2021年度オペレーティングシステム期末試験

2021年2月1日 (火)

- **問題は 4 問**
- **この冊子は，表紙が 1 ページ (このページ), 問題が 2-8 ページからなる**
- 解答用紙の Submit は何度行っても良く, 最後に Submit したものだけが受け付けられるので, **試験終了や，その間際になるのを待たずに各自のタイミングで行うこと**
- 試験終了後に Submit するのは不正行為となる (Submit はできてしまうが, Submit した時刻が記録されており, 後からそれが発覚することになるので, 決して行わないよう注意せよ)

1

次の会話を聞いて以下の問いに答えよ.

長谷川 (以下 H): ども〜〜!! こ〜んに〜ちは〜!!

渡辺 (以下 W): 10:00 はおはようございますだろ!

H: OS の, 本を買ったよ!!

W: 読んだのかよ!

H: 俺ね, 今度 OS の試験を受けるんだよね!

W: ちゃんと勉強したのかよ!

H: 試験受けるからさ, ちょっとその前に練習させてよ

W: 練習, いいよ

H: 最初に execve システムコールを習ったよ! execve システムコールは(1) 指定したコマンドを実行するプロセスを生成するシステムコールなんだよね!!

W: へ〜そうなの

H: ところでプロセスってな〜に?

W: execve の前にそれ勉強しろよ!

H: あとさ, 機械語でプログラムを書けるようになろうと思うんだ

W: ほ〜, それまたなんで?

H: 機械語でプログラムを書けば自分の好きなアドレスをアクセスできるよね. だから, (2) 他のプロセスのデータを読むことが (アドレスさえわかれば) できちゃうんだよね!!

W: ほんとかよ, 危ねえな〜

H: ところでアドレスってな〜に?

W: 機械語の前にそれ勉強しろよ!

H: あとさ, コンピュータを立ち上げてすぐ後に, 同じファイルを 2 度続けて読むと, 多くの場合, 2 度目の方が圧倒的に速くなるんだよね. それは, (3) コンピュータが温まって, 高速動作に適した温度になるからなんだよね!!

W: 温まってないとダメなんて, コンピュータも人間みたいなところがあるんだなぁ〜

H: あとさ, こないだ僕のノートパソコンでプロセスの数を数えたら, 100 個くらいプロセスが存在してたんだよね. この状態でこのラップトップの最高性能の 1/2 くらいの性能で自分のプログラムを走らせるには, (4) 100 個くらいのスレッドを作る必要があるよね. だから 100 個スレッドを作ってやったよ!!

W: 無茶するねぇ〜

H: あとさ, 特権命令ってのがあって, (5) 管理者 (Unix であれば root ユーザ) しか実行できないんだよね. これで, 僕のデータが他の人に勝手に見られたりしないように, なってるんだってね.

W: そりゃ〜安心だねぇ

H: ところでさ,

W: 何だよ

H: (6) OSってな〜に?

W: それ知らねぇで言ってたのかよ, もーいーよ

H,W: ありがとうございました〜

問: (1)〜(5) の下線部がそれぞれ正しいか否かを述べ, 正しくなければ何が間違っているかの指摘, または正しい文 (の一部) への訂正をせよ. (6) については, OS とは何の略かを, カタカナまたは英語で述べよ.

2

以下の機能の実現方法を, 指定したハードウェアの機能がどのように使われているかに焦点をおいて説明せよ.

(1) 物理メモリを超える量のメモリを割り当てることができる. ハードウェアの機能: **ページテーブル**

(2) プロセスを生成 (複製) するシステムコールである, fork を高速に実行することができる. ハードウェアの機能: **ページテーブル**

(3) mmap システムコールで, 異なるプロセスの間でメモリを共有することができる. ハードウェアの機能: **ページテーブル**

(4) システムコールを使って OS の機能を呼び出せば, ハードディスクなど外部記憶装置からファイルを読み出すことができるが, システムコールを使わなければ読み出すことができない. ハードウェアの機能: **トラップ命令**

(5) 多数のスレッドに公平に CPU 時間を割り当てる. ハードウェアの機能: **割り込み**

3

Reader writer lock は, 通常の排他制御 (mutex) と似ているが, 読み出しロックと書き込みロックをサポートし, 前者を複数のスレッドが同時に取得できるようにした同期の機構である. 後者は mutex 同様, 高々1 つのスレッドしか同時に取得できず, 書き込みロックと読み出しロックも同時には取得できない. これをどう実現するかを考える.

