社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－ARC－176 (5)
2008－EMB－ 7 (5)
2008／1／15

# 2.6系カーネルに対するLinux Super Pageの実装と性能評価

木下 修平 †, 清水 尚彦 ††

† 東海大学大学院 工学研究科 †† 東海大学 情報理工学部

メモリを多く使用するアプリケーションはTLB(Translation Lookaside Buffer) ミスが原因で高速に実行できない場合がある．最近のプロセッサはTLBの有効範囲を拡張するために複数のページサイズをサポートしているが、多くのオペレーティングシステムは固定ページサイズまたは1プロセスで1つのページサイズのみをサポートしている．我々はユーザプログラムから不可視な形で要求されたメモリサイズに対し、動的に複数のページサイズを割り当てるSuper Pageを Linuxに実装することで、TLBミス削減によるアプリケーション性能の向上を図る．

# An Implementation of Linux Super Page on Linux Kernel 2.6 and Performance Evaluation

Shuhei KINOSHITA † Naohiko SHIMIZU ††

† Graduate School of Engineering, Tokai University

†† School of Information Technology and Science, Tokai University

We might not be able to execute applications which use a lot of memory on high speed due to the TLB misses. Recent processor has supported two or more page sizes to expand the TLB coverage. But most operating systems support only the fixed page or only one size in each process. Therefore, we implement Super Page on Linux. Super Page dynamically allocates two or more page sizes for the memory size which demanded from user program. As a result, we reduce the TLB misses and improve the application performance.

## 1 はじめに

近年、アプリケーションの大規模化やオペレーティングシステムのサポート要求に伴い、従来より増して多くのメモリを要求するようになった．メモリを多く使用するアプリケーションではTLBのマッピング不足が性能低下を引き起こす場合がある．TLBは仮想アドレスとメモリにあるアドレス変換テーブルを用いて算出した実アドレスの変換結果を蓄えておき，次回からのアドレス変換を高速に行うためのバッファである．アクセスした仮想アドレスに対する適切な実アドレスがTLBエントリにないことをTLBミスと呼び、これは余分な処理サイクルを発生させる．TLBのエントリを増やすことはプロセッサの設計上困難であるため、近年のプロセッサは複数のページサイズをサポートすることでTLBの有効範囲を拡張している (表1)．このため、以前より本研究室ではLinux Sper Pageプロジェクトを行ってきた [1) 2) 3)]．本稿では、今回我々が実装した2.6系Linux Super Pageカーネルについて、その設計方針や実装方法について述べる．またSuper Page実装後に行列転置ベンチマークプログラムを用いて行ったオリジナルカーネルとのメモリ転送性能比較を示す．

Table 1 ページサイズ

| CPU | Page Size |
|---|---|
| Alpha | 8K, 64K, 512K, 4MB |
| Sparc64 | 8K, 64K, 512K, 4MB |
| IA32 | 4K, 4MB |

## 2 従来技術

これまでにSGI社製のIRIXやHP社製のHP-UXといった商用オペレーティングシステムが複数ページサイズをサポートするSuper Pageの機構を持つと報告されている．しかし、これらのオペレーティングシステムではユーザが適切なページサイズを設定する必要があるなど問題点がある [2)]．
また、最近では2.6系Linuxカーネルに HugeTLBファイルシステムが導入され、プロセッサのサポートするラージページを利用することができるようになった．Linuxはシステム起動時に HugeTLBファイルシステム専用のメモリ領域の確保を行い、確保したメモリ領域をフリーリストで管理する．ユーザは疑似ファイルシステムであるhugetlbfsを通し

て HugeTLB 機能を利用でき、プロセスがメモリを要求した際、ラージページのフリーリストからメモリを割り当てる．図 1 に HugeTLB ファイルシステムの概念図を示す．

