Linuxのトップは、Sandia National Laboratoriesの「CPlant Cluster」です。彼らは、XP1000の500MHzのプロセッサをノードとしていたはずですが、580プロセッサで232.6GFLOPSを叩きだして堂々62位にランクインです。これは、東大のSR2200をわずかに抜いています。Los Alamosの「Avalon」は、48.6GLOPSで364位になっています。最近のトップ500は、要するに台数を増やせばいいというものになってきたので、ちっとも面白くありませんね。CPlantも今年中に1400台に増強するといっていますが、増やせば性能が上がるのは当然ですから、このリストそのものが興味を引くものではなくなっています。

仮に誰かが、トップ500に入るサイトを作りたいと思ったら、一番簡単なのは、CrayのT3Eを購入することです。Crayはもう開発をやめたのかと思っていたら、案外としぶとく、今年の後半には、T3Eの拡張版を発表するとしていますし、次世代機SV2も、2002年には発表するとしています。SV2の発表を見て、新たなベクトル機を開発するだけのパワーがあるのかと、私は実は危ぶんでいたのですが、Crayはいくつかの政府機関から開発費をもらっているようですね。それはともかく、今年後半に発表されるとするT3Eの拡張版がどのようなものになるのか、興味深いですね。

画面2　スーパーコンピュータTOP 500サイト

Alphaアーキテクチャに最適化したLinuxカーネルパッチの作成

●Alphaのユニークなメモリ管理機構

さて、ここからが本題です。原稿が消えたショックが大きくてなかなか立ち直れないのですが、がんばって書きましょう。

Alphaのアーキテクチャには、メモリ管理に関してユニークな機構が備わっています。実は、命令セットとしてもプリフェッチを行う命令が定義されているなど、アーキテクチャの設計者が、メモリ性能がプロセッサの性能に与える影響によく気をつけていることが分かります。残念ながらこれらの機構や命令は、これまで、True64においてすら実装されていません。プリフェッチの専用命令は、Crayでは別の意味に使っている可能性が高いのですが、コンパイラやライブラリの設計者から見ると、この命令仕様は、必ずしも使いやすくないハードウェアの都合だけを優先するようなものになっていたことも確かです。

一方、メモリ管理のユニークな機構とは、ページテーブルの各エントリにセットする「Granuality Hint(GH)」というビットです。このヒント情報があってもなくても、プロセッサは正しく動作しなくてはならないし、ソフトウェアもこの情報をプロセッサが使うかどうかを仮定せずに使っても使わなくても正しく動かなくてはならないという制限がついています。ヒント情報は2bitの情報で、ページテーブルエントリのbit 6とbit 5にセットします。この2bitで表される数値が0、1、2、3となっているページテーブルエントリは、そのエントリが示すページフレームを1、8、64、512ページ境界のアドレスから連続する1、8、64、512ページが同一のページ保護情報を持っていて、ハードウェアは、このページのブロックを1つの大きなページとみなして処理をしても構わないと定義されています。こちらの機構も、UNIXの普通のページ管理とあまり相性がよくなく、今まで使われずにきています。

ところが、高性能なアプリケーションを作ろうとすると、キャッシュのミスはブロック化やプリフェッチで解消することができても、TLBのミスを解決するのはかなり難しいのです。というのは、キャッシュのエントリは(キャッシュ容量が大きくなってきたこともあって)かなりの数が備わっているのが普通ですが、TLBのエントリはあまり多くないのが最近のRISCプロセッサの常だからです。例えば、21264には128エントリのデータTLBがあるのですが、1エントリで8Kbytesのページをマップするので、全部のTLBを使っても1Mbytesの空間しかマップできません。これに対して、例えば1000×1000

