平成25年度オペレーティングシステム期末試験

2014年1月28日 (火)

- **問題は 3 問, 表紙を入れて 9 ページある**
- 解答用紙は1枚. **おもてとうらの両面あるので注意すること**
- 各問題の解答は所定の解答欄に書くこと

1

次の会話を読んで後の, (1) から (7) の問いに答えよ (イラスト: Piro. シス管系女子より. 問題の内容とは関係ありません).

みんとちゃん (以下 M): う〜ん, おかしいなぁ.

大野桜子先輩 (以下 O): みんとちゃん, どうしたの?

M: あ, 大野先輩! お疲れ様です. これ, 私が書いた, ウェブサーバへのアクセスのログの解析をするプログラムなんですけど, これを使って現場で解析をしているオペレータさんたちから「遅すぎる!」って苦情が出て, へこんでるんです〜.

O: どれどれ, 見せてみて.

M: はい, 日付と時刻を 2 つ与えて, その間にあったアクセスの件数を返すんです.

O: てことはどれどれ, 例えば, 去年の紅白の時間帯だったら (カタカタ...),

```
$ count_accesses 2013-12-31-18-00-00 2013-12-31-23-45-00
385
```

385 件ってなるわけね.

M: はい. で, 今みたいに私のマシンでこのコマンドを実行すると, 大概の場合, 1 秒もせずに答えが帰ってくるんです. でも, オペレータたちが言うには, 20 秒以上かかることもざらだって.

O: ふーん, データはどこにあって, どのくらいの大きさで, どんな形式で格納されてるの?

M: はい, まず私の開発用のマシンと, オペレータたちが使うマシンとがあって, 両方に同じものが置かれています. マシンの種類は同じです. メモリはどちらも 2GB ほど搭載しています.

O: データの形式は?

M: 一個のアクセスにつき, アクセスの時刻を表す `seconds_since_epoch` っていうフィールドと, その他の情報が格納されたレコードがあって, 一個のレコードはぴったり 256 バイトになっています. ウェブサーバの生のログをこの形式に変換するプログラムを別に作って, それを使ってこの形式に変換しています. `seconds_since_epoch` は, ある基準の時点からの秒数です.

```
typedef struct {
  long seconds_since_epoch;
    ... ;                    // 以降，詳細省略
} access_record;
```

O: ファイルにはこの形式のレコードが並んでいるわけね. レコードの数はどのくらい?

M: 200 万くらいです. で, レコードは**時刻の昇順**にならんでいます. なので, ファイルを配列に読み込んだら, 「2 分探索」で $O(\log n)$ で検索しています. あ, n はレコードの数です. こんな感じです.

```
int main(int argc, char ** argv) {
  long t0 = convert_to_abstime(argv[1]);
  long t1 = convert_to_abstime(argv[2]);
  long n = n_records_in_file("records");
  int fd = open("records", O_RDONLY);
  long sz = sizeof(access_record) * n;
  access_record * A = (access_record *)malloc(sz);
  long r = read(fd, A, sz);
  long n_founds = binary_search(A, n, t0, t1);
  printf("%ld\n", n_founds);
  return 0;
}
```

注: `convert_to_abstime` は, 2013-12-31-23-45-00 のような形式の時刻を, 基準点からの秒数に変換する関数, `n_records_in_file` は, ファイルのサイズを調べ, ファイル中レコードの数を求める関数である. ともにそれらにかかる時間はほとんど無視できる.

O: じゃ, まずはレコードの数を色々変えて測ってみましょ.

M: ... $O(\log n)$ のアルゴリズムなんだから, 爆速のはずです...

みんとは, 横軸に検索対象のレコード数 (上記プログラムの `n`), 縦軸にプログラムが起動してから終了するまでの時間をとって, グラフを書いた. `n` は最大で, 200 万くらいまで増やした. すると, ...

M: じぇじぇじぇ, ぜんぜん $O(\log n)$っぽくない!

O: ね, まずはちゃんと測定してみないとね. これはどちらかというと $O(\log n)$ じゃなくて, [(1)] のグラフだね. で, そうなっている理由は, [(2)]

M: 言われてみれば当たり前ですね... で, でも, n が 200 万でも, 実際にかかっている時間は, 250 ミリ秒くらいです. オペレータさんたちが言うような, 20 秒とか, 全然そんなんじゃありません. やっぱりこれって, 他に原因があると思うんですけど...