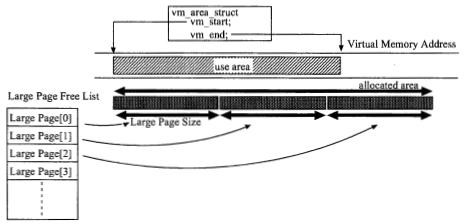



Fig. 1　HugeTLB ファイルシステム概念図

HugeTLB の導入により多くのメモリを利用するアプリケーションの性能向上が期待できるが、以下の問題点も存在する．

- 移植性のないプログラム
  特殊な hugetlbfs をマップする必要があり、HugeTLB 用に記述されたプログラム以外は利用することができない．このため HugeTLB ファイルシステムを利用するプログラムに移植性はなく、利用する場合は HugeTLB 用にプログラムを書き直す必要がある．
- スワップ
  HugeTLB ファイルシステム上のメモリは常に物理メモリに配置され、スワップ対象外となる．このため、ラージページ用に確保したメモリを使い果たした場合、プロセスはアプリケーションの実行を終了させる．

このように、いくつかのオペレーティングシステムでは、Super Page を利用するための仕組みが提供されているがユーザにとって利便性の良いものではない．このため、我々は実用的で透過性のある Super Page 機構を Linux に実装する．

## 3　Linux のメモリ管理方法

### 3.1　仮想メモリ管理

図 2 に Linux における仮想メモリ管理の概要を示す．

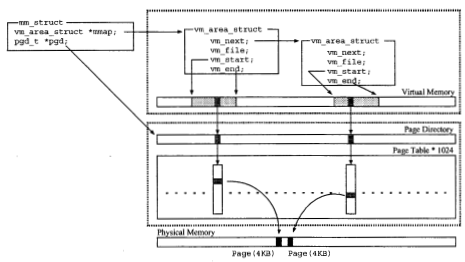



Fig. 2　Linux の仮想メモリ管理

各プロセスは仮想アドレスをマップするために'mm_struct' 構造体を持つ．'mm_struct' 構造体は仮想アドレスを実アドレスへ変換するための重要な情報を持つ．そして仮想アドレスは'mm_struct' 構造体からリンクされている'vm_area_struct' 構造体によってマップされ、アドレス変換はページディレクトリを用いて行われる．実際の IA32 アーキテクチャにおけるアドレス変換を図 3 に示す．32 ビットの仮想アドレスは PGD、PTE、ページオフセットの三部分に分けられ、物理アドレスを得るために PGD と PTE を用いて 2 段階のページ変換を行う．Linux ではアドレス変換の実装部分を図 4 に示すような 4 段のページ変換テーブルによりモデル化している．このため、IA32 アーキテクチャを扱う場合は変換テーブルの PUD、PMD のエントリ数を 1 とし、さらにその参照先を PGD の参照先とすることで 2 段階のアドレス変換を 4 段階に見せかけている．

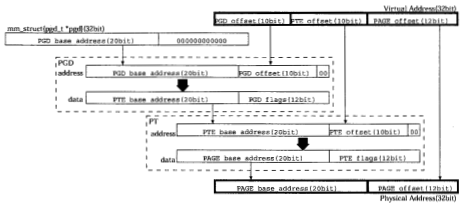



Fig. 3　IA32 の仮想アドレス変換

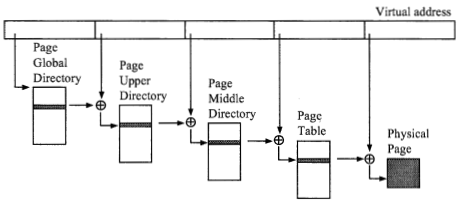



Fig. 4　Linux のページ変換テーブルモデル

### 3.2　物理メモリ管理

利用可能な実メモリはバディシステムによって空きページとして管理される．バディシステムは空きページを 2 のべき乗単位 (オーダー) で管理し、各オーダーのページ境界に合う場合はより大きなオーダーで管理する．図 5 にバディシステムによるページ管理の概要を示す．

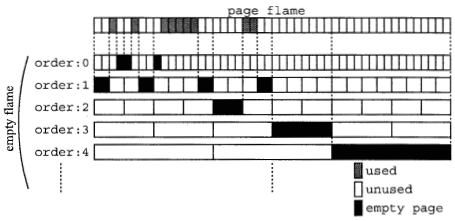



Fig. 5　バディシステムによる空きページ管理

例えばバディシステムが16ページ分の連続した空きページの要求を受けると、オーダー4のグループに未割り当ての空きページを探しにいく．そして空きページがあれば、その先頭ページのアドレスを返し、ない場合はオーダー5のグループから32ページを2つの連続した16ページに分割し、片方の先頭アドレスを返す．このように未割り当ての空きページを見つけるまで順にオーダーの大きいグループへ探しにいく仕組みとなっている．

