平成28年度オペレーティングシステム期末試験

2017年1月24日 (火)

- **問題は 3 問**
- **この冊子は，表紙 1 ページ (このページ), 問題 2-10 ページからなる**
- 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問いに答えよ．[1]

谷町さん (以下 T): じゃあみんな，これから OS の試験の傾向と対策をやるぞ．桜子君，よろしく．

大野桜子先輩 (以下 O): あ，その前にひとつ宣伝をさせてください．2/8 に Google エンジニアの Parisa Tabriz さん [2] の講演があります．場所は多分，工学部 2 号館のどこかになります．多分，セキュリティに関してと，女子 IT 教育について話してくれると思うわ．詳細未定だから，掲示板か，OS 講義のホームページをチェックしてね．

利奈みんとちゃん (以下 M): 2/8 か〜．もう試験も終わってますね．3 年生が終わり，これから晴れて卒論生だって時に，一日くらい授業がなくても大学に来るのもいいですね．

T: そうだ．セキュリティも，女子 IT 教育もどちらも大事な話だからな．電子情報を始めとする情報系の学科が女子学生に人気がないというのは，決して日本だけの問題だけではない [3] が，アメリカでも日本よりはマシ，[4] インドのように半分近くが女子という国もある．[5] 東大の情報系学科は，日本全体の電気通信の平均値よりもなお低い．

M: 今の時代，就職にも有利だし，わりと直接，世の中の役にも立てるのにねぇ．

O: 仕事をする場所や時間だって，本来は融通が聞くはずだから，ワークライフバランスだって本来は取りやすいはずですよね．

T: そうなんだ．しかし実際にそうなっていないところが日本の職場，もしかしたら文化の問題だよね．さて，今はこのくらいにして，桜子君の講義を始めようか．

O: はい，OS なしにはセキュリティも何も語れませんからね．まず，τ 先生の OS の試験は毎年いろいろ変わった問題が出ているように見えるけど，よく見るとワンパターンなのよ．特に必ず一問は，[(a)] に関する問題が出るの．

M: ギソウケッコン?

O: おいおい，ほとんど原形をとどめてないわね．

M: あー，そう言えばなんか聞いた気がしますけど，全くわかりませんでした．

O: ええっとじゃあ，コンピュータにはメモリが付いているってのは知ってるよね．

M: あー，はい，4GB とか，8GB とか．

O: そう，Bっていうのは,「バイト (byte)」の頭文字で，1 バイトっていうのは，8 bit のこと．G は $2^{30} \approx 10^9$ だから，8GB だと，約 80 億バイトを記憶できるできることになるの．で，当然のことながら 8GB のメモリを読んだり書いたりするときには,「どの」1 ないし数バイトを読みたいかっていうのを指定しないといけないよね．それを指定する数字のことを，[(b)] っていうんだったよね．

M: そうでした．

---

[1] この話のキャラクタは，シス管系女子（原作 Piro）のキャラクタを利用させていただいています．http://system-admin-girl.com
[2] https://research.google.com/pubs/author36241.html
[3] http://tinyurl.com/hfubgbj “The gender gap in computer science is hurting U.S. businesses”
[4] e-stat http://tinyurl.com/2pcs5n によれば,「電気通信工学」「機械工学」など分野で 10%程度
[5] http://tinyurl.com/jl47ykf “Decoding Femininity in Computer Science in India”

O: で，この (b) には実は 2 種類あるの．実際に，8GB のメモリにある，80 億のバイトのどこを読むかを指定する (b) は，(c) というの．でも実際にプログラムの中で使っている (b) はそれとは別で，(d) というのよ．

M: あ〜，そのへんから訳がわからなくなるんですよね．そもそも私，C でプログラムを書いてますけど，メモリをアクセスしているなんて，意識してないんですけど．

O: うーん，まあ，そのへんを意識せずにかけるようになるのが高級言語の役割でもあるから，意識してない事自身は OK なんだけどね．でも，普通の変数でも配列でも，代入した値を覚えるために，データは (e) かメモリに入ってないといけなくて，(e) は容量が小さいから，ほとんどのデータはメモリのどこかに入っているわけよ．で，C 言語の場合，あるデータがおいてある (d) を，知ることもできるよ．例えば以下のプログラムで，変数 x の (d)，それと，5 行目の malloc で確保された領域の (d) を表示したければ，こんなプログラムを書けばいいわ．

```
#include <stdio.h>
#include <stdlib.h>
int x;
int main() {
  char * a = malloc(100);
  (f)
  return 0;
}
```

このプログラムをコンパイルして，実行するよ，それ!

```
$ gcc -Wall print_addr.c
$ ./a.out
x : 6295620, malloc : 20836368
```

ここで表示されているそれぞれの数字が，(d) なのよ．

M: それと，(c) とはどういう関係にあるのでしょうか?