O: そうね. オペレータさんたちは, どんなふうにこのプログラムを利用しているの?

M: はい, 10 人くらいのオペレータさんが使っています. データが置かれているマシンにウェブサーバをたてて, ウェブページ経由で, 日付と時刻を入力します. そうするとそれを受け取ったウェブサーバが私のプログラムを起動するという仕組みです.

O: へぇ, ってことは, どういうタイミングでいくつ, みんとちゃんの書いたプログラムが起動されるかわからないってことね. だったら, 「20 秒かかることもざら」っていうのも, わかる気がするわ.

M: え, どうしてですか?

O: (3)

M: そ, そうなんですか. 先輩, 私はどうしたらいいんでしょう?

O: 今回みたいなケースに, 実に簡単な解決方法があるんだな. (4) を使えばいいよ.

M: エ, エンマ君? ··· 

O: どんな聞き間違いよ, それ··· みんとちゃんのプログラムをこんなふうに書き換えればいいのよ.

```
1  int main(int argc, char ** argv) {
2    long t0 = convert_to_abstime(argv[1]);
3    long t1 = convert_to_abstime(argv[2]);
4    long n = n_records_in_file("records");
5    (5)
6    long n_founds = binary_search(A, n, t0, t1);
7    printf("%ld\n", n_founds);
8    return 0;
9  }
```

M: で, でもどうしてこうすると, オペレータさんの環境で遅い問題が解決されるのでしょうか?

O: それは, (6).

M: ふーん, なんかよくわかりませんが, とりあえず現場に送ってみます.

(しばらくして)

M: あ, オペレータさんからメールで反応がかえって来ました. (メールを読みながら) え, すごい, n が 200 万でも, 40 ミリ秒くらいですって! 遅くならないどころか, さっきまでの私のマシンでの結果よりも断然速い!

O: そうそう, (4) を使って書きなおしたやつは, さっきまでの (1) の性能ではなく, 本当にほぼ $O(\log n)$ と言って良い性能になるのよ. それも (4) がファイルを読む仕組みと関係あるのよ. (7).

M: ふーんなるほどね. うまくできてますね. [(4)], えらいぞっ! あ, そうだ, こないだ書いたウェブサーバの生のログを変換するプログラムも, [(4)]を使って書き換えちゃおーっと

O: あ, いや, いつでも使えばいいってもんじゃ無いんだけど... (でも説明してると長くなるから今日はまあいっか ^^;)

(1) [(1)]に当てはまる計算量を答えよ.

(2) [(2)]には, 前問の答えがそのようになる理由が入る. 適切な文章を答えよ.

(3) [(3)]には, みんとちゃんのプログラムが, 「20 秒かかることもざら」になる理由が入る. 適切な文章を答えよ. その際, みんとちゃんと大野先輩の実験ではせいぜい 250 ミリ秒だったのに, オペレータさんたちの環境では 20 秒かかることもざらだった理由がわかるように説明した文章を入れよ.

(4) [(4)]に当てはまる単語を答えよ.

(5) [(5)]に当てはまる適切なプログラムの断片を答えよ. 1 行とは限らない. 呼び出す API の詳細 (引数の順番など) が不安な場合は適宜コメントで補え.

(6) [(6)]には, [(4)]を使うと, 「20 秒かかることもざら」だった問題が解消された理由が入る. 適切な文章を答えよ.

(7) [(7)]には, 大野先輩のプログラムでは, 本当にほぼ $O(\log n)$ の計算量が達成される理由が入る. 適切な文章を答えよ.

2

今日のオペレーティングシステムを構成するにあたって, CPU に備わる MMU は必須の機能である. MMU を用いて実現されている以下の機能について, その実現方法の概要, MMU がそこでどのような役割を果たしているかについて述べよ.