### 3.3 メモリ割り当て

ユーザプロセスからのメモリ領域の確保要求が発生するとカーネルは'vm_area_struct'構造体を新しく生成し、これを'mm_struct'構造体のリンクに追加することでメモリ領域の確保を行う．Linuxでは不要な物理メモリの使用を防ぐ仕組みを持つため、実際に要求されたメモリ領域に対してアクセスがあるまでは実ページの確保は行わない．このため、アプリケーションが確保要求したメモリ領域へ実際にアクセスするとページフォルト割り込みを発生させ実ページの確保を行う．メモリの開放の際は、メモリ領域を表す'vm_area_struct'構造体を'mm_struct'構造体のリンクから削除し、そのアドレスに対応するPTEをクリアする．そして、割り当てられていたページが他のプロセスから参照されていなければページを未割り当てリストに追加する．図6にメモリ割り当ての流れを示す．

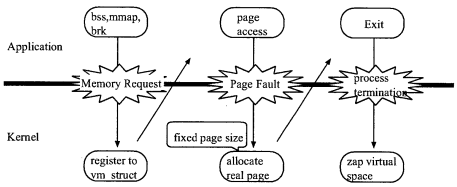



Fig. 6　メモリ割り当ての流れ

## 4 Super Page カーネル

### 4.1 設計方針

シンプルで実用的なSuper Page Kernelを目指し、参考文献[1) 2) 3)]に示される基本方針を基礎とした．また今回の設計においては、Super Pageの利用効率を向上させるため新たに以下の2つの設計方針を基本方針に加える．

- ZAP時のSuper Pageブロックの返却
  従来のSuper Pageカーネルではラージページを開放する際、一度ページサイズをベースページにまでダウングレードさせ、その後バディシステムにベースページ単位での返却を行っていた．この返却方法では、返却途中に他のプロセスにラージページ領域の一部を割り当てられる可能性が生じる．もし返却途中のラージページの一部が他のプロセスにより利用された場合、現在返却中のラージページを次回のラージページ割り当て時に利用することができなくなり、バディシステムの保有するラージページ数が減少してしまう．実際に2.5系Super Pageカーネルを用いてラージページの割り当てと開放を繰り返し、バディシステムが保有するラージページの保有数を調べた(図7)．

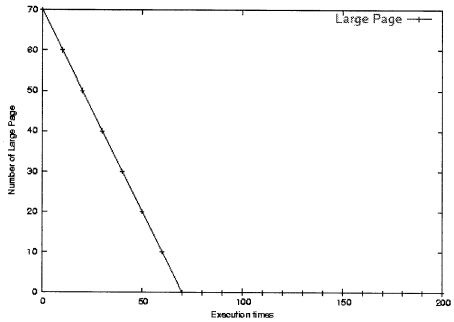



Fig. 7　ラージページ数の推移

この結果から、割り当てと開放の繰り返し回数の増加がラージページ保有数の減少に繋がることが明らかとなった．このため、ラージページブロックをまとめてバディシステムに返却することで、上記の問題を解決するとともに返却時間の短縮を図る．

- Super Page境界へのメモリ割り当て
  Super Pageを利用するためにはバディシステムとの関係上、要求メモリがSuper Page境界にある必要がある．Super Page境界にない場合は端数が増加し、ベースページでのメモリ利用が増加する．ベースページの利用はTLBミスの増加を招くため、端数の増加は性能低下を招くと予想できる．このため、Super Page境界からSuper Pageサイズ境界長で可能な限りメモリ割り当てを行い、端数の増加を抑える．

### 4.2 実装

実装はIA32アーキテクチャをターゲットとして行い、実装方法は参考文献[1) 2) 3)]を参考にしている．本Super Pageカーネルの実装方法を説明する．Linuxはメモリ管理においてソフトウェアからPTへの参照を必要とする場合があるため、4MB利用時においても4KBページでのPTを参照可能にしなければならない．このため、我々はソフトウェア用とTLB用の2つのPGDを確保し、この二つのPGDを使い分けることでSuper Pageカーネルを実現している．Super Pageカーネルのメモリ割り当ての流れを図8に示す．ユーザプログラムからメモリ要求された場合、仮想アドレス範囲内の4MB境界で、4MBの割り当てが可能である場合、その4MB領域の1024個のPTEにSuper Page予約フラグを設定する．具体的にはPTEのシステムソフトウェア用の空きビットにフラグを立てる(図9)．これを我々はSuper Page予約と呼ぶ．ただしここ

