

平成24年度オペレーティングシステム期末試験

2013年2月5日

注意事項

- 問題は3問, 8ページある.
- 1枚の解答用紙に1問解答する(複数の問題の解答を1枚に混ぜたり, 1問の解答を複数の用紙にまたがって書いたりしない) こと
- 各解答用紙にはっきりと, どの問題に解答したのかを明記すること
- 問題文をよく読み, 解答として要求されている内容, 答え方の形式に従って答えること
- 提出時は, 3枚の答案を問題1, 2, 3の順に重ねてホチキスでとめること

1

スレッドに公平に、かつ十分な頻度で CPU 資源を割り当てるために、以下のようなスケジューラを考えた。簡単のため、CPU はひとつしかないものとする。

- 各スレッドに、変数 v, l を管理する。
- スレッドが新しく生成された際、そのスレッド (t とする) は、それを生成したスレッド (p とする) の v, l を引き継ぐ。つまり、 t の v, l は p のそれと同じ値に初期化される。
- 1 ms ごとにタイマ割り込みをかける。
- (タイマ割り込みを含む) 割り込み発生時には以下の処理をする。

```
1  t = その時までCPUを使っていたスレッド;  
2  c = 現在時刻;  
3  t->v += c - t->l;  
4  m = 実行可能なスレッドの中で最小のvを持つスレッド;  
5  if (t->v > m->v + 10ms) {  
6      m->l = c;  
7      m に CPU を割り当てる;  
8  } else {  
9      t->l = c;  
10     t に CPU を割り当てる;  
11 }
```

- 現在 CPU が割り当てられていたスレッドの実行が 中断 した際には、上記と似た以下の処理をする。

```
1  t = その時までCPUを使っていたスレッド;  
2  c = 現在時刻;  
3  t->v += c - t->l;  
4  if (実行可能なスレッドが存在する) {  
5      m = 実行可能なスレッドの中で最小のvを持つスレッド;  
6      m->l = c;  
7      m に CPU を割り当てる;  
8  } else {  
9      次の割り込みまでCPUを停止させる  
10 }
```

このスケジューラについて以下の問いに答えよ。

- (1) 下線部、スレッドの 中断 とは、何を契機としておきる事象か？ 知る限りの例を 5 つまで述べよ。
- (2) 常に実行可能なスレッドが N 個、長時間実行しており、それ以外に実行可能なスレッドは存在しない状況を考える。各スレッドは、どのくらいの間隔で、1 回にどのくらいの時間、CPU の割り当てを受けることになるか？ 「 X ms ごとに 1 回、 Y ms の割り当てを受ける」という形式で答えよ。スケジューラの挙動に基づいて、その理由も説明せよ。
- (3) 上記の N 個のスレッドに加え、以下のように、1 文字入力を受け取っては、5ms 程度 CPU を消費する処理 `process_char` を繰り返すスレッド I が 1 つ存在する状況を考える。

```
1  while (1) {  
2      c = getchar();    // 1文字入力  
3      process_char(c);  // 5ms CPU を消費  
4  }
```

文字入力 は 100ms に一文字程度の頻度で発生するとする。実際に文字入力が発生してから、スレッド I が `process_char` を開始するまでの時間は最大でどのくらいか？ また、`process_char` を完了するまでの時間は最大でどのくらいか？ 理由も含めて述べよ。 N によって異なる場合、適切に場合分けなどをして答えよ。

- (4) 実は、上記のスケジューラには、スレッドの挙動によっては、一スレッドが長時間 CPU を専有できてしまうという致命的な欠陥がある。どのようなスレッドの挙動によってその欠陥が現れるかを、スケジューラの挙動に基づいて具体的に示しながら、説明せよ。
- (5) 上記の欠陥の解決法について述べよ。

解答

- (1)
- I/O の完了待ち: read, recv, write, send, select など
 - 同期による待ち: pthread_mutex_lock, pthread_cond_wait, pthread_join, wait など
 - 自主的休眠: sleep, usleep など
 - ページフォルト

- (2) $10N$ ms に一度, 10ms の割り当てを受ける.

3 行目により, v には各スレッドが実行された時間が記録されている. そして 5 行目によりそれが他のスレッドよりも 10ms 以上上回ると, 他のスレッドに切り替わることになる. したがって十分長時間後には各スレッドが 10ms ずつ実行しては他のスレッドに CPU を譲るという動作が繰り返される.

- (3) $N \leq 19$ の場合最大で 0ms. 多い場合, $10N$ ms.