の倍精度の浮動小数点の行列を定義すると、それだけで8Mbytesの空間が必要なわけで、TLBがいかに小さな空間しかサポートしていないかが分かると思います。

私はこういった実装はあまり気に入らないのですが、自分で作っているわけではないので文句も言えません。しかも、キャッシュに入るワーキングセットであっても、アドレスの連続性がなければTLBをミスするということになります。先ほどの行列を列方向に参照するプログラムを考えてみると、1列を全部参照してもキャッシュブロックサイズを64bytesとして64Kbytesのワーキングセットになり、21264の1次キャッシュに入ってしまいますが、TLBのエントリは、ほぼ1000個必要になるので、TLBのエントリ数は全然足りません。TLBのミスが起きたときには、ページテーブルの参照をして本当にページフォルトを起こすべきかどうかを決定しなくてはなりません。ページフォルトが起きる場合には、正確な割り込みを発生させなくてはいけませんから、多くのプロセッサの実装では、TLBミスはパイプラインをストップして割り込みを発生させます。そこで、プリフェッチのように十分時間をおいてメモリを先読みしようとしても、プリフェッチがTLBミスを起こすと、結果として割り込みルーチンが起動して大きなオーバーヘッドになってしまい、先読み効果は生まれません。これを解消するには、ページ参照をハードウェアで実行して、その間にも後続の命令を投機的に実行する必要があります。

Alphaの今の世代では、ページ参照はPALコードで実行されています。これは、おそらく将来的にもあまり変わらないと思います。というのは、アーキテクチャとして、ページテーブルの実現方法がOSごと複数定義されていて、その違いをPALコードで解消しているからです。OpenVMSとOSF/1(Linuxはこちらと互換)はあまり変わらない(それでも少し違います)のですが、Windows NTに至っては、全く異なるページ管理の方式になってしまいます。Windows NTのサポートを止めたのであれば、残りの2つはあまり違いはないので、ハードウェアでほとんど対応できそうな気がしますが、現行のプロセッサではそうはしていないはずです。

そこで、TLBミスはパイプラインを止めるのみならずPALコードへの割り込みを発生させるため、非常に長い時間がかかりますし、あらかじめプリフェッチを起動してもオーバーヘッドは全く隠蔽できません。TLBのミスをなくすには、変換しないという方法が1つ考えられます。初期のスーパーコンピュータは、アドレス変換の機構を持たずに、実アドレスで処理をしていました。これはベクトルプロセッサの設計者のスキルの問題もあったかもしれませんが、高性能計算では、基本的にメモリ常駐のプログラム実行を必要とするため、実メモリであまり困らなかったという読みがあったのでしょう。ただし、値段の高いマシンを1人で占有する使い方しかできないと、ユーザーからはあまり評判がよい方法ではないはずです。

Alphaでも、カーネルモードで実行するときにアドレス変換を抑止することが可能です。いわゆる「kseg」のアドレス空間のアクセスは、アーキテクチャでアドレス変換をしないことになっています。これならTLBを使わなくてもいいのですが、ユーザーのプログラムをksegの空間で実行するのはあまりに危険なので実用的ではありません。もちろん、MS-DOSのように、メモリ管理を諦めてユーザーに任せるというやり方もあってもよいのではと思いますが、割り切りが必要ですね。CrayのUNICOSはどうなっているのでしょうか？　昔のベクトル機は、アドレス変換をしなかったと思うのですが、UNIX互換をうたっている最近のUNICOSではどうなのでしょうね。

●GHビットの活用

さて、ksegを使わずにTLBミスを減らすことを考えたい場合には、先ほどのGHビットを使って、マップ可能な領域を増やすことで対応できます。GHビットを最大限に使うと、1エントリ当たり512ページがマップできるので、21264では、512Mbytesまでのメモリ領域をTLBでマッピングできることになります。これでも足りない人たちも多いかと思いますが、少なくとも、キャッシュに入るのにTLBでミスするから性能が出ないという情けない事態は避けられます。せっかくの機構なのに、現在のLinuxでは全く使っていませんし、True64にしても、おそらく使っていません。ハードウェアの設計者からみると自然な実装のように見えるのですが、UNIX系のシステムのメモリ管理の方式と合わない部分が多いからです。

というわけで、今回と次回の2回に分けて、このGHビットをLinuxに組み込む実験的なカーネルパッチを作ってみましょう。GHビットのサポートを組み込むには、当然ながら、Linuxのメモリ管理を理解する必要があります。そこで、Linuxのカーネルを読むことから始めていきたいと思います。対象とするカーネルは、執筆時点で最新である「2.4.0-test1」とします。カーネルの中心部分はあまり変化しないほうがいいのですが、2.2.xとそれ以降のカーネルでメモリ管理の部分は大きく変わっています。安定版でないと動作確認できる環境を整えることが難しいのですが、新しいメモリ管理の機能を使うと、目的とする実験システムの構成が簡単だということがあって、こちらを使うことにしました。