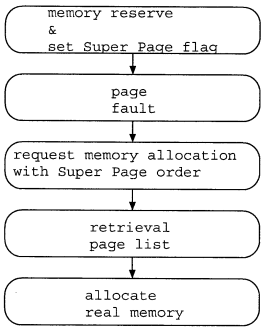



Fig. 8　Super Page カーネルのメモリ割り当ての流れ

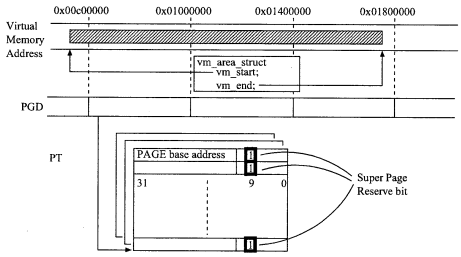



Fig. 9　Super Page 予約

では実メモリの割り当ては行わない.
実メモリの割り当てはオリジナルカーネルの方針に従い、ページフォルト割り込み時に行う. 割り当て時に Super Page 予約が設定されている場合、バディシステムに対して 4MB の連続したメモリを要求する. 4MB メモリの確保に成功した場合、TLB 用の PGD に 4MB ページの割り当てを行う. また、ソフトウェア用の PGD には 4MB ページを 1024 に分け、ページエントリに 4KB ページを割り当てる. 4MB の割り当てに失敗した場合は 4KB のベースページでの割り当てを行う. 図 10 に 4MB ページにおけるアドレス変換、図 11 に 4KB ページにおけるアドレス変換を示す. 4KB ページではソフトウェア用 PGD と TLB 用 PGD は同じ PT への参照値を持ち、4MB ページでは TLB 用 PGD は物理メモリを直接参照可能となる.

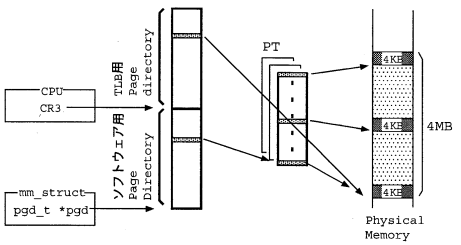



Fig. 10　4MB ページでのアドレス変換

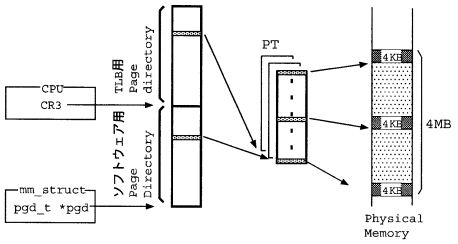



Fig. 11　4KB ページでのアドレス変換

4MB ページを割れ当てたページではスワップの発生による一部のページの開放やソフトウェアからの 4KB ページ単位での作業が行われる場合があるため、Super page カーネルにはダウングレードを実装している. ダウングレードは 4MB ページ内の PTE の Super Page 予約フラグをクリアし、ソフトウェア用の PGD を TLB 用の PGD にコピーすることにより行う. また、この際 TLB との整合性をとるため一度 TLB をフラッシュする.

## 5　評価

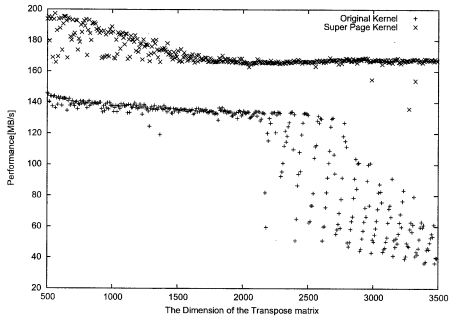
### 5.1　オリジナル対 Super Page

オリジナルカーネルと Super Page カーネルにおいて図 12 に示す行列転置ベンチマークプログラムを実行し、メモリ転送性能の比較を行った. Super Page カーネルではページアラインを動的に Super Page 境界に合わせるよう実装を行うが、今回の評価時までにその実装が間に合わなかったため、mmap 関数でのメモリ確保時にユーザが Super Page 境界にアラインを合わせるよう調整を行っている. 表 2 は評価を行ったマシン環境であり、以下に評価条件と内容を示す.

