平成16年度オペレーティングシステム試験

2005年2月14日

問題は3問，5ページある．

1

Unixや Windowsなどの現代的なオペレーティングシステムには，ファイルマッピング (メモリマップドファイル) という機能が提供されている．参考までに，Unixでそれを行うシステムコールである mmapのマニュアルページの抜粋を次ページに掲載している．以下の問いに答えよ．

(1) このシステムコールを用いて，あるファイルの中身を読み込むプログラム片の概要を記せ．具体的にするために，`"my_file"`というファイル中に含まれる改行記号 (`'\n'`) の個数を数えるプログラムを例にとって，擬似コードの形で書け．ただし，ファイルの大きさ (N バイトとする) はあらかじめ与えられているとする．システムコールの詳細な知識を要求しているのではないので，引数の順序や詳細などを忘れた場合は，自然言語で意図が分かるように書けばよい．また，エラー検査なども説明を簡単にするために省略してよい．

(2) 一般論として，mmapを用いてファイルの中身を読むのと，通常の open/readや fopen/freadを用いて読むのとで，性能上どのような利害得失があるか．以下のようなパラメータの違いにより，どのような場合にどちらが優れているかを，オペレーティングシステムの動作を踏まえた上で，理由とともに述べよ．

- ファイルの大きさ
- 搭載物理メモリ量
- 論理アドレス空間の大きさ
- ファイル全体の中でアクセスされる領域．その大きさやアクセスパターン
- そのファイルを同時期に読み込むプロセスの数

参考資料: mmap マニュアルページ

```
MMAP(2)             Linux Programmer's Manual             MMAP(2)

NAME
       mmap, munmap - map or unmap files or devices into memory

SYNOPSIS
       #include <unistd.h>
       #include <sys/mman.h>

       void  *  mmap(void  *start,  size_t length, int prot , int flags, int fd, off_t offset);

DESCRIPTION

       The mmap function asks to map length bytes starting at offset offset from the file (or
        other object) specified by the file descriptor fd into memory, preferably at address
        start.  This latter address is a hint   only, and is usually specified as 0.  The
        actual place where the object is mapped is returned by mmap.  The prot argument
        describes the desired memory protection (and must not conflict with the open mode
        of the file). It has bits

       PROT_EXEC  Pages may be executed.

       PROT_READ  Pages may be read.

       PROT_WRITE Pages may be written.

       PROT_NONE  Pages may not be accessed.

       The  flags  parameter  specifies  the  type  of the mapped object, mapping options and
        whether modifications made  to the  mapped copy of the page are private to the process
        or are to be shared with other references.  It has bits

       MAP_FIXED  Do not select a different address than the  one specified.   If the specified
                address cannot be used, mmap will fail.  If MAP_FIXED  is  specified, start
                must be a multiple of the pagesize. Use of this option is discouraged.

       MAP_SHARED Share this mapping  with  all  other  processes that map this object.
                Storing to the region is equivalent to writing to the  file.   The  file
                  may  not  actually be updated until msync(2) or munmap(2) are called.

       MAP_PRIVATE
                  Create a private copy-on-write mapping.  Stores to  the region do not
                affect the original file.

       You must specify exactly one of  MAP_SHARED  and  MAP_PRIVATE.

       offset should ordinarily be a multiple of  the  page  size returned by getpagesize(2).
```

2

次の文章を読んで後の問いに答えよ．

男: 「くそっ，またセグメンだよ」

バンッ（キーボードをたたく音）

男: 「あ，壊れちった… 学校から借りてるコンピュータなのに… こら先生に怒られるわ…」

女: 「ご機嫌ななめのようね」

男: 「だってよう，このプログラムがさっぱり訳が分からないんだよ．ちゃんと動くときもあれば，間違った答えが出るときもあれば，セグメンで落ちることもあるんだよ．提出の締め切りはもう 2 週間後だっつうのに (注: B 演習の締め切りは 3/1 とします)．おまけに今日はバレンタインデーだっつうのに」

女: 「それは○○君のプログラムがおかしなメモリアクセスをしているだけの話じゃなくて?」

男: 「いやだってさ，意外と落ちないときもあるんだよ」

女: 「ぷーっ，○○君は意外と分かってないんだね，コンピュータのこと．電気なのに」

男: 「カッチーン… あ，でもこないだなんか，最適化オプションをつけたとたんに動かなくなったよ．これってコンパイラのバグだろ．他にも，デバッガで見てみたら，malloc の中でセグメン起こしてたり．これは malloc のバグだろ」