●カーネルソースを読むための準備

まずは、カーネルソースを読むための環境構築からです。linux.2.4.0-test1.tar.gz(本誌2000年8月号の付録CD-ROMに収録)を、適当なディレクトリに展開します。展開してできる「linux」というディレクトリの下ですべての作業を行います。まず、メモリ管理の関連ファイルをすべてプリントアウトしましょう。この記事で参照するファイルは以下の通りです。

```
mm/memory.c
mm/mmap.c
mm/mlock.c
mm/page_alloc.c
arch/alpha/mm/init.c
arch/alpha/mm/fault.c
include/linux/mm.h
include/asm-alpha/page.h
include/asm-alpha/pgtable.h
include/asm-alpha/pgalloc.h
```

次に、エディタで簡単に参照できるように、タグファイルを作りましょう。タグファイルを作るコマンドはエディタによって違いますが、vi系列のエディタを使う人はctagsを使います(実行例1)。

私が普段常用しているnvi-m17nは、タグジャンプができなかったので、elvisをインストールしました(viでタグジャンプができないなんて、文句が出ないわけがないので、おそらくx86版nviではできるのでしょうね。このあたりにも、どうやら64bitサポートの不具合がありそうです)。

例えば、「alloc_page」という関数を探すには、

```
$ elvis -t alloc_page
```

とします。すると必要なファイルを開いて関数の定義のところにカーソルが点滅します。エディタの中で、さらにタグジャンプしたいときには、2つの方法があります。1つはexコマンドを使う方法です。

```
:tag alloc_pages
```

と入力すると、対象の関数にタグジャンプします。また、エディタで見えている部分に関数をコールしているところがあれば、その関数名のところまでカーソルを進めて「Ctrl+]」(コントロール+閉じブラケット)を入力します。一度ジャンプして元に戻りたいときには、「Ctrl+t」を入力してください。ジャンプする前にいた場所に戻ることができます。

写真1　Linux Core Kernel Commentary

さて、タグジャンプとともにプログラムを読む場合に便利なものがクロスリファレンスです。有名なUNIXの解説書である「Lions 'Commentary on UNIX」([3])には、プログラムに先だってクロスリファレンスリストがついています。Linuxでもクロスリファレンスを作成するツール「cxref」があるのですが、少し試してみた感じでは、カーネルのリファレンスを出すのは少し難しいようです。Scott Maxwellの「LINUX Core Kernel Commentary」と([4]、写真1)いう本には、Linuxのカーネルソースコードとコメントが出ていて、Lionsの本のLinux版のように書いてありますが、これにもクロスリファレンスがついていません。紙の上でのリファレンスを諦めて、必要ならgrepコマンドで探すことにしましょう。カーネルのソースツリー全体を探すときには次のようにします。

```
$ find . -name \*.c -exec grep alloc_page {} \;
-print
```

すると、今いるディレクトリから下のすべての.cのファイルを探して、grepをかけてくれます。こうすると、クロスリファレンスがなくても何とかオンラインではやっていけます。

もちろんクロスリファレンスがあるほうが便利に決まっているので、cxrefを動かすために「cxref-cc」にパッチを当てました。これが動作しない一番の原因は、シェルコマンドで

実行例1　ctagsコマンドでタグファイルを作成

```
$ ctags mm/* arch/alpha/mm/* include/linux/* include/asm-alpha/*
```

あるcxref-ccが、渡されたオプションを解釈するときに"(ダブルクォート)の扱いが難しいからデリミタを"^M"に変更したことに由来しているようです。デリミタの変更がきちんと動作していないので、ファイルが見つからないとエラーになってしまいます。