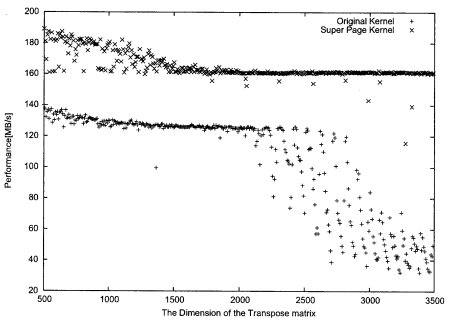
Table 2　評価環境

| CPU | AMD Athlon(tm) |
|---|---|
| CPU Frequency[MHz] | 1094.152 |
| Cache Size[KB] | 256 |
| Memory [MB] | 512 |
| Linux Kernel | 2.6.23 |

- Dimension*Dimension の正方配列 b から同サイズの正方配列 a への転置複製を行う.
- mmap 関数によりメモリを確保する.
- 複製元の配列を縦方向に読み出す方式 (Load Stride) と複製先の配列に縦方向に書き込む方式 (Store Stride) を行う.
- Dimension の値を 500 から 3500 の間で設定する.
- 変更を加えていないオリジナルカーネルと今回実装した Super Page カーネルでプログラムを実行する.

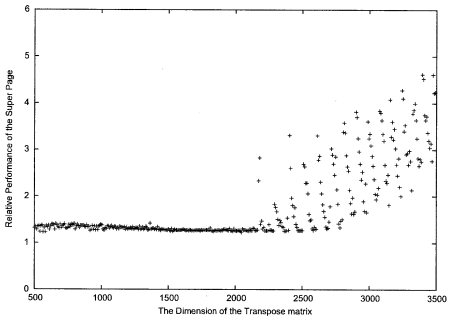



(a)Load Stride のメモリ転送性能

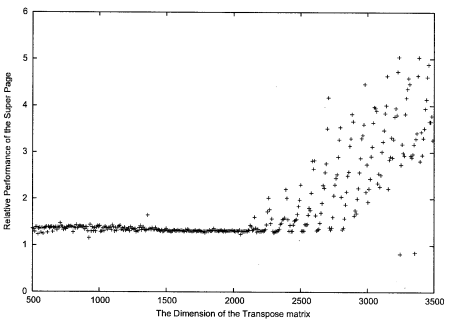



(b)Store Stride のメモリ転送性能

Fig. 13　Load Stride 性能結果

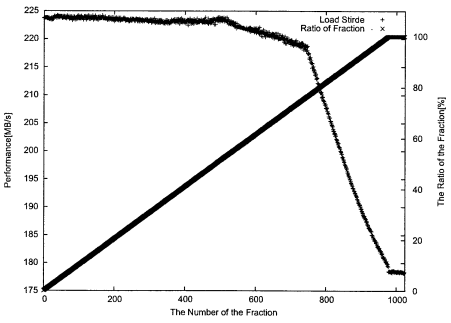



(a)Load Stride のオリジナル対 Super Page 性能比

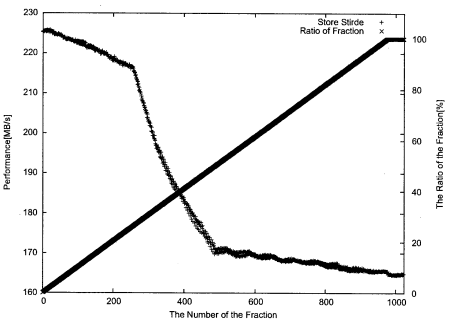



(b)Store Stride のオリジナル対 Super Page 性能比

Fig. 14　オリジナル対 Super Page 性能比




(a)Load Stride




(b)Store Stride

Fig. 15　500 × 500 Matrix における端数増大における性能推移

```
for(k=0; k<ITR; k++) {
    for(i=0; i<dim; i++) {
        for(j=0; j<dim; j++) {
            a[i][j] = b[j][i];
        }
    }
}
```

