# 平成27年度オペレーティングシステム期末試験

2016年1月19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 2-8 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

巨大なメモリを搭載するコンピュータ上で，以下の A0〜A4 までの 5 種類プログラムを走らせることを考える．

**A0:**

```
int main() {
  char * a = malloc(A);
  /* --- */
}
```

**A1:**

```
int main() {
  int fd = open(file, O_RDONLY);
  char * a = malloc(A);
  ssize_t r = read(fd, a, A);
  read_array(a, B); /* 中身は後述 */
  /* --- */
}
```

**A2:**

```
int main() {
  int fd = open(file, O_RDONLY);
  char * a = mmap(0, A, PROT_READ|PROT_WRITE, MAP_PRIVATE, fd, 0);
  read_array(a, B); /* 中身は後述 */
  /* --- */
}
```

**A3:**

```
int main() {
  int fd = open(file, O_RDONLY);
  char * a = mmap(0, A, PROT_READ|PROT_WRITE, MAP_PRIVATE, fd, 0);
  write_array(a, B); /* 中身は後述 */
  /* --- */
}
```

**A4:**

```
int main() {
  int fd = open(file, O_RDWR);
  char * a = mmap(0, A, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
  write_array(a, B); /* 中身は後述 */
  /* --- */
}
```

ここで read_array(a, m) および write_array(a, m) は，それぞれ a から始まる m バイトを読み出すまたは書き込む関数で，以下に相当する動作をする．

```
void read_array(char * a, ssize_t m) {
  for (ssize_t i = 0; i < m; i++) {
    a[i];
  }
}
void write_array(char * a, ssize_t m) {
  for (ssize_t i = 0; i < m; i++) {
    a[i] = 'x';
  }
}
```

(1) A0 を走らせた時消費する**物理**メモリの量 (おおよその値) を答えよ．答えはプログラム中のパラメータ $A$, $B$ を用いた式で表せ．ただし $A$ や $B$ はある程度 (少なくとも数十 M) の大きな値である．A1〜A4 についても同様に答えよ．

以下を仮定してよい．

- システム関数呼び出し (open, malloc, mmap, read) は全て成功する．
- read は要求したバイト数分のデータを実際に読み出す．
- 関数呼び出しやメモリアクセスが実際の計算には不要だからといって，コンパイラの最適化で除去されることはない (プログラムはあくまで書かれた通りのことを実行する)．
- 上記プログラム中で明示的に行われるメモリ割当量に比べて，それ以外に必要な (例えばプログラムのコードやスレッドのスタックのための) メモリ量は小さいと考え，無視する (0 とする)．

なお，消費されるメモリ量は，走らせているプログラムが/* --- */の行に達した時点での消費量とする．

(2) A0 を $P$ 個同時に立ち上げるとどうか? 答えは $A$, $B$, $P$ を用いた式で表せ．消費されるメモリ量は，$P$ 個全てのプロセスが/* --- */に達し (そこでしばらく停止し) たところでの消費量とする．A1〜A4 についても同様に答えよ．

(3) 以下は OS がメモリ管理のために用いてる技法であるが，それぞれどのようなもので，どのような時にどのような利点があるか，簡潔に説明せよ．

(a) 要求時ページング

(b) Copy on write

(4) (2) の A2 および A3 の物理メモリ使用量の結果を，要求時ページング，copy on write という言葉を，関連があれば適宜使いながら，それぞれ説明せよ．

## 2

以下の関数 f0〜f3 はいずれも，あるメモリ領域に $N$ 回 1 を足し，終了後にその値を表示するという関数であり，対象となるメモリ領域だけが異なる (for 文以降はほとんど共通; 重要部に下線が引いてある).

**f0:**

```
int x = 0;
void * f0(void * _) {
  for (int i = 0; i < N; i++) {
    x = x + 1;
  }
  printf("it's %d\n", x);
  return 0;
}
```

**f1:**

```
void * f1(void * _) {
  int * p = (int *)calloc(1, sizeof(int));
  for (int i = 0; i < N; i++) {
    *p = *p + 1;
  }
  printf("it's %d\n", *p);
  return 0;
}
```

**f2:**

```
void * f2(void * arg) {
  int * p = ((arg_t *)arg)->m;
  for (int i = 0; i < N; i++) {
    *p = *p + 1;
  }
  printf("it's %d\n", *p);
  return 0;
}
```

**f3:**

```
void * f3(void * _) {
  int fd = open("/tmp/zero_file", O_RDWR);
  int * p = mmap(0, sizeof(int), PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
  for (int i = 0; i < N; i++) {
    *p = *p + 1;
  }
  printf("it's %d\n", *p);
  return 0;
}
```