このスレッドを I (Interactive) と呼ぶ. I は最大でも 100ms の間に 5ms 程度 (5%) の CPU しか消費しない. そのため長時間 (T) たった後の v は高々,

$$T/20$$

となる. 残る N 個のスレッドが残り時間を分け合うので, 各スレッドには最低でも,

$$19T/20N$$

ずつが割り当てられる. $N \leq 19$ であれば前者の方が小さい. つまり, この I の v の値は, 残りの N 個の v よりも小さいという状態が常に維持される. したがって割り込みが発生すると直ちに I が実行される (注 1). $N \geq 20$ ならば I の v が他と比べて小さくなる理由はなく, $(N + 1)$ 個のスレッドに $10N$ ms の遅延が発生しうる (注 2).

- 注 1: 1 文字入力 that 割り込みによって発生するのだとすると遅延は (ほぼ) 0 ms となるが, 「次のタイマ割り込み時に実行されるので 1 ms」というのも正解.
- 注 2:

- (4) あるスレッド A が中断することなく実行している状態で, 別のスレッド B が長時間 (例えば 10 秒) 中断していた後に再開したとする. 結果, A の v にと B の v に 10 秒の差が出来, B が再開すると 10 秒間 (A は相変わらず実行可能であるにも関わらず) CPU を占有できてしまう.
- (5) あるスレッドが中断から再開した際, そのスレッドの v を, 最低でも他のスレッドの v - 一定時間 (たとえば 20ms) とする. 結果としてどれだけ中断していても, CPU を占有できるのは 20ms までとなる.

2

以下はスギちゃんという学生がひとりごとをつぶやきながらオペレーティングシステムの勉強をしているときの記録である。よく読んで後の問いに答えよ。

スギちゃん: OS の試験が明日なんだぜえ。

でもまだまったく勉強してないぜえ。ワイルドだろお。

スギちゃん: ファイルの読み方について勉強するぜえ。練習問題を解くぜえ。

問題: 「 x を 0 以上 $2^{64} - 1$ 以下の整数, F を, 0 以上 $2^{64} - 1$ 以下の整数が多数, 昇順に格納されたファイルとする。

```
1 find_num F x
```

として起動すると, F 中に x が現れるか否かを答えるプログラム `find_num` を書け。ただし, 必要ならば二分探索を行う C ライブラリ関数 `bsearch` を用いても良い (`bsearch` の使い方は付録のマニュアルページ参照)。

スギちゃん: そんなの簡単だぜえ。

まずファイルを開くには `open` システムコールを使うぜえ。

その後ファイルの中身を `read` システムコールを使って読み込んで, `bsearch` を呼べばおしまいだぜえ。

こんな感じだぜえ。ワイルドだろお。

```
1 int main(int argc, char ** argv) {
2     char * F = argv[1];          /* 引数 1: ファイル名 */
3     long x = atol(argv[2]);       /* 引数 2: 見つけたい x */
4     long sz = file_size(F);       /* F のサイズ(バイト数) */
5     long N = sz / sizeof(long);   /* F の要素数 */
6     int fd = open(F, O_RDONLY);
7     long * a = malloc(sz);
8     read_bytes(fd, a, sz);
9     void * found = bsearch(&x, a, N, sizeof(long), cmp_long);
10    close(fd);
11    if (found) {
12        printf("%ld は見つかったぜえ\n", x);
13    } else {
14        printf("%ld はなかったぜえ\n", x);
15    }
16    return 0;
17 }
```

注: ただし,

- 簡単のためエラー検査などは省略している
- 4 行目の `file_size(F)` は, F のサイズをバイト単位で返す。
- 8 行目の `read_bytes(fd, a, bytes)` は, 指定されたバイト数 (`bytes`) を, `read` 関数を使って読み込むもので, 以下で定義されている。

```
1 void read_bytes(int fd, void * a, long bytes) {
2     long n_read = 0;
3     while (n_read < bytes) {
4         ssize_t m = read(fd, a + n_read, bytes - n_read);
5         if (m == -1) { perror("read"); exit(1); }
```

```

6     if (m == 0) { fprintf(stderr, "reached EOF\n"); exit(1); }
7     n_read += m;
8 }
9 }

```

- `cmp_long` は要素 2 つの大小を比較する関数であり, 以下で定義されている.