もう1つは、cxref-ccに渡すオプションは.cxref というファイルに書き込んでおくのですが、makeのカレントディレクトリを変更したときにこのファイルを見失ってしまうのです。これはすべてのディレクトリに.cxrefをリンクしておくことで対応します。この辺り、スクリプトを書き換えたりといろいろやって何とかできたのですが、きれいな方法ではないので公開はためらわれます。しかも出来上がったファイルはTeXのキャパシティエラーで作成に失敗したので、HTMLで作り直しました。これを「html2ps」コマンドでPostScriptに変換すれば、印刷できるものになるのですが、かなりの分量になるので、できたことだけ確認しただけです。そもそもカーネル全体のクロスリファレンスなんて必要ないのですが、細かな調節はツールを使いこなさないとなかなか難しいものです。

とにかくこれで一通りの準備は整ってきたので、あとはカーネルのソースをじっくり眺めればLinuxがどんなメモリ管理方式をとっているか分かるという仕組みです。

●Linuxのメモリ管理

ここで、Linuxのメモリ管理方法をざっと概観してみましょう。カーネルのソースを読む場合に、闇雲に端から読んでいっても難しいものがあります。ある程度の動作のイメージを持って、その流れに沿って読んでいくために、大体の概要を知っておく必要があります。

・mem_map

プロセッサの実記憶をメモリマネージャが扱うページ単位に分割して管理します。管理のために「mem_map」という大域配列を利用します。この配列は、page構造体を要素としています。page構造体の詳細は「include/linux/mm.h」に定義されています。この構造体はかなり大きくて、100bytes以上になります。そこで、2Gbytesのメモリを持つシステムがあるとすると、この管理用の配列だけで20Mbytes以上、メモリが必要になります。もっとも、そのおかげでページフレーム番号を知るだけで管理情報にマップできるので、やっぱり必要なのでしょうね。ページテーブルの場合には、プロセスごと必要だから、このようなリニアな配列は使いものにならないのですが、システムに1つしかない実メモリ管理テーブルだから許されるのでしょう。

・エリア単位での管理

実記憶は、10個のエリアと呼ばれるブロックごとに分けて管理されます。1つのエリアに属するブロックの大きさは、エリアの番号をNとすると、2のN乗にページサイズをかけたものになります。実メモリが要求されたときには、要求メモリの大きさを超えるエリアの管理リストから最小のエリアのブロックを割り当てます。後で述べるように、普通のユーザーは1ページ以上のメモリを要求することはないので、エリアの管理は、カーネルのデバイスドライバなどで特に大きなメモリを要求する場合にだけ有効なものです。しかし今回のパッチでは、この部分をうまく活用しようというものです。

・current

実行中のプロセスの状態は、currentと呼ぶtask_struct構造体の変数に管理されています。この中に、メモリ管理の構造体であるmm_structのメンバとして、mmという変数があります。プロセスのメモリ管理に関する情報はこの変数からたどっていきます。

・mmap

プロセスの仮想空間の情報は、同じくmmの中のmmapというメンバから連続する仮想アドレスのブロックごとにvm_area_struct構造体に情報が記憶されます。この連続というのは、アドレスの連続だけでなく、管理情報が等しいことも必要です。あるアドレスのブロックがテキストセグメントで読み込みのみとなっていて、それと連続するアドレスに書き込み可能なデータセグメントがある場合には、それらのブロックは別々のブロックとして登録されなくてはなりません。一方、メモリ管理の途中でアドレスを確保したり開放したりすることがあるのですが、その場合には、積極的に隣接するブロックとマージ可能かどうかを確かめて、できるだけ大きな単位で管理が行われるようにしています。

・forkの仕組み

プロセスが新たに生成される場合には、forkシステムコールが呼ばれます。このシステムコールは、呼び出したプロセス(親プロセス)のコピー(子プロセス)を作成するものです。コピーを作ると言っても、子プロセスは通常すぐにexecvシステムコールで親のメモリを捨ててしまうので、本当にメモリのコピーをしても無駄になります。そこで、UNIX互換の実装では、子プロセスのメモリは実際にメモリの参照を行うまでは割り当てないということが行われます。これは「Copy On Access」というのですが、このときにも、まだメモリをコピーする必要