女: 「ふう．残念ながら○○君の私の中の評価は下がる一方だわ．そもそも C のプログラムでセグメンテーションフォルトって言うのはどういうときにおきるんだったかしら，言ってごらん」

男: 「(な，なんだよこの女王様みたいな口の利き方は．ま，これでバグが取れるなら，この場は我慢して従っとくか) ええっと，配列の大きさを超えてアクセスしたときとか… です」

女: 「あー，その理解が甘いってゆってるの．もちろんそれでセグメンテーションフォルトがおきるときもあるけど，常にそうなるとは限らないのよ」

男: 「… ええっと，あ，そうか，思い出してきたぞ．セグメンテーションフォルトは,『不正なメモリアクセスをしたときにおきる』でどう？」

女: 「うん，まあ間違ってはいないけどほとんど意味のない文よね．そもそも『不正なメモリアクセス』っていうのがどう決まるのかを理解しないと．じゃ，具体的に聞くけど，以下の (A)(B)(C) は全部，配列の大きさを 129 バイトだけ超えてアクセスしてるんだけど，この中で一個だけ，まず セグメンテーションフォルトに ならない ものがあるの．○○君にはどれだかわかるかな? N はせいぜい数万くらいまで，と思って」

```
char a[N];
void work()
{
  char b[N];
  char * c = (char *)malloc(N);

  a[N + 128] = 0;                  /* (A) */
  b[N + 128] = 0;                  /* (B) */
  c[N + 128] = 0;                  /* (C) */
}
int main()
{
  char h[10000];
  work();
  exit(0);
}
```

男: 「ええとつまり，a は大域配列，b は局所配列，c は動的に割り当てられた配列ってわけだな．うーーーんっと，[　　]です.」

女: 「.... ファイナルアンサー?」

男: 「はしゃぎすぎだよ．わかったから先行けよ」

女: 「ダーーーン… 残念！答えは (1) です.」

男: 「ふーん，どうして?」

女: 「 (2) . 故に，このアクセスは，確かに配列のサイズを超えていても，オペレーティングシステムから見たら『不正なアクセス』には分類されないのです」

男: 「こ，これからはセンセイと呼ばせていただきます．センセイ，質問があります．Java の場合，配列のサイズを超えたアクセスは必ず，ArrayIndexOutOfBoundsException として捕まえてくれるよね.」

女: 「そうね．Java に限らずね．まあ言語の数から言ったら圧倒的にそっちの方が常識なんだけど」

男: 「僕は，C でも同じようにしていてほしかったです!」

女: 「うーん，良い指摘だけど，Java の場合，基本的には配列をアクセスするごとに，添え字がサイズ未満かどうか，ソフトウェアでチェックをしているの．それは C の配列アクセスと比べるとずいぶん遅いときもあるんだな」

男: 「でも，セグメンテーションフォルトがおきるということは，なんらかのチェックはソフトウェアによって入っているんでしょう? つまり，そのアクセスが『不正でないかどうか』の」

女: 「あらあらまた大きな勘違い．セグメンテーションフォルトがおきるかどうかは，ソフトウェアがチェックしているのではないのよ．そのために CPU に備わっている仕組みを何と言うんだったかしら?」

男: 「あ， (3) です… そうか，ずいぶん複雑なことを CPU がやってくれているんだね」

女: 「よくできました．やればできるじゃないの．ところで，さっきの 3 つのアクセスの中で， (1) 以外の残りの二つについては，セグメンテーションフォルトがおきるかどうかは OS，言語処理系，ライブラリの実装次第と言うところがあって，なかなか予測するのは難しいのだけど，できればセグメンテーションフォルトを出してあげたほうが親切よね？」

男: 「ええーーっ，セグメンテーションフォルトなんて見たくもないんだけど」

女: 「あのねぇ… やればできるってほめて上げたばかりなのに… 落ちようが落ちまいがバグはバグでしょう．セグメンで落ちてくれるのと，しばらく走った後，関係ないところで不可解な挙動を示すのとどちらがいいの?」

男: 「… お，落ちてくれたほうがいいです」

女: 「でしょう．あなたの言うコンパイラの最適化オプションを変えると出るだとか，malloc の途中で落ちるとか言うのも，結局そういう，本当はいけないアクセスが，セグメンできちんとつかまらずに突っ走ってしまっているだけの話なのよ」