O: CPU は，すべてのメモリアクセス時に，(g) というデータ構造，と，その一部をキャッシュしている (h) を参照して，(d) を (c) に変換して，メモリをアクセスしているのよ．この変換機構を実装しているハードウェアを，(i) と呼んでいるよ．

M: 何のためにそんなややこしいことをするのですか? 素直に，(d) = (c) にしてしまえば，単純で，余分な仕組みも要らなくていいんじゃないでしょうか（CPU もちょっとは速くなるだろうし）．

O: うん，それが一番大事なところだよね．やっぱりいちばん大きい理由は，(j) だね．これがないとセキュリティも何もないからね．

M: う〜ん，具体的にはどのように，(j) が達成されるのでしょうか?

O: (k)

M: なるほど〜，なんとなくわかりました．

O: また，(i) があるおかげで，(l) 物理メモリを越えるメモリ割り当て をすることもできるようになるよ．その自然な延長で，プロセスからメモリ割り当て要求を受けた際，その場ですぐに物理メモリを割り当てず，実際にアクセスが合った時に初めて物理メモリを割り当てる，(m) という方式も可能になるよ．

M: なるほど〜，物理メモリを越えるメモリを割り当てられるなんてすごいですね．なんとなく (i) の意義がわかってきました．

O: もっとすごいこともできるよ．例えば，ファイルを効率的に読み書きできる，(n) というものがあるわ．その仕組みはね，(o) ．

M: うぁー，なんか，コンピュータも奥が深いぞって思えてきました．

O: そうなのよ．とくに (i) は，普段あまり目に見えていないし，一見そんなことをする必然性がわかりにくいんだけど，現代の計算機環境を構成する，超重要技術の一つなのよ．最近は，GPU とか FPGA とか，余分なものをそぎ落として計算を効率化・高速化するということの重要性が高まってきていて，高速化のためには (i) なんてものは要らない，なんて言う人もいるんだけど，(i) が (p) 高速化に果たしている役割を無視して語ると，後で痛い目に合うとおもうわ．

M: ありがとうございました．ところで今は，τ 先生の OS の試験の傾向と対策の時間ですが，他にはどんな問題が出るんですか?

O: そうねぇ，レパートリは (a) に関する問題，スレッドの競合状態，ファイルの読み込みとかキャッシュとかに関する問題，資源の保護に関する問題，スレッドのスケジューリングに関する問題，色々な場合のメモリ使用量に関する問題，くらいしかないのよ．あと，形式的には，会話に沿って言葉を答えさせたり，説明をさせる問題，実際のプログラムを読ませたり書かせたりする問題，あとは自由記述形式の問題，を混ぜてることがほとんどね．最近は会話形式がないこともあるけどね（きっとネタが思いつかなかったんだろうね）．

M: ずばり今年の予想は?

O: ずばり，(a) に関する問題が会話形式で，スレッドの競合状態に関する問題がプログラム形式で，さらに資源保護に関する問題が自由記述形式で出るわ．間違いない．

M: お〜，じゃあ，これをツイートしときます!

O: これを読んでる今じゃもう遅いけどね（笑）！

- (a) 〜 (e) に当てはまる単語を答えよ．
- (f) に入る適切なプログラム片を答えよ．
- (g) 〜 (i) に当てはまる単語を答えよ．
- (j) に当てはまる言葉 (短い句) を答えよ．
- (k) に適切な，(j) を達成するための仕組みを文章で答えよ．
- 下線部 (l) を達成するための仕組みを，(i) がそのために備えている仕組みを踏まえながら答えよ．
- (m) に当てはまる単語を答えよ．

- (n) に当てはまる単語（一般的な単語，または，システムコール名）を答えよ．
- (o) に適切な，(n) を実現するための仕組みを文章で答えよ．
- 下線部 (p) にある，(i) が計算機の高速化に貢献している例をひとつ述べよ．**何の**高速化に貢献しているかと，**どのように**その高速化が達成されているか，を述べよ．

2

(1) 以下の会話を読んでその次の問いに答えよ．

代官 (以下 D): 越後屋，例のものは手に入れたか?