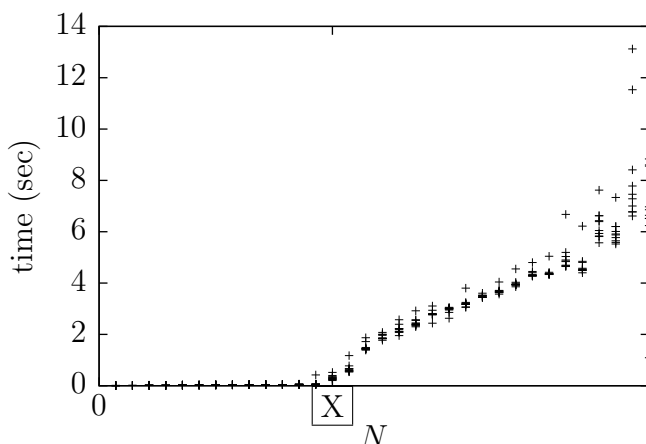
```

1 int cmp_long(const void * a_, const void * b_) {
2     long a = *(long*)a_;
3     long b = *(long*)b_;
4     if (a < b) return -1;
5     if (a > b) return 1;
6     return 0;
7 }

```

スギちゃん: 走らせるぜえ.

ここでスギちゃんは色々な大きさのデータに対して `find_num` を呼び出した. 横軸に要素数 N , 縦軸に `find_num` が実行を開始してから終了するまでの時間をグラフにしたところ以下のような結果が得られた.



測定方法の詳細は以下である.

- N を, 適当な数からはじめて少しずつ増やす.
- 各 N に対し, N 要素からなるファイル F を生成する. 同じ x に対して 10 回, コマンド `find_num F x` を続けて起動する.
- グラフの各点が一回の測定を表している

なお, このマシンは主記憶を 256MB, 2 次記憶はハードディスクを 500GB 搭載しており, `find_num` 以外にはほとんどプログラムは動作していないとする.

スギちゃん: おかしいぜえ. 二分探索は計算量が (a) のはずだぜえ. でも全然そう見えないぜえ. しかもこのプログラムは, (b) N が X を超えたあたりで, 急に挙動が変わるぜえ(グラフ中の X を参照).

そこへ多田という学生がやってきて, 唄い出した.

多田: ♪あたりまえー, あたりまえー, あたりまえコンピュータ♪

♪ N がでかくて read を使うとー... 遅いよ♪

♪当たり前コンピュータ♪

(c) ♪ N でかくても mmap 使うとー... 速いよ ♪

♪当たり前コンピュータ♪

スギちゃん: うーん, 本当か? だったらやり方を知りたいぜえ...

- (1) (a) に当てはまる適切な式を答えよ. $O(\cdot)$ 記法を用いて要素数 N の式として書け.
- (2) 下線部 (b) で, N が X を超えたあたりで急に遅くなったのはなぜか? N が X より十分小さい時, およびそれより大きい時, コンピュータ内で何が起きているのかを具体的に明らかにしながら説明せよ.
- (3) X の値はだいたいどのくらいの値だったと推測されるか? 根拠と共に述べよ.
- (4) グラフの X の左側は, 右側に比べるとほとんど 0 にしか見えないが, そこを拡大すると, 実際にはグラフはそこでどのような形をしているか? グラフの形を書き, そうなる理由も述べよ.
- (5) 下線部 (c) で言われている, mmap を用いたプログラムへの書換えを実際に行なえ. 上記の main 関数を元に, 変更点を簡潔に書け. 元々の main 関数と共通の場所をいちいち書く必要はない. 参考として mmap の API を以下に示す.

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

- (6) mmap を用いたプログラムを使って同じ実験をして, グラフを書くとどのようなになるか? 問題文に現れた read を用いた場合のグラフとの違いがわかるよう, 両者を同一のグラフに書き, 必要な説明を加えよ. 特に, N が X より小さい領域でも, 両者の違いがわかるよう, 適切に拡大して書くなどせよ. また, そうなる理由を述べよ. (1) と同様, コンピュータ内で何が起きているのかを具体的に明らかにしながら説明せよ.

付録: bsearch マニュアルページ (抜粋)

NAME

bsearch - binary search of a sorted array

SYNOPSIS

```
#include <stdlib.h>
```

```
void *bsearch(const void *key, const void *base,  
              size_t nmemb, size_t size,  
              int (*compar)(const void *, const void *));
```

DESCRIPTION

The `bsearch()` function searches an array of `nmemb` objects, the initial member of which is pointed to by `base`, for a member that matches the object pointed to by `key`. The size of each member of the array is specified by `size`.