(1) fork() システムコール (プロセス生成) の高速化

(2) LRU を近似したページ置換アルゴリズムの実現

(3) プロセス間共有メモリ

3

以下の find_primes_serial(n, $primes$) は, n 未満の素数を全て求め, 配列 $primes$ に格納し, その個数を返すプログラムである. $primes$ は, n 未満の素数を格納するだけの十分な領域を持っているものとする (配列の添字あふれは気にしなくて良い).

```
1  long find_primes_serial(long n, long * primes) {
2    long n_primes = 0;
3    long next_p = 2;
4    while (next_p < n) {
5      long p = next_p++;
6      if (is_prime(p)) {
7        long idx = n_primes++;
8        primes[idx] = p;
9      }
10   }
11   return n_primes;
12 }
```

ここで 6 行目の is_prime(p) は, $\sqrt{p}$ 以下の整数全てで p を試し割り算して, 素数かどうかを判定する関数で, 例えば以下である.

```
1  int is_prime(long p) {
2    long x;
3    for (x = 2; x * x <= p; x++) {
4      if (p % x == 0) return 0;
5    }
6    return 1;
7  }
```

find_primes_serial を並列化することを考える. 具体的には, 複数のスレッドが並行して, 4 行目から始まる while 文を実行する. スレッド間で仕事や結果を共有するために, n, primes, n_primes, next_p は大域変数にする. つまり, 各スレッドが実行する以下の関数 (worker) を作り,

```
1  long n, n_primes, next_p;
2  long * primes;
3  void * worker(void * _) {
4    while (next_p < n) {
5      long p = next_p++;
6      if (is_prime_x(p, n_primes)) {
7        long idx = n_primes++;
8        primes[idx] = p;
9      }
10   }
11   return 0;
12 }
```

本体は以下のようにする (N_THREADS はスレッドの個数).

```
long find_primes_para(long n_, long * primes_) {
  /* 大域変数でデータを共有 */
  n = n_;
  primes = primes_;
  n_primes = 0;
  next_p = 2;
  /* N_THREADS 個, スレッドを起動 */
  int i;
  pthread_t tid[N_THREADS];
  for (i = 0; i < N_THREADS; i++)
    pthread_create(&tid[i], NULL, worker, 0);
  /* スレッドの終了待ち */
  for (i = 0; i < N_THREADS; i++)
    pthread_join(tid[i], NULL);
  return n_primes;
}
```

以下の問いに答えよ.

(1) この, find_primes_para を実行した結果, 以下のような間違いが生じ得るか否かを答えよ (解答欄の生じ得る, 生じ得ないのいずれかに○をつけよ). 生じ得ると答えた場合, それが生ずる実行例を具体的に示せ.

(a) 偽陽性. つまり, 素数でないものが配列 primes_ に格納された状態で終了する.

(b) 偽陰性. n 未満の素数が, 配列 primes_ に格納されていない状態で終了する.

(c) 終了しない.

(2) (1) で述べた問題点を解消するため, 排他制御を用いた解決方法を示せ. プログラムのどこをどのように直したら良いかはっきりと示せ. 必要ならば適宜変数や関数を追加して良い. ただし, 渡された配列 primes_ に, どのような順番で素数が格納されるかは問わない.

(3) (1) で述べた問題点を解消するため, **排他制御を用いず,** compare&swap 命令を用いた解決方法を示せ. compare&swap 命令が以下の関数を呼び出すことで使用可能であるとし, 必要ならば適宜変数や関数を追加して良い. 渡された配列 primes_ に, どのような順番で素数が格納されるかは問わない.

bool cmp_and_swap(T * p, T a, T b);

ここで, T は, int または long とする. この関数は, 以下に相当する動作を不可分に行う

```
bool cmp_and_swap(T * p, T a, T b) {
  if (*p == a) { *p = b; return 1; }
  else         {         return 0; }
}
```

(4) 上記の is_prime(p) 関数は, 2 以上 $\sqrt{p}$ 以下の全ての数で試し割り算をしているが, 実際には 2 以上 $\sqrt{p}$ 以下の全ての**素数**で試し割り算をすれば十分である. これに注目して, 配列 primes にすでに格納されている素数を使って試し割り算をするよう, 以下のように is_prime を書き換えたとする.

```
1 int is_prime(long p) {
2   long i;
3   for (i = 0; i < n_primes; i++) {
4     if (p % primes[i] == 0) return 0;
5     if (primes[i] * primes[i] > p) return 1;
6   }
7   return 1;
8 }
```

このように変更したプログラムで生じ得る間違いについて, (1) と同様に答えよ. ただし, (2) または (3) の修正はすでに施されているものとする. ここでも, 渡された配列 primes_に, どのような順番で素数が格納されるかは問わない.

(5) find_primes_para を正しくするためにさらに必要な修正について, 概要を述べよ. 渡された配列 primes_に, どのような順番で素数が格納されるかは問わない.

問題は以上である