Fig. 12　行列転置ベンチマークプログラム

図13にオリジナルカーネルとSuper Pageカーネルのメモリ転送性能を示す．縦軸は転送性能[MB/s]、横軸はdouble型の正方配列の一辺の長さである．図14はSuper Pageカーネルの転送性能をオリジナルカーネルの転送性能で割り求めた性能比である．オリジナルカーネルでは4KBページサイズを用いるため、利用メモリが増加はTLBミスの増加を招き性能を低下させていることが分かる．

### 5.2　端数増加による性能影響

従来のSuper Pageでは、Super Page境界に動的にアラインを合わせる実装を行ってきた．しかし、ページアラインを合わせることによる効果についての評価を行っていない．このため、Super Pageカーネルにおいて4MBページ境界から4KB単位で0から1023までの端数を発生させ、端数増加による性能への影響を調べ、ページアラインの必要性について考察する．図16に本評価の概念図を示す．4MBページ境界からのメモリに対しては4MBページによるアクセスが可能であり、4MBページ境界からあふれた端数に対しては4KBページでのアクセスとなる．つまり端数の増加は4KBページの増加を意味する．図15に評価結果を示す．左の縦軸は転送性能[MB/s]、右の縦軸は端数領域に含まれる配列の割合[%]、横軸は端数の数である．

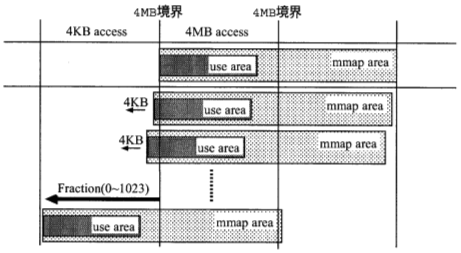



Fig. 16　端数増大における性能評価

Store Strideの場合では、端数の増加が性能低下に直接結び付いており、端数の割合が利用領域の約50[%]に達すると性能は約50[%]低下している．これはStrideアクセスする領域から端数となり、TLBミスを招きやすいことが原因であると考える．また、Load Strideの場合では、端数の割合が80[%]付近で性能が著しく低下する．これはStrideアクセスする領域が利用メモリの後半部分にあり、端数の割合が50以上[%]になるまでSuper Page領域にStrideアクセスする領域が存在すること、さらに端数割合が50[%]以下のときは端数領域のアクセスは連続したアクセスとなるため、Store Strideアクセスに比べてTLBミスが発生しにくい状況であることが原因であると考える．

Store StrideとLoad Stride実行結果より、端数の増大が性能低下を招くことは明らかである．このため、Super Pageカーネルに対して動的にページアラインを合わせる機能を実装することは必要であると言える．

## 6　まとめ

本稿では、2.6系Linuxカーネルに実装したSuper Pageの設計方針や実装方法について述べた．そして、実装後にオリジナルカーネルとの性能比較を行い、Super Pageの有効性を示した．また、端数増加による性能への影響を調べ、端数の増加が性能の低下を招くことを明らかにした．これにより、これまでのLinux Super Pageプロジェクトで実装してきた、動的にページアラインを合わせる機能の必要性を証明することができた．

今後、本Super Pageカーネルにページアライン機能とバディシステムへのSuper Pageブロック単位でのメモリ返却の実装を行う．

## 参考文献


1) 高取 研, "IA32版Linux Super Pageの開発", 2001F年度 東海大学 工学部 通信工学科 学部論文, 2001

2) Naohiko Shimizu, Ken Takatori, "A Linux Super Page Kernel for Alpha, Sparc64 and IA32 -Reducing TLB Misses of Applications", MEDIA 2002 workshop, 2002

3) 早坂 晴康, "シャドーページーディレクトリを用いたIA-32版Linux Super Pageの実現とSuper Page端数増大によって起こるメモリ転送性能低下に対するPage Coloringによる改善", 2003年度 東海大学大学院 工学研究科 電気工学専攻 修士論文, 2003

4) 高橋 浩和, 小田 逸郎, 山幡 為佐久, "Linuxカーネル2.6解読室", ソフトバンク クリエイティブ株式会社, 2007

5) Daniel P. Bovet, Marco Cesati, "詳解 Linuxカーネル 第3版", 株式会社オライリー・ジャパン, 2007

6) Linux Kernel Hack Japan, http://hira.main.jp/wiki (2007年12月12日)

7) 清水研究室 Linux Super Page -Transparent super page support for Linux, http://shimizu-lab.dt.u-tokai.ac.jp/lsp.html (2007年12月12日)