越後屋 (以下 E): はいお代官様，共用計算機のアカウントを手に入れました．

D: よし，この共用計算機は教員も使っている計算機じゃ．これで今度の試験問題も覗き放題じゃ．

E: で，ですがお代官様，単にアカウントを手に入れただけでは，他のユーザのファイルを覗き見ることはできないようになっています．

D: な，なに，それが南蛮から渡来したオーペーなんとかの力なのか?

E: オペレーティングシステムでございます．ファイルは，システムコールを使って読まれますが，オペレーティングシステムがそこでしっかりとファイルの読み書き権限をチェックします．

D: え，え〜い，なんとかならんのか，越後屋!?

E: お代官様，拙者に知恵がございます．ファイルはハードディスクという金物に入っております．オペレーティングシステムとて単なるソフトウェア．システムコールもチェックをした後でハードディスクに命令を出して，読んでいるにすぎません．したがって，オペレーティングシステムが発行するのと同じ命令を出すプログラムを自分で書いてしまえばよいのです．

D: だ，だが越後屋，そのオペレーティングシステムとやらがどんな命令を出しているかはどうやって知るのじゃ?

E: このオペレーティングシステムは，オープンソースと言いまして，ソースコードが公開されております．ここにその，ソースコードを収めた，巻物がございます．これをそっくり真似すれば，直接ハードディスクに命令を出せるというわけです．

D: 越後屋，おぬしも相当のワルよのう．

E: 何をおっしゃいますか，お代官様こそ．

D, E: うわーはっはっはっはっ...

越後屋は大きな誤解しており，このような試みは成功しない．オペレーティングシステムが，周辺機器を保護しつつも，正当な呼び出し手順（システムコールの呼び出し）を経由すれば周辺機器を利用可能にしている仕組みを，どのように**「保護」と「利用可能」を両立させているのかに注目しながら**，説明せよ．

(2) 以下の会話を読んでその次の問いに答えよ．

代官 (以下 D): え〜い，試験問題を盗み見ることはできなかった．こうなったらせめて，試験問題ができないよう，共用計算機を使えなくしてしまえ．越後屋〜

越後屋 (以下 E): はい，お代官様

D: ここに，ひたすら CPU を使うプログラムがある．

```
1 int main() {
2   for ( ; ; ) {  }
3 }
```

ワシのパソコンでこれを立ち上げると，CPU が 100%消費される．共用計算機でこれを立ち上げてしまえば，あの OS の試験を作っておる，ほれ，なんとかいう，教員の作業はできなくなるはずじゃ．

E: タウラでございます．ですがお代官様，ご覧の通り，CPU が 100%消費されていても，マウスは動くし，ctrl-c でそのプログラムを終了させることもできてしまいます．

D: それはなぜなのだ，越後屋!?

E: ひとつには，今時のパソコンは，お代官様のものを含め，ひとつのみならずたくさんの CPU を搭載しているということがございます．また，C 言語で書かれたプログラムは，C のコンパイラによって機械語に変換されているということもございますので，無限ループを書いたつもりでも CPU を独占できていないという可能性もございます．

D: では一体どうすればよいのだ?

E: 機械語でプログラムを書き，そのプログラムを多数走らせるというアイデアはいかがでございましょう?

D: 越後屋，おぬしも相当のワルよのう．

E: 何をおっしゃいますか，お代官様こそ．

D, E: うわーはっはっはっはっ...