ただし，f2 における arg_t は以下のような構造体である．

```
typedef struct {
  int * m;
} arg_t;
```

また，f3 における/tmp/zero_file は事前に作られ，中身はすべて 0 で埋まった 4 (sizeof(int)) バイトのファイルであるとする．

f0〜f3 それぞれの関数を，$P$ 個のスレッドもしくは $P$ 個のプロセスで並行に走らせる．例えば f0 を $P$ 個のスレッドで走らせる場合，

```
void run_threads() {
  arg_t args[P];
  int * m = (int *)calloc(1, sizeof(int));
  for (int i = 0; i < P; i++) {
    args[i].m = m;
  }
  pthread_t tids[P];
  for (int i = 0; i < P; i++) {
    pthread_create(&tids[i], 0, f0, &args[i]);
  }
  for (int i = 0; i < P; i++) {
    pthread_join(tids[i], 0);
  }
}
```

のようにする．f0 を $P$ 個のプロセスで走らせる場合，

```
void run_procs() {
  arg_t args[P];
  int * m = (int *)calloc(1, sizeof(int));
  for (int i = 0; i < P; i++) {
    args[i].m = m;
  }
  pid_t pids[P];
  for (int i = 0; i < P; i++) {
    pids[i] = fork();
    if (pids[i] == 0) {
      f0(&args[i]);
      exit(0);
    }
  }
  for (int i = 0; i < P; i++) {
    waitpid(pids[i], 0, 0);
  }
}
```

のようにする (2-6 行目は run_threads と run_procs で共通である)．

`f1`〜`f3` を走らせる場合は，それぞれの中の `f0` と書かれた部分 (`run_threads` の 9 行目および `run_procs` の 11 行目) を，適切な関数名に置き換えるだけである．これで都合 2 (`run_threads`，`run_procs`) × 4 (`f0`〜`f3`) = 8 通りのプログラムができたことになる．各スレッド (ないしプロセス) が，“`it's` $x$” という表示をするので，いずれのプログラムもそのような行を $P$ 個表示することになる．

以下では $N = 10000000, P = 2$ とする．以下の問いに答えよ．

(1) 8 通りのプログラムそれぞれについて，その出力としてあり得るものを，以下の中から**全て**選び，記号で答えよ．

(ア)

```
it's 8193611
it's 10000000
```

(イ)

```
it's 8193611
it's 10242445
```

(ウ)

```
it's 10000000
it's 10000000
```

(エ)

```
it's 10000000
it's 10242445
```

(オ)

```
it's 10193611
it's 10242445
```

(カ)

```
it's 19706152
it's 20000000
```

(キ)

```
it's 19706152
it's 20000001
```

(ク)

```
it's 20000000
it's 20000000
```

解答は，解答用紙の表で，あり得るケースにチェック (✓) を入れて答えること．

(例)

| | | (ア) | (イ) | (ウ) | (エ) | (オ) | (カ) | (キ) | (ク) |
|---|---|---|---|---|---|---|---|---|---|
| run_threads | f0 | ✓ | ✓ | ✓ | ✓ | | | | |
| run_threads | f1 | | ✓ | ✓ | | ✓ | | | ✓ |
| ⋮ | ⋮ | | | | | | | | |

(2) 例えば (オ) のような出力が生じうるケースについて，それがどのようにして生じるのかを説明せよ．

(3) pthread_mutex_t 型の大域変数 M を用意し，各繰り返しの中で 1 を足す操作の前後を pthread_mutex_lock(&M) と，pthread_mutex_unlock(&M) ではさむことにする．

具体的には，f0 であれば以下のように変更する．下線部 が変更した部分．

```
1  pthread_mutex_t M;
2  int x = 0;
3  void * f0(void * _) {
4    for (int i = 0; i < N; i++) {
5      pthread_mutex_lock(&M);
6      x = x + 1;
7      pthread_mutex_unlock(&M);
8    }
9    printf("it's %d\n", x);
10   return 0;
11 }
```

f1〜f3 についてもほぼ同じ変更を施す (唯一の違いは，pthread_mutex_lock(&M)，pthread_mutex_unlock(&M) の呼び出しではさむのが，*p = *p + 1; である点)．

このようにしてできたプログラム達の出力はどうなるか? (1) と同じ形式で答えよ．

(4) 多くのプロセッサに備わる compare-and-swap 命令について，その動作を説明せよ．

(5) compare-and-swap 命令を用いて，pthread_mutex_lock，pthread_mutex_unlock で達成したのと同じようなことを達成することができる．具体的にどうすればよいか述べよ．答え方としては，f0 の x = x + 1; または f1〜f3 の*p = *p + 1; の行を，どのように変更すればよいかを記せ (どちらを使っても良い)．

