平成25年度オペレーティングシステム期末試験
Operating Systems: the Final Exam

2015年2月10日 (火)
February 10th 2015

- **問題は 3 問**

  There are three problems.

- **この冊子は，表紙 1 ページ (このページ)，日本語の問題 2-7 ページ，その英訳が 8-13 ページから成る．**

  This booklet consists of a cover page (this page), problems in Japanese (2-7 pages), and their English translations (8-13 pages).

- 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．**

  There is an answer sheet. **It is printed both sides.**

- 各問題の解答は所定の解答欄に書くこと．

  Write your answers to each problem in the designated box in the answer sheet.

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, 不思議だなぁ.

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. 今日大学の授業で，自分の PC で立ち上がっているプロセスやスレッドを調べる方法を習ったんです. [(a)] ていうコマンドラインでできるって. そしたらなんとプロセスが 195 個も立ち上がっていたんです!

O: ふ〜ん，そうかもね. でも，何が不思議なの?

M: はい, こんなに周りにプロセスが立ち上がっていたら，普段私が授業や演習で作っているプログラムって，本来のコンピュータの速度の，1/196 の速度でしか動いていないんじゃないかと思って. で，余分なプロセスがほとんど走っていないサーバにログインして同じプログラムを走らせたんですけど，ほとんど変わらないんです.

O: そりゃそうだよ，195 個プロセス—てことは少なくとも 195 個のスレッド—がいると言っても，そのほとんどは，中断中という状態になっていて，CPU は割り当てられないのよ.

M: あ，そういえば授業でもそんなことを言っていたような. でも OS はどうやって，中断中のスレッドとそうでないスレッドを区別するんでしょうか?

O: [(b)]

M: なるほど〜. で，実行可能なスレッドが複数いる場合は，当然それらの間で CPU を分け合うんですよね.

O: そうね.

M: 実行可能なスレッドの中には，私が演習で作ったみたいに，ひたすら計算だけをやってシステムコールなんて一切呼ばないものも有りますね．システムコールを呼べば OS がそのタイミングで他のスレッドに切り替えるとかできそうですけど，そうでないスレッドの場合，どうやってそのスレッドから CPU を奪って，他のスレッドに切り替えるのでしょうか?

O: (c)

M: なるほどですね〜．OS は，実行可能なスレッドを代わりばんこに実行しているんでしょうか?

O: ん〜，ま，大雑把にはそう思っていてもいいけどもう少し賢いことをしているんだな．まず,「代わりばんこ」っていうのは，もう少しちゃんとアルゴリズムとして定式化するとどうなるかしら?

M: そうですね，まず,「実行可能なスレッド」のキュー (run_queue) というのがあって,

(i) 実行中スレッドが他のスレッドに CPU を奪われる時は，もともと実行中だったスレッドを run_queue の末尾に入れ，run_queue の先頭にあるスレッドを次に実行する．

(ii) 実行中のスレッドが中断するときは，run_queue から外され，run_queue の先頭にあるスレッドを次に実行する．

(iii) 中断中のスレッドが復帰する (実行可能になる) ときは run_queue の末尾に入る

そんなところでしょうか?

O: そうね, たしかにこれだと，常時実行可能なスレッドが複数あったら，きれいに代わりばんこに実行されて,「公平性」っていう意味ではいいよね．でも，これだけだと，特にデスクトップ OS なんかで重要な，ある目標が達成されないんだな．その目標は,「(d)」というものなの．

M: たしかにそうですね．あ，じゃあ，こうしたらどうでしょうか?

(i) さっきと同じ

(ii) さっきと同じ

(iii) 中断中の状態から復帰する (実行可能になる) ときは，(e)

O: それはいいかもしれないけど，でもそうすると今度は基本的な目標である「公平性」を損なうことになるんじゃない? 例えば，(f) こんなプログラム を書いたら，そのスレッドは，不当に多くの CPU 時間を得ることができてしまうよね．