ここでも越後屋は大きな誤解しており，彼らの試みは成功しなかった．オペレーティングシステムが，特定のプロセスによって CPU が独占され，他のプロセスを実行できなくなることがないようにしている仕組みを述べよ．

(3) 以下の会話を読んでその次の問いに答えよ．

代官 (以下 D): 共用計算機を使えなくすることもできんのか．こうなったらせめて少しでも，なんとかいう担当教員 (まだ覚えていない) の作業を邪魔することができないものか．越後屋〜

越後屋 (以下 E): はい，お代官様

D: CPU を独占することはできなくても，その担当教員が試験問題を作る時の道具，ほれ，イーなんとかいうプログラム

E: イーマックス (Emacs) でございます．

D: そうそれじゃ，その動作を邪魔するとか，キー入力に反応するまでの時間を長くするとか，そういうことができんのか?

E: お代官様．たしかに，OS はタイムシェアリングをしていますので，先ほどのプログラムをじゃんじゃん走らせれば，そのようなことは可能かと．

D: 越後屋，おぬしも相当のワルよのう．

E: 何をおっしゃいますか，お代官様こそ．

D, E: うわーはっはっはっはっ...

またしても越後屋の浅い理解により，彼らの試みは成功しなかった．オペレーティングシステムが，独占を防ぐのみならず，高負荷時においても対話的なプログラムの応答性が損なわれないようにしている仕組みを述べよ．

3

n 未満の素数をすべて求める並行プログラムを作りたい．ある数 x が素数であるか否かを判定するには，x を $\sqrt{x}$ 以下のすべての素数で割ればよい（余りが 0 になるものがひとつもないとき及びその時に限り，x は素数である）．

まず逐次プログラムの説明をする．以下は計算の途中状態を管理するための構造体である．

```
typedef struct {
  long * primes; /* これまでに見つかった素数の配列 */
  long n_primes; /* これまでに見つかった素数の数 */
  long next;     /* 次に素数かどうかチェックする数 */
} prime_gen;
```

main 関数ではこの構造体をひとつ以下のように初期化し，gen_primes という関数を呼ぶ．なお $n \geq 55$ のとき，n 未満の素数の数は，

$$1 + \frac{n}{\log n - 4}$$

個以下であることが知られており，以下ではそれを用いて配列の要素数を割り当てている．

```
int main() {
  long max_n_primes = (n < 55 ? n : 1 + n / (log(n) - 4));
  long * primes = (long *)calloc(max_n_primes, sizeof(long));
  prime_gen pg;
  pg.primes = primes; // 素数を格納する配列(初期状態 :空)
  pg.n_primes = 0;    // 素数の数(初期状態: 0)
  pg.next = 2;        // 最初は 2から判定開始
  gen_primes(&pg, n); // 本題
  return 0;
}
```

gen_primes が主要な関数で，初期状態の prime_gen 構造体へのポインタ pg と，整数 n を受け取り，n 未満の素数を pg（の中の primes 配列）に格納する．そのために gen_primes は再帰的関数になっており，n がある程度大きい時は，まず $\sqrt{n}$ 未満の素数をすべて生成し，それを用いてそれ以上の数が素数か否かを判定する．$n \leq 3$ の時は自明（2 以上 n 未満の数が全て素数）である．以下で，$\lceil x \rceil$ は，x 以上の最小の整数を表す．

```
void gen_primes(prime_gen * pg, long n) {
  if (n <= 3) {
    gen_primes_until(pg, n);    /* 自明 */
  } else {
    long s = ⌈√n⌉;
    gen_primes(pg, s);        /* √n 未満を生成 */
    gen_primes_until(pg, n); /* n 未満を生成 */
  }
}
```

gen_primes_until(pg, n) は，pg にすでに $\sqrt{n}$ 未満の素数が格納されている $(*)$ という条件のもと，n 未満の素数を生成する関数で，以下の通り．

```
void gen_primes_until(prime_gen * pg, long n) {
  /* √n 未満は格納されている条件のもと，n 未満の素数を生成 */
  while (1) {
    long x = get_next(pg, n);
    if (x == n) break;          /* 終了 */
    if (is_prime(pg, x)) {      /* 素数? */
      add_prime(pg, x);         /* であれば追加 */
    }
  }
}
```

get_next は，次に判定すべき数を返す関数で，next を返しつつ，n に達しない限り next を 1 増加させるだけである．

```
long get_next(prime_gen * pg, long n) {
  /* 次に判定すべき数を返す */
  long x = pg->next;
  if (x == n) return n;
  pg->next = x + 1;
  return x;
}
```

is_prime は，先と同じ条件 $(*)$ のもと，x が素数か否かを判定する (素数であるときに 1, そうでないときに 0 を返す) 関数である．

```
int is_prime(prime_gen * pg, long x) {
  long np = pg->n_primes;
  /* これまでに見つかった√x 以下の素数で試し割る */
  for (long i = 0; i < np; i++) {
    long p = pg->primes[i];
    if (p * p > x) break;
    if (x % p == 0) return 0;   /* 約数が見つかった */
  }
  return 1;
}
```