があるかどうかの判断を保留しておきます。というのは、親も子も、メモリを読むだけならば、別のメモリを割り当てずにページテーブルエントリ(PTE)の中のページフレーム番号(PFN)を一致させておくだけでよく、その方がはるかにオーバーヘッドは少ないからです。ただし、親や子がforkの後で勝手に値を書き換えると困るので、親と子の両方のページフレームを書き込み禁止にしてしまいます。こうしておいて、どちらか片方のプロセスが書き込みをしようとすると、アクセス例外の割り込みが発生するようにします。アクセス例外の割り込みハンドラでは、発生した割り込みが本当にユーザープロセスが権限のないアドレスにアクセスしたのかどうかをvm構造体のアクセスフラグを見て判定し、アクセス権限があるのにページの保護がされている場合には、「Copy On Write」のリクエストと判断して新たにメモリのページを取得した上で、当該ページをコピーします。そうして書き込み禁止を解除してプロセスを再スタートさせることによって、プロセスは固有のデータ領域を持つことになります。

・メモリ領域の確保

プロセスが実行上の都合で、新たにメモリ領域を要求する場合があります。これには、代表的な2つのCの関数が使われます。

1つは「brk」で、これはシステムコールとしても用意されています。brkは、未使用のヒープ領域の上限値を設定する関数ですが、brkそのものは使いにくいので、このラッパ関数として「sbrk」が用意されています。

もう1つの関数の方が一般的に使われるのですが「malloc」と呼ばれる関数です。この関数は、glibcではmmapシステムコールを呼び出すような実装になっています。もちろん、本来のmmapもメモリ領域の確保のために利用可能です。これらのシステムコールが呼び出されたときには、カーネルはまずプロセスの仮想アドレスの拡張を行います。これは、前出のmmで管理されるvm_area_strut構造体のしかるべき場所に新たな構造体を追加するか、既存の構造体のマップする仮想アドレスの開始や終了のアドレスを変更することで行われます。プロセスが普通にこれらのシステムコールを呼び出したときには、mm以外の変更はなされません。というのは、もともとマップされていない仮想アドレスに相当するページテーブルのエントリは無効とされているはずで、Copy On Accessと同様の考え方を踏襲すると、実際にアクセスにくるまではメモリを割り当てる必要はないということになるからです。mmに有効なアドレスが正しく設定されていれば、割り込みハンドラは、本来アクセス可能なところからのアクセス例外と判断できるので、そのときに初めてメモリを割り当ててあげればよいということになります。

ただし、この部分に関しては、例外的な扱いをする場合があります。それは、mlockシステムコールによってメモリがロックされている場合です。ロックして常駐することが求められているプロセスは、メモリを頻繁にアクセスするつもりがあるのですから、こんな回りくどいことをせずに、最初からページを割り当てればいいということになります。そこで、ロックされているときにはページを割り当てるのですが、ここでもこのvm領域が書き込みを許可しているのでなければ、本当に新しいページを割り当てるのではなく、中身が0であるページがカーネルの中に用意されているのですが、そのページフレーム番号を持つPTEをプロセスのページテーブルにセットするだけです。書き込みがなされる場合には、ようやく新たにページを割り当てます。

●実験的GHサポートの方針

以上のような仕組みでLinuxはメモリを管理しているわけですが、見てきたように、ページの大きさを超えたブロックの概念を導入するチャンスはあまりないように見えます。というのは、実ページの割り当ては最後の最後までなされないという方針なので、アクセス例外やページ例外があったときだけしか実際にメモリを割り当てるタイミングはないわけで、このときには、まさに例外を起こしたそのものズバリのページしかカーネルは認識できないからです。これらの事情は、True64でもそれ以外のOSでも大差ないものと思われます。Linuxのカーネルツリーを注意深く探していくと、「富士通のAP1000では連続したDMA領域が通信のために必要になる」という記述がありますが、これも、デバイスドライバレベルでよければ、前出のエリアを使えば簡単に割り当てができます。しかし、ユーザーのアドレス空間に大きなブロックを導入する必要のあるGHのサポートとは異なります。

カーネルに大規模に手をいれてGHのサポートをする方針はできていますが、面倒くさがり屋の私は、大規模に手を入れたくないので、とりあえずGHのサポートを実験的に行うことにしてみます。実験的なサポートが通常のプログラムの動作の邪魔をしたらいけないので特定条件でしかこの機能が働かないようにします。