実際の OS ではこの (g)「公平性」と(d)を両立 すべく，スケジューリングアルゴリズムが作られているのよ．

M: ふえ〜，たかが CPU を割り当てるだけなのに，結構頑張ってるんですねぇ．

(1) [(a)] には全てのプロセスを列挙する Linux のコマンドが入る．それを (必要ならばオプションも含めて) 書け．

(2) [(b)] にはどのような事象に対して OS がスレッドを，中断中にするかの説明が入る．そのような事象を 3 つ以上あげながら，適切な文章を書け．

(3) [(c)] には，長時間システムコールを発行せずに計算だけをするスレッドから，OS がどのように CPU を奪うかの説明が入る．適切な文章を書け．

(4) [(d)] には，OS がスケジューリングの目標のひとつとしている項目が入る．適切な言葉を書け．

(5) [(e)] には，みんとちゃんが当初示したアルゴリズムに対する単純な変更が入る．適切な文を書け．

(6) 下線部 (f) はどんなプログラムか．概要を書け．

(7) 下線部 (g) の，両方の目標を達成するようなスケジューリングアルゴリズムを一つ考え，その概要を示せ．

2

N バイトのファイルを配列に読み出し，その配列中のランダムな W バイトを読み出す様々なプログラムを考える．以下の (1)〜(5) の各プログラムについて，**物理メモリ**の使用量はどのくらいになるか，答えよ．いずれの場合も，OS カーネルおよび関わるプロセス全体での合計使用量を答えよ．ファイルと配列を読み出すのに必要な以外の物理メモリは考えなくて良い．

ページサイズを P とする．解答は，以下の項，その定数倍 ($2N$, $2W$ など)，およびそれらの和 ($N+2W$, $N+CN$ など) で表わせ．また，$CWP \ll N$ を仮定して良い．

$$N,\ W,\ CN,\ CW,\ NP,\ WP,\ CNP,\ CWP.$$

(1) malloc で N バイトを確保し，read を用いて読みだす．プログラム片は以下．

```
char * a = malloc(N);
int fd = open(filename, O_RDONLY);
read(fd, a, N);
random_access(a, N, W);
```

ただし，random_access(a, N, W) は，アドレス $[a, a+N)$ から乱数で W 個のアドレスを選び，そこをアクセスする，以下のような関数とする．

```
void random_access(char * a, long N, long W) {
  char s = 0;
  long i;
  for (i = 0; i < W; i++) {
    long k = 0以上N 未満の乱数;
    s += a[k];
  }
  printf("%d\n", s);
}
```

(2) mmap でファイルを N バイト分，マップする．プログラム片は以下．

```
int fd = open(filename, O_RDONLY);
char * a = mmap(0, N, PROT_READ, MAP_PRIVATE, fd, 0);
random_access(a, N, W);
```

以下では，C 個の子プロセスを起動し，子プロセスがファイル・配列をアクセスする．ただし，random_access 中の乱数はプロセスごとに異なる列を返す．

(3) 親プロセスが malloc で N バイトを確保し，各子プロセスが read を用いて読みだす．

```
char * a = malloc(N);
long i;
for (i = 0; i < C; i++) {
  if (fork() == 0) {
    int fd = open(filename, O_RDONLY);
```

```
    read(fd, a, N);
    random_access(a, N, W);
    exit(0);
  }
}
```

(4) 親プロセスが malloc, read を行い，子プロセスが配列をアクセスする．

```
char * a = malloc(N);
int fd = open(filename, O_RDONLY);
read(fd, a, N);
for (i = 0; i < C; i++) {
  if (fork() == 0) {
    random_access(a, N, W);
    exit(0);
  }
}
```

(5) 各子プロセスが mmap を行う．

```
for (i = 0; i < C; i++) {
  if (fork() == 0) {
    int fd = open(filename, O_RDONLY);
    char * a = mmap(0, N, PROT_READ, MAP_PRIVATE, fd, 0);
    random_access(a, N, W);
    exit(0);
  }
}
```

3

次のような動作をする二つの関数を作りたい.

- `track_modifications(void * a, long n);`
- `is_modified(void * p);`

`track_modifications`(a, n) を呼び出すと，$[a, a+n)$ の範囲にあるのアドレス p へ，その呼び出し以降書き込みが起きたかどうかを，`is_modified`(p) を呼び出すことで知ることができる．より正確には，

- a および n はページサイズの倍数とする.
- p は $a \leq p < a+n$ を満たす.
- 簡単のため `track_modifications` は一度だけ呼ばれるとする.
- これらの条件を満たす時，`is_modified`(p) は，`track_modifications`(a, n) の呼び出し以降，p を含むページへの書き込みが起きていれば 1, 起きていなければ 0 を返す.

(1) これらの関数を，OS カーネルで (つまり，必要とあらば OS を変更して) 実現する方法の概要を記せ.

(2) これらの関数を，Unix に既に備わっているシステムコールを用いて，ユーザレベルで実現する方法の概要を記せ.

(3) これらの関数の応用を一つ記せ.

平成 25 年度オペレーティングシステム期末試験 2015 年 2 月 10 日 実施 解答用紙

所属学科	学生証番号	氏名

1	(1)	
	(2)	
	(3)	
	(4)	
	(5)	
	(6)	
	(7)	

2	(1)	
	(2)	
	(3)	
	(4)	
	(5)	

3	(1)	
	(2)	
	(3)	