また，add_prime(pg, x) は，x を素数の primes 配列の末尾に追加する関数で，以下の通り．

```
void add_prime(prime_gen * pg, long x) {
  /* x を素数の配列に加える */
  long np = pg->n_primes;
  pg->primes[np] = x;
  pg->n_primes = np + 1;
}
```

さて今このプログラムを並列に実行して高速化するために，gen_primes_until を複数のスレッドで実行する．

OpenMP の parallel 構文:

```
#pragma omp parallel
  文
```

は，ある数（通常，搭載プロセッサ数．本問題では複数であるという以外，重要ではない）のスレッドを生成し，各スレッドが（同じ）「文」を実行する（スレッドが実行するのは**#pragma omp parallel の直後の一文**のみ）．全スレッドがその文の実行を終えたら，parallel 構文全体の実行が終了し，その次の文に移る（それは再び一つのスレッドだけで実行される）．

これを用いて，gen_primes 内の gen_primes_until(pg, n) を各スレッドが実行するように，プログラムを変更した．

```
void gen_primes(prime_gen * pg, long n) {
  if (n <= 3) {
    gen_primes_until(pg, n);    /* 自明 */
  } else {
    long s = ⌈√n⌉;
    gen_primes(pg, s);        /* √n 未満を生成 */
#pragma omp parallel
    gen_primes_until(pg, n); /* n 未満を生成 */
  }
}
```

このプログラムには色々な間違いがある．以下ではその間違いを一つずつ考察し，修正する．なお，目標は配列 primes に n 未満の素数が漏れ・重複なく並ぶことであり，順番は必ずしも逐次プログラムで実行した時のそれと同じ（昇順）でなくても良い．

(1) 間違いのひとつとして，get_next 関数が，同じ値を 2 度以上返してしまう（結果として同じ値が 2 度以上，primes 配列に格納されてしまう）というものがある．どのようにそれが生じずるかを，あり得る実行順序を具体的に説明し，それが起きないように，get_next 関数を修正せよ．なお，もし必要ならば，prime_gen 構造体に要素を追加し，main 関数に適切な初期化文を追加をしてもよい．

(2) 次の間違いとして，add_prime で加えた数が，終了時に primes 配列に入っていない，ということも起きうる．前問題同様，それを生ずる実行順序を具体的に説明し，それが起きないように，add_prime を修正せよ．もし必要ならば，prime_gen 構造体に要素を追加し，main 関数に適切な初期化文を追加をしてもよい．

(3) 前問までの修正を施したプログラムにおいて,「素数でない数が素数と判定されてしまう」という間違いが起きうるか? 起き得るならばそれを生ずる実行順序を具体的に説明し，起き得ないならばその理由を説明せよ．

問題は以上である

学生証番号		氏名	

1	(a)	仮想記憶	(b)	アドレス	(c)	物理アドレス	(d)	論理アドレス	(e)	レジスタ
	(f)	printf("x : %d, malloc %d¥n", &x, a);								
	(g)	ページテーブル	(h)	TLB	(i)	MMU				
	(j)	プロセス間のメモリの分離								
	(k)	異なるアドレス空間（プロセス）に対応する物理アドレスが重複しないようにページテーブルを設定する.								
	(l)	一部の論理ページには, 対応する物理ページが不在, という情報をセットしておく. プロセスがそのようなページをアクセスするとページフォルトが発生しOSはその例外を処理して, アクセスされた論理ページに対する物理ページを割り当てる. 必要に応じて他の物理ページをページング領域に追い出し（ページアウト）, アクセスされたページの内容をページング領域から取り戻す（ページイン）.								
	(m)	要求時ページング	(n)	mmap						
	(o)	mmapされた領域を初めてアクセスした際に発生するページフォルトを処理して, アクセスされたページの内容を, 対応するファイルの対応する部分から取り出す.								
	(p)	(1) copy-on-writeによるmmapの高速化. 複数のプロセスが同じファイルの同じ部分をmmapした際, いずれかのプロセスによる書き込みがおこるまで物理メモリを共有することで, ファイルのコピー時間やメモリ使用量を削減する. (2) プロセス生成(fork)の高速化. fork時にメモリをコピーせず, copy-on-writeを設定し, 実際に書き込みがあったらそのページの内容をコピーする. (3) プログラム立ち上げ時のプログラムの読み込みの高速化, 仕組みはプログラムやライブラリをmmapを用いて読み込むだけ. あとは上記(1)と同じ.								

