3L－7

OSの仮想記憶インターフェースの自然言語処理への応用

國吉 芳夫　中西 正和
慶應義塾大学 理工学研究科 計算機科学専攻

1. はじめに

近年、テキストコーパスに基づく自然言語処理の研究が盛んだが、これには大量のデータを処理する必要がある。大量のデータはファイル操作で処理することが一般的であったが、効率を上げるには高度の技術を要する。一方最近では商用OSでも、仮想記憶の機能を利用するためのシステムコールを用意していることが普通である。本稿では、OSの仮想記憶機能を利用して任意のファイルを簡単に仮想記憶として使用できるようにするためのライブラリの仕様を提案する。

2. 仮想記憶機能のためのシステムコール

仮想記憶機能を利用するための中心的な役割をするシステムコールで標準的なものはmmapである[1]。このシステムコールは、指定されたmemory objectが適当なアドレス空間に写像されるようにする。パラメターの指定により、共有メモリを実現したり、読み書きや実行許可のモードを設定する事ができる。

3. 仮想記憶ライブラリ関数

3.1 仮想記憶ライブラリ関数が満たすべき条件

mmapを利用すれば大規模のデータに対しても簡潔で効率の良いプログラミングが可能になる。しかし、システムコールは基本的な機能しか提供しないしシステムによって仕様が微妙に異なるため、高位のライブラリ関数が提供される事が望ましい。本稿で提案するライブラリでは以下のような要求を満たすものとした。

1. ファイル名を渡すとマップされるメモリ空間へのポインタを返すというような直感的で分かりやすいインターフェースを持つこと。
2. マップするファイルのサイズを越えて書き込んだ時、自動的にサイズを拡張すること。
3. 異なるシステムで互換性を保てるようにする。

3.2 関数の仕様

本稿で提案するライブラリはmapfile, 及び関連する関数群からなる。

3.2.1 mapfile

```
void *mapfile(char *name, char *mode[, size_t len])
```

関数mapfileはnameで指定したファイルをmodeで指定されるモードでマップしたアドレス空間の先頭番地を返す。自動拡張時の最大サイズをlenとする。モード指定には次の表で示す記号が使用できる。

モード	意味
r	読み込みのみ許可
w	読み書き許可
c	読み書き許可, copy on write
w+	読み書き許可, 自動拡張
c+	読み書き許可, copy on write, 自動拡張
x	実行許可 (r,w,c,w+,c+に付加して使用できる)

自動拡張する際の最大サイズを指定するのは、複数のmemory objectを使用する際にプロセスのアドレス空間が小さいサイズに分断されて拡張ができなくなることを防ぐためである。

3.2.2 他の関数の仕様

unmapfile: マップされているアドレス空間の開放。
mapbase: 領域内を指すポインタを渡すとマップされているアドレス空間の先頭を指すポインタを返す。
mapsize: オブジェクトのサイズを返す。
mapmaxsize: マップされているアドレス空間の最大のサイズ (mapfileで指定したサイズ) を返す。
maptouch: 指定した長さのページを主記憶に読み込むことをOSに対して指示する。後述のアルゴリズムプリフェッチのために使用できる。
setmapsize: マップしているファイルのサイズを指定した大きさに調整する。

Yoshio KUNIYOSHI　(yoshio@nak.math.keio.ac.jp)
Masakazu NAKANISHI
Department of Computer Science, Graduate School of Science and Technology, Keio University 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Kanagawa 223, Japan

4. 写像空間の自動拡張

4.1 ファイルのサイズを越える領域に対するアクセス

mmapで割り当てたファイルのサイズを越えた領域にアクセスすると、所定のシグナル (SIGBUSなど) を引き起こす。そこで、このシグナルを捕獲し、ページフォールトを生じたアドレスが含まれるようにファイルのサイズを大きくすることで空間の自動拡張をする。

4.2 OSの差異の吸収

システムコールの仕様はOSによって微妙に異なることが多い。主な制限としては、フォールトアドレスが取得できない (BSD/OS 2.0など) ことや、共有ページが使えない (Linux 1.2など) ことなどがある。ライブラリは、これらの制限があっても論理的には互換性が保てるように実装する。

5. メモリのアクセスパターンによる処理効率の向上

最近ではCPUと主記憶との速度差が激しいので、キャッシュのヒット率を向上させるようなアクセスパターンを採用することが効率上重要である[2]。同様のことが、ファイルアクセスと主記憶のアクセスとの速度の落差に関してもいえるので多重ループ処理のブロック化、アルゴリズムプリフェッチなどといった技法を使用することでディスクアクセスを減らす工夫をすることが重要である。本ライブラリでは、アルゴリズムプリフェッチのためにmaptouchという関数を用意している。

6. ソートによる実験

UNIXに標準装備のsortコマンドはファイル入出力によるソートのための高度の最適化がなされている複雑なプログラムである。配列を使用したソーティングによる単純なプログラムの実行時間と比較することで、本稿で提案するライブラリを使用することの有効性を示す。

約62万行、サイズ約230MBのEDR共起辞書を対象として、mmapを利用したソートプログラムmmsort (mergesortを使用) とSunOS4.1のsortコマンドの実行時間を測定した。時間は2回の平均値である。使用可能な主記憶の量は約80MBであり、他に走行中のプロセスはなかった。

手法	real	user	system
mmap (mergesort)	209.7	86.1	43.7
sort (SunOS4.1)	526.6	247.6	83.8

いたって単純なプログラムであるmmsortでも効率良く実装できていることが分かる。ただし、mmsortで使用したmergesortはアクセスパターンの局所性が比較的良いことに注意しなければならない。局所性の悪いradixsortを使用した実験では実時間で10倍近い処理時間を要した。

7. 結論

最近のOSに標準の仮想記憶インターフェースを使用したライブラリ関数の仕様を提案した。これを使用すれば、極めて単純なプログラムでも大規模なデータを効率良く扱うことができ、互換性も維持できる。しかし、次のような問題点がある。

- mmapシステムコールの実装がいい加減なシステムでは著しく性能を落す可能性がある。
- 主記憶のサイズを越えるデータに対してランダムアクセスをすると小さいブロックサイズのディスクアクセスの頻発を招くことも重なって極端に遅くなる。そのようなことが起きないように使用するアルゴリズムに注意する必要がある。
- 現在のOSはまだ32bitが主流であり、アドレス空間は4GBを越えられない。この値は、実験レベルであっても必ずしも十分な値ではない。もっとも、今後普及するであろう64bitのOSでは、この問題は解決するはずである。

参考文献

[1] Gingell, R. A., Moran J. P., Shanon W. A. 1987. Virtual Memory Architecture in SunOS. In *Summer Conference Proceedings, Phoenix 1987.* USENIX Associtation.

[2] 寒川 光. 1995. RISC超高速化プログラミング技法. 共立出版.