男: 「わかりました…」

女: 「じゃあ質問に戻るけど，残りの二つについて，できれば確実にセグメンテーションフォルトがおきるようにしてあげるには，どういう風にしてあげたらよいと思う? もちろん，C コンパイラを改造してソフトウェアで毎回チェックを入れるというのは原理的には可能だけれど，今はそういうことではなく，もっと単純に」

男: 「はいセンセイ分かりません．お願いします」

女: 「ったく．たとえば (4) 」

男: 「そうかぁ，なんか，今日だけでコンピュータのメモリ管理についてずいぶん詳しくなったような気もするな… あ，でも，結局バグがどこにあるかは全然わかってないんだけど… センセイ，お願いします」

女: 「あ，それは自分で突き止めなきゃ，演習なんだから．それはそうと気分転換に工場見学にでも行って来たらどう? まだ事務に行けば間に合うみたいよ」

問:

- (1) に当てはまるのは (A), (B), (C) のいずれであるかを書け.
- (2) には，(1) に入った答えを選んだ根拠が入る．適切な文章を書け.
- (3) に適切な言葉を入れよ．また，この会話で話題になっていること以外に，(3) が持つ機能，それが OS の中で果たしている役割を述べよ.
- (4) には，(1) で選ばなかったアクセスに対して，言語処理系，ライブラリ，OS の動作によって，セグメンテーションフォルトがおきるようにする一方法が入る．適当な文章を入れよ.

3

A だけの時間計算をしては，B だけの時間休眠することを繰り返す以下のようなプロセスを考える．

```
int main(int argc, char ** argv)
{
  while (1) {
    double t0 = current_time();
    while (current_time() - t0 < A) {
      do_comp();
    }
    if (B > 0) sleep_a_little(B);
  }
}
```

ここで，

- `do_comp` は非常に短い時間 (せいぜい 1ms 以下)，決してブロックしない計算をする関数であるとする．
- `current_time()` はかなり正確に (高々数 μ 秒の誤差で) 現在時刻を返す関数であるとする．
- `sleep_a_little(`B`)` は与えられた引数の時間，そのプロセスを休眠させる．ただし，実際に休眠から返るのは，経過時間の後，そのプロセスが実際に OS によってスケジューリングされる時であることに注意．

以下のような (Linux 2.4 風の) スケジューラを仮定する．CPU は一つであるとする．

- タイマ割り込み間隔 =10ms
- 各プロセスに，スケジューリングに用いられる「適合値」と呼ばれる整数が維持される．プロセスが生成された直後は 10 が割り当てられる．
- タイマ割り込み時には以下を行う．
  - その時走っていたプロセスの適合値を 1 減らす．その結果としてそのプロセスの適合値が 0 になれば，(以下に述べる) 再スケジュールを行う．
  - また，`sleep_a_little` で指定された休眠時間が経過していないかどうかが検査され，そうならばそのプロセスを実行可能にした上で，やはり再スケジュールを行う．
- 再スケジュールは，すべての実行可能プロセスの中で適合値最大のものを次に実行する．
- ただしこのとき，すべての実行可能プロセスの適合値が 0 であった場合，すべてのプロセスの適合値を次にしたがって再設定した上で再スケジュールを行う．

$$\text{新適合値} = \text{旧適合値} / 2 + 10$$

以下の問いに答えよ．

(1) $B = 0$ (つまり決して休眠しない) プロセスを二つ走らせた場合，何 ms ごとにプロセスが切り替わるか

(2) $A = 50$ms, $B = 50$ms としたプロセスをひとつだけ走らせた場合，当然のことながらこのプロセスは，ほぼ 50% の CPU 時間を得ると予想される．しかし実際に測定をしてみるとそれより少し小さな値 (45% 程度) が観測された．なぜか? (他のプロセスによる影響はないものとする．つまり，このコンピュータにはこのプロセスしか走っていない)

(3) (2) で述べたプロセス P と，$B = 0$ としたプロセス Q をひとつずつ走らせた場合，P は長期間の平均としてだいたいどの程度の CPU 時間を得ると予想されるか．(a) 33% よりかなり少ない, (b) 33%程度 (c) ほぼ 45%程度 ((2) と同様), (d) 45% よりずっと多い，の中から選び，理由を述べよ．(a), (d) を選択した場合，それがだいたいどの程度になるかもあわせて述べよ．

(4) $A = 5000$ms, $B = 5000$ms としたプロセス P' と Q を併走させた場合についてはどうか? 理由とともに応えよ．

問題は以上である