# 3

少年は指導教員から,「自分のノート PC のハードディスクの性能を測れ」と言われた．そこで少年は，自分の PC (これはやや旧式で，SSD ではなくハードディスクを搭載している) 上で 1GB のファイルを作り，それを読みだすのにかかる時間を測定した．

```
# large という，1GB (1024 * 1024 * 1024 バイト)のファイルを作る
$ dd if=/dev/zero of=large bs=$((1024 * 1024)) count=1024
1024+0 レコード入力
1024+0 レコード出力
1073741824 バイト (1.1 GB) コピーされました、3.03388 秒、354 MB/秒
$ time cat large > /dev/null  # cat で読み出し,時間を測る
real    0m0.149s
user    0m0.000s
sys 0m0.148s
```

少年はこの結果を持って,「1GB が 0.149 秒で読めたとあるから，ハードディスクの読み出し性能は 1GB/0.149 秒 ≈ 6.7GB/秒くらいです，ついでに言うと，書き込み性能は dd の結果から，354MB/秒くらいのようですね」と報告したところ，指導教員に (1) アホかと言われ てしまった．そして,「ハードディスクの性能が測りたいなら，あることに気をつけて測定しないとダメだ」と言われた．そこで (2) きちんと気をつけて測定 したところ，

```
$ time cat large > /dev/null
real    0m7.902s
user    0m0.007s
sys 0m0.212s
```

という結果が得られ，約 120MB/秒程度の性能となり，指導教員にはその数字ならあり得ると言われた．

次に,「1GB のファイルが 7.902 秒で読み出せることはわかった．じゃあ，1GB のファイル中の，指定された 1KB の部分 (だけ) を読めと言われたら，それにかかる時間はどのくらいか」と聞かれた．少年は調べてきますと言いつつ,「1GB が 7.902 秒だから 1KB にかかる時間は，実験などしなくても簡単に計算できる」と思い，

$$7.902\text{秒} \times 1\text{KB}/1\text{GB} \approx 7.902 \times 10^{-6} \approx 8\mu\text{秒}$$

さも実験をしたようなふりをしつつ「約 8 マイクロ秒でした」と答えたところ,「ウソつけ」と言われしまった．(3)「カタログに載っているこの数字を見れば，数ミリ秒より速いはずがないんだよ」とのことであった．

少年は，OS が行っているファイル読み書き性能が，いくつもの (4) ソフトウェア的な工夫によって達成されていることを学んだのであった．

以下の問いに答えよ．

(1) 下線部 (1) について，少年の測定はなぜ間違いなのか，説明せよ．6.7GB/秒という数字がハードディスクの性能でないとしたら，いったいなぜこんな数字が出たのか?

(2) 下線部 (2) について，正しく測定するための手順 (の例) を述べよ．

(3) 下線部 (3) について，カタログに載っているどのような数値を元に,「数ミリ秒より速いはずがない」と判断できるのか?

(4) 下線部 (4) について，仮に 1KB 読むのに数ミリ秒はかかるのだとしたら (仮に 1 ミリ秒かかるとする) これを比例させると，1GB 読むのには 1000 秒はかかる計算になる．それが実際には 7.9 秒で読み出せているのはどのような工夫を OS がしているからなのか?

**問題は以上である**

| 所属学科 | | 氏名 |
|---|---|---|

# 1

(1)

| | |
|---|---|
| A0 | |
| A1 | |
| A2 | |
| A3 | |
| A4 | |

(2)

| | |
|---|---|
| A0 | |
| A1 | |
| A2 | |
| A3 | |
| A4 | |

(3)

| | |
|---|---|
| (a) | |
| (b) | |

(4)

| | |
|---|---|
| A2 | |
| A3 | |

# 2

(1)

| | | | (ア) | (イ) | (ウ) | (エ) | (オ) | (カ) | (キ) | (ク) |
|---|---|---|---|---|---|---|---|---|---|---|
| run_threads | f0 | | | | | | | | | |
| run_threads | f1 | | | | | | | | | |
| run_threads | f2 | | | | | | | | | |
| run_threads | f3 | | | | | | | | | |
| run_procs | f0 | | | | | | | | | |
| run_procs | f1 | | | | | | | | | |
| run_procs | f2 | | | | | | | | | |
| run_procs | f3 | | | | | | | | | |

(2)

(3)

| | | | (ア) | (イ) | (ウ) | (エ) | (オ) | (カ) | (キ) | (ク) |
|---|---|---|---|---|---|---|---|---|---|---|
| run_threads | f0 | | | | | | | | | |
| run_threads | f1 | | | | | | | | | |
| run_threads | f2 | | | | | | | | | |
| run_threads | f3 | | | | | | | | | |
| run_procs | f0 | | | | | | | | | |
| run_procs | f1 | | | | | | | | | |
| run_procs | f2 | | | | | | | | | |
| run_procs | f3 | | | | | | | | | |

(4)

(5)

| | | |
|---|---|---|
| 3 | (1) | |
| | (2) | |
| | (3) | |
| | (4) | |