実装する reader writer lock の構造体の名前を `rwlock_t` とする. l を `rwlock_t` へのポインタ (`rwlock_t *`型) としたとき, 構造体を初期化する関数

- `rwlock_init(`l`)`

に加え,

- `rwlock_rd_lock(`l`)` (読み出しロック取得),
- `rwlock_rd_unlock(`l`)` (読み出しロック解放),
- `rwlock_wr_lock(`l`)` (書き込みロック取得),
- `rwlock_wr_unlock(`l`)` (書き込みロック解放)

の 4 つの操作がある. 使用前にある 1 つのスレッドによって一度 `rwlock_init(`l`)` が呼ばれ, その後に 4 つの操作が任意個のスレッドによって任意回呼び出される. 各スレッドは, `rwlock_rd_lock(`l`)` または `rwlock_wr_lock(`l`)` を呼び出した後, 次は必ず `rwlock_rd_unlock(`l`)` または `rwlock_wr_unlock(`l`)` をそれぞれ呼び出すことを繰り返す.

それ以外の呼び方をすることはないとしてよい. 例えば, `rwlock_rd_lock(`l`)` を呼んだ後, `rwlock_rd_unlock(`l`)` を呼ばずにそれ以外の 3 つの関数を呼んだり, `rwlock_rd_lock(`l`)` を呼ばずに `rwlock_rd_unlock(`l`)` を呼び出すことはない.

あるスレッドが

- `rwlock_rd_lock(`l`)` から return してから `rwlock_rd_unlock(`l`)` をまだ呼んでいない状態が, 読み出しロックを取得している状態,
- `rwlock_wr_lock(`l`)` から return してから `rwlock_wr_unlock(`l`)` をまだ呼んでいない状態が, 書き込みロックを取得している状態

であり, 前者を満たすスレッド数を r, 後者を満たすスレッド数を w とすると, 常に以下が常に成り立つのが, reader writer lock の正しい動作ということになる.

- $0 \leq w \leq 1$
- $w = 1$ ならば $r = 0$

`rwlock_t` は以下の構造体に, 同期のために必要なフィールドを付け加えて作るとする.

```
1 typedef struct {
2   int readers;
3 } rwlock_t;
```

ただし, `readers` は, 32 bit の整数で, 書き込みロック取得中のスレッドがいる ($w = 1$) 場合は, `-1` であり, そうでない場合は読み出しロック取得中のスレッド数 (r) を保持するフィールドとする. 計算に参加するスレッドの数は, $2^{31} - 1$ 未満と仮定して良い (全部のスレッドが読み出しロックを取得しても, `readers == -1` となることはない).

(1) `rwlock_t` 構造体と 5 つの関数を, それぞれを以下のように実現した. どのような問題が起きるか述べよ.

```
typedef struct {
  int readers;
} rwlock_t;

void rwlock_init(rwlock_t * l) {
  l->readers = 0;
}

void rwlock_rd_lock(rwlock_t * l) {
  while (1) {
    if (l->readers != -1) {
      l->readers++;
      break;
    }
  }
}

void rwlock_rd_unlock(rwlock_t * l) {
  l->readers--;
}

void rwlock_wr_lock(rwlock_t * l) {
  while (1) {
    if (l->readers == 0) {
      l->readers = -1;
      break;
    }
  }
}

void rwlock_wr_unlock(rwlock_t * l) {
  l->readers = 0;
}

#include "rwlock_test.h"
```