・ページを常駐させる

mlockallシステムコールでMCL_FUTUREのオプションが指定されて、新たな割り当てメモリが常駐される場合だけを対象にします。というのは、GHビットが立っているページがス

ワップアウトされたりすると、ブロック内でPTEがばらばらになって正しく動作しない可能性があるわけで、簡単に済ますためには、スワップされないようにしておかなくてはなりません。それには、ページを常駐させるのが手っ取り早いのです。ただし、このシステムコールは、rootからしか呼び出しできないため、GHの実験はroot権限を持つユーザーしかできなくなります。

・brkシステムコールにのみ対応

メモリ管理で、普通のプログラムがあまり使わないbrkシステムコールからメモリを要求された場合にだけ、GHのサポートをします。これによって、例えmlockを立てたプロセスが動いていたとしても、まずGHのページが勝手に割り当てられる可能性は少なくなると思っています。もっともmlockが立っていれば、GHのページを割り当てて困ることはないような気もするので、このあたりは要検討ですね。

●パッチの解説は次号

さて、これらのサポート方針で2.4.0-test1用のパッチを作成したのですが、詳細に関する話は次号に回します。しかし、すぐにでもパッチの中身を見てみたいという人もいるでしょうから、以下のURLにパッチとドキュメントを置いておきます。ご参照ください。

パッチの大きさは、圧縮しない状態で約9Kbytes程度と、ごく小さなものです。しかも、そのうちのかなりの部分はドキュメントだったりします。ただし、今のパッチは若干問題点があって、ロックされたページがリリースされないようなのです。プロセスが終了するときにメモリを全部開放しなくてはいけないのですが、どうやらうまく開放されていないように見えます。この辺り、はっきりしたことは分かりません。もしかしてパッチのせいではなくて、もともとのカーネルのバグの可能性が高いのではないかと思ったりもしています(その後の調査でパッチの不具合が判明しました)。

```
http://shimizu-lab.et.u-tokai.ac.jp/~nshimizu/
```

また、安定していない新しいバージョンのカーネルを使っているので、いろいろとトラブルが多く発生します。特にパッチ以前の問題が多くて苦労します。

例えば、この実験をするためのシステムですが、Debian/potatoには必要なバージョンのツールがそろっているので、これをベースに、先月号に解説した例のディスクレス設定にしようと考えました。ディスクレスなら、ハングアップしても何が起こっても困らないと思っていたのですが、なぜかディスクレスでうまく動いてくれず、結局、ローカルディスクをつないでDebian/woodyを導入しました。

こちらでは、ワーニングが出ながらでもきちんと動くのですが、何か起きると次の起動のときにfsckが動作するので、時間がとられてしまいます。なかなかうまくいかないですね。

また、DP264のマシンは、カーネルの起動の途中でSCSIドライバのパーティションを認識する辺りでエラーになるようで、きちんと動いていません。こちらも時間の都合で詳しい調査は進んでおりません。申し訳ありません。

最後に、このパッチの作成に時間がとられていて、あまりその他の作業が進まないのですが、ビジュアルテクノロジー社から、DP264のSMPシステムを長期借用できました。誌面を借りてお礼申し上げます。せっかくなので今後このシステムを使った評価などもやってみたいと思います。今は、私は時間が取れないので、研究室の学生たちが寄ってたかってKondara MNU/Linuxの設定をいろいろ試しているようです。このシステムは3Gbytesのメモリがあるのですが、通常のカーネルでは2Gbytesしか認識しないので、この辺りも早急に手を入れて3Gbytesをフルに使えるようにしたいものです。

R E S O U R C E

[1] AlphaNew.netホームページ
http://www.alphanews.net/

[2] スーパーコンピュータランキング
http://www.top500.org/lists/TOP500List.php3?Y=2000&M=06

[3] 「Lions' Commentary on UNIX」John Lions 著
邦訳は、岩本信一 訳でアスキーから。
価格3800円、ISBN4-7561-1844-5
http://www.ascii.co.jp/books/detail/4-7561/4-7561-1844-5.html

[4] 「LINUX Core Kernel Commentary」Scott Maxwell著
邦訳は、小嶋隆一 訳でセレンディップから刊行予定。
価格8925円(税込)、ISBN4-7978-2006-3
http://www.serendip.co.jp/