2	(1)	周辺機器を保護する仕組み: 実際に周辺機器にアクセスする命令は特権命令で, 通常の(OSカーネル以外の)ユーザプログラム実行時は実行できない（実行しようとするとCPUによって例外が発生する)
		システムコールを介せば周辺機器が利用可能になる仕組み: トラップ命令という命令があり, 特権モードに移行するとともに, 割り込みベクタを参照して, 割り込みベクタが指定するアドレスへ実行を写す. 割り込みベクタの内容や割り込みベクタのアドレスを指定するアドレスは特権命令でのみ変更できるようにしておくことで, OSの決められたアドレス（システムコールの入り口）にのみ, 特権命令でジャンプすることができる.
	(2)	CPUには外部からの割り込みを受け付けるラインがあり割り込みを受けると割り込みベクタが指定するOS内の番地へジャンプする. かつ周辺機器のタイマが周期的にタイマ割り込みを発生させており, ユーザプログラムが何をしても定期的にOSが制御を奪い, 他のプログラムを実行する機会を持てる
	(3)	各スレッドに公平にCPUが割当たるよう, 各スレッドがCPUを消費した時間(virtual time)を管理し, 次に走るスレッドを決める際, virtual timeが小さいスレッドを優先する. 対話的なプログラムの大クハ普段はほとんどCPUを消費しないため, 実際に実行可能になった際は多くの場合高い優先度になっているため, そのようなプログラムの応答性は, CPUをを多く消費するプロセスの数に余り影響を受けない.

3	(1)	どのように間違いが生ずるか: 2つのスレッドA, Bがほぼ同時にget_nextを実行し，(A) long x = pg->next; (B) long x = pg->next; と，両者が同じ値をnextから読んだ場合

		プログラムの修正:
		一例:
		long get_next(prime_gen * pg, long n) {
		  while (1) {
		  long x = pg->next;
		  if (x == n) return x;
		  if (__sync_bool_compare_and_swap(&pg->next, x, x + 1)) {
		  return x;
		  }
		  }
		}

	(2)	どのように間違いが生ずるか: 2つのスレッドA, Bがほぼ同時にadd_primeの3行目を実行し，(A) long np = pg->n_primes; (B) long np = pg->n_primes; となり，同じnp の値を得た場合. 両者が配列の同じ場所に書き込むことになり，先に書いた値が失われる.

		プログラムの修正:
		一例:
		void add_prime(prime_gen * pg, long x) {
		  while (1) {
		  long np = pg->n_primes;
		  if (__sync_bool_compare_and_swap(&pg->primes[np], 0, x)) break;
		  }
		  __sync_fetch_and_add(&pg->n_primes, 1);
		}

	(3)	起き得る
		どのように間違いが生ずるか，またはなぜ生じないか: これまでの間違いを修正しても，素数が昇順でない順番で格納されることがあり得る. すると，is_primeによる判定の6行目 if (p * p > x) break ; によって，sqrt(x) 未満の素数すべてでxを割るする前にループを抜けて，素数でないものを素数であると判定してしまう事があり得る. より具体的に，n = 10でgen_primesを実行したとすると以下のようなことが起きうる. Gen_primes(&pg, 10); が実行される 再帰呼出しで gen_primes(&pg, 4); が実行される gen_primes_until(&pg, 4); が実行される. 2, 3が（別々のスレッドによって）それぞれ素数と判定されるが，その際，primes = { 3, 2 } の順に格納されうる. Gen_primes_until(&pg, 10)の途中，is_prime(&pg, 4) で，if (3 * 3 > 4) break; によって，素数と判定されてしまう.