(2) 前問の問題点を改善するつもりでそれぞれを以下のように実現した．どのような問題が起きるか述べよ．

```
typedef struct {
  int readers;
  pthread_mutex_t m[1];
} rwlock_t;

void rwlock_init(rwlock_t * l) {
  l->readers = 0;
  pthread_mutex_init(l->m, 0);
}

void rwlock_rd_lock(rwlock_t * l) {
  while (1) {
    pthread_mutex_lock(l->m);
    if (l->readers != -1) {
      l->readers++;
      break;
    }
    pthread_mutex_unlock(l->m);
  }
  pthread_mutex_unlock(l->m);
```

```
21 }
22
23 void rwlock_rd_unlock(rwlock_t * l) {
24   pthread_mutex_lock(l->m);
25   l->readers--;
26   pthread_mutex_unlock(l->m);
27 }
28
29 void rwlock_wr_lock(rwlock_t * l) {
30   while (1) {
31     pthread_mutex_lock(l->m);
32     if (l->readers == 0) {
33       l->readers = -1;
34       break;
35     }
36     pthread_mutex_unlock(l->m);
37   }
38   pthread_mutex_unlock(l->m);
39 }
40
41 void rwlock_wr_unlock(rwlock_t * l) {
42   pthread_mutex_lock(l->m);
43   l->readers = 0;
44   pthread_mutex_unlock(l->m);
45 }
46
47 #include "rwlock_test.h"
```

(3) `rwlock_t` 構造体と 5 つの関数を, Pthread の排他制御 (`pthread_mutex_t`), 条件変数 (`pthread_cond_t`) を使って実現せよ. 解答用のセルにあるコードを適宜変更し, コンパイル, テストの実行まで行うこと.

解答用紙には, 解答したコードを呼び出して実行するためのテスト用コードと, コンパイルしたプログラムをテストするコマンドが書いてあるので, それらを使ってテストせよ.

(4) compare-and-swap 命令とはどのような命令か説明せよ.

(5) GCC で compare-and-swap 命令を使うための関数 (intrinsics) の名前を答えよ.

(6) `rwlock_t` 構造体と 5 つの関数を, Pthread の排他制御 (`pthread_mutex_t`), 条件変数 (`pthread_cond_t`) を使わず, compare-and-swap 命令と futex を用いて実現せよ. 解答欄にあるコードを適宜変更し, コンパイル, テストの実行まで行うこと.

4

(1) コマンドラインでひとつ, ファイル名を受け取り, そのファイルのサイズ + 改行を, 標準出力に表示する C プログラム `file_size.c` を指定されたセルに書け. ファイルのサイズ, とはそのファイルに含まれるバイト数のこと.

```
$ cat summers
Kazuki
Ohtake
Masakazu
Mimura
$ gcc -o file_size file_size.c
$ ./file_size summers # summers は改行文字含め 30 バイト
30
$
```

のような動作をする.

引数が足りなければ以下のようなエラーメッセージを標準エラーに出力して終了すること (以下でも同様).

```
$ ./file_size
error: missing an argument
$
```

(2) コマンドラインで第一引数にファイル名 (F とする), 第二引数に数字 (i とする) を受け取り, F の i バイト目を含む行を標準出力に表示する C プログラム `print_line_of_ith_char.c` を書け. ファイルの先頭のバイトは「0 バイト目」と数える. F が存在しなかったり, i が F のサイズ以上だったときは, それぞれ以下にあるようなエラーメッセージを, 標準エラー出力に表示せよ.

ここでファイル中の「行」とは, ファイル中で以下を満たす部分バイト列のことである.

- 先頭は, ファイルの先頭バイト (0 バイト目) であるか, 改行文字の直後のバイトである.
- 末尾は, 改行文字であるか, ファイルの最後のバイトである.
- 末尾以外には改行文字を含まない.

なお, F は巨大かもしれず, コンピュータの物理メモリ量を上回っているかもしれない. その状況でも少ない物理メモリ (一定量 + 実際に表示する行と同程度の物理メモリしか消費しない) で, 効率的に動作するようにすること.

```
$ gcc -o print_line_of_ith_char print_line_of_ith_char.c
$ ./print_line_of_ith_char summers 8
Ohtake
$ ./print_line_of_ith_char summers 13
Ohtake
$ ./print_line_of_ith_char summers 14
Masakazu
$ ./print_line_of_ith_char summers 30
error: the index >= file size
$
```