The contents of the array should be in ascending sorted order according to the comparison function referenced by `compar`. The `compar` routine is expected to have two arguments which point to the key object and to an array member, in that order, and should return an integer less than, equal to, or greater than zero if the key object is found, respectively, to be less than, to match, or be greater than the array member.

RETURN VALUE

The `bsearch()` function returns a pointer to a matching member of the array, or `NULL` if no match is found. If there are multiple elements that match the key, the element returned is unspecified.

解答

- (1) $O(\log N)$
- (2) このプログラムは、ファイルのデータをすべて (read を呼ぶことにより) メモリに読み込んでいる。 (X) が小さい時は、ファイルの中身がすべてメモリ上にキャッシュされている。そのためこの読み込みにかかる時間はメモリ上 (ファイルキャッシュからプロセスのアドレス空間へ) のコピーの時間である。 (X) が大きい時はメモリ上にデータを保持することが出来ず、プログラムを起動するたびに 2 次記憶 (ディスク) からデータを読み出すことになる。
- (3) 128MB (コンピュータのメモリサイズの半分) 程度。 find_num は a に F のデータを読み込むため、128MB 分の物理メモリを消費する。 read 終了後、物理メモリは 128MB 以下しか空いていない。つまり、F の一部は OS のファイルキャッシュから追い出される。そのため find_num がもう一度起動されるとその部分は 2 次記憶 (ディスク) から読み出されることになる。
- (4) 線型。 ファイルをすべて読み込むのにかかる時間 ($O(N)$) + 検索時間 ($O(\log N)$)
- (5) 7,8 行目を削除していかに置き換えればよい。

```
1 long * a = mmap(NULL, sz, PROT_READ, MAP_PRIVATE, fd, 0);
```

- (6) グラフ上でのポイントは 2 つ
- (X) の左でのグラフは $O(\log N)$ だけであり、そこを拡大してもなお、read を使った場合よりはるかに速い
 - しかも、read を使った場合のように、ある点から先、急に遅くなることはない

まず、read と異なり、mmap が呼ばれた時には実際のデータの移動はおこさず、mmap が返す領域をページテーブルでアクセス不可に設定するだけである。その後実際にあるデータにアクセスした時にページフォルトがおき、そのデータのアドレスが、実際にデータのあるページに設定される。したがって、read のように $O(N)$ のコストは発生せず (注 1)、グラフは $O(\log N)$ 、特に read の場合よりもはるかに小さくなる。さらに、消費する物理メモリも純粋に bsearch が触ったページ ($O(\log N)$) の大きさであり、これが物理メモリを越えることは実際問題考えられない (したがってあるサイズを越えると急に遅くなることはない)。

3

C 言語でプログラムを書いて、それをコンパイルした実行可能ファイル `a.out` がある。このプログラムを起動する際、以下のような様々なステップを経て `main` 関数の実行開始までたどり着く。以下は Unix の場合に、このステップを示したものである。

1. `fork` システムコールによりプロセスが生成される。
2. 子プロセスが `exec` システムコールを実行し、アドレス空間が初期化される。
3. `a.out` を子プロセスのアドレス空間に読み込む。
4. `a.out` がリンクしているライブラリが格納されたファイル (共有ライブラリファイル) をアドレス空間に読み込む。
5. `main` 関数の実行を開始する。

Linux, BSD, Solaris などの Unix 系オペレーティングシステムは、小さなプログラムが頻繁に起動されても快適に動くように、上記の処理を様々な工夫によって効率化している。それらについて知る限りを述べよ。説明するそれぞれの工夫について、以下の項目が明らかになるよう整理して説明せよ。

- (a) 上記のどこで行われている工夫か?
- (b) その効果は何か? 高速化 (時間の短縮) か, メモリ消費量の節約, など。
- (c) 具体的にどのような工夫が行われているか? そのような工夫のない, 単純な実装と対比させながら説明せよ。

問題は以上である

解答

アドレス空間の copy on write.

- (a) 1.
- (b) 高速化およびメモリ消費量の節約
- (c) 親プロセスのアドレス空間をまるごとコピーするのではなく、ページテーブル上で(書き込み不可にしながら)物理ページを共有する

mmap によるプログラムの読み込み

- (a) 3, 4.
- (b) 高速化およびメモリ消費量の節約
- (c) プログラムを読み込むのに mmap を用い、必要のない読み込みを省略する。同じプログラムやライブラリを読み込んでいる他のプロセスと、物理メモリを共有することで、メモリ消費も読み込み時間も節約する。