2023年度オペレーティングシステム期末試験

2024年1月24日 (水)

- **問題は 3 問**
- **この冊子は，表紙が 1 ページ (このページ), 問題が 2-7 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

以下の問いに答えよ. ただし, 問い自体に誤った言明が含まれている場合もあり, その場合は以下の例に習ってそれを正した上で問いに答えよ.

例 1: 豊臣秀吉が長篠の戦いにどのようにして勝利したか, その方法を 30 字程度で述べよ.

(解答例)

問い自体が誤り. 豊臣秀吉は長篠の戦いを戦っておらず, 長篠の戦いに勝利したのは織田信長である.

方法: 鉄砲の音によって武田軍の騎馬隊を大混乱に陥れた. (24 字)

例 2: ドラえもんが青い理由を 50 字程度で述べよ.

(解答例)

ネズミに耳をかじられ手術を受け, 手術後耳がなくなった自分の姿を見て, 文字通り青ざめてしまったのです... (50 字)

括弧内の字数は参考として書いたもので解答に含める必要はない.

(1) マルチプログラム (複数のプログラムが動く) 環境では, 一般に mutex よりもスピンロックのほうが望ましいとされるのはなぜか, その理由を簡潔に 50-100 字程度で述べよ.

(2) アドレス空間はプロセス間で分離しており, 通常はメモリを共有することはできないが, 共有する方法もある. どのようなプログラムを書けばそれができるか, 具体的なシステムコールをあげつつ, 方法を 50-100 字程度で説明せよ (詳細なコードを示す必要はない).

(3) P 個の仮想コアを持つ計算機で, P^2 個以上のスレッドがシステムコールを呼び出すことなく無限に計算を続けると, 他のスレッドに実行の機会が回ってくることがなくなるのはなぜか? 50-100 字程度で述べよ.

(4) キャッシュに収まりきらない大きなファイルを 2 分探索で検索するプログラムがある. このようなプログラムを多数同時に走らせてもスループット (単位時間あたりに処理できる検索の数) が向上しない理由はなにか? 50-100 字程度で述べよ.

(5) プログラムの起動を高速にするのに, ページテーブルが果たしている役割を 2 つ以上述べよ. 100-200 字程度でまとめよ.

2

非負の整数を保持し, 以下のような操作を許すデータ構造「カウンタ」(counter_t) を考える. 以下で v は非負の整数である.

データ型定義: 保持する値を構造体の中に持つ.

```
typedef struct {
  long v;
} counter_t;
```

初期化: カウンタ c が保持する値を v に初期化する.

```
void counter_init(counter_t * c, long v) {
  c->v = v;
}
```

(下限付きの) 減算: カウンタ c が保持する値が x 以上であればそれを x 減らして 1 を返し, x 未満であれば減らさずに 0 を返す.

```
int counter_dec(counter_t * c, long x) {
  if (c->v < x) {
    return 0;
  } else {
    c->v -= x;
    return 1;
  }
}
```

さて上記の実装は, 複数のスレッドが並行して counter_dec を呼んだときは正しく動作しない. つまり, スレッドセーフではない.

(1) mutex を用いて, counter_dec がスレッドセーフになるように, counter_t, counter_init, counter_dec に必要な変更を施せ. ただし counter_init は counter_dec に先立って一度だけ呼ばれ, counter_dec と並行して呼ばれることはないものとしてよい.

(2) mutex やスピンロックを用いず, compare-and-swap 命令を用いて同様のことを行え. C 言語で compare-and-swap 命令は, __sync_bool_compare_and_swap(long *p, long old, long new) という関数で利用できる.

以下では, たくさんの種類の商品の在庫数を管理することを考える. M 種類の商品があり, 商品の種類を $0, 1, \cdots, M-1$ の整数で表すことにする. 在庫数を, M 要素の counter_t の配列 C で管理する. 商品 p の在庫数を管理するカウンタが C[p] である.

買い物客がやってきて買い物をする. 簡単のため, 一回の買い物ではちょうど 2 種類の商品を求める (買おうとする) とする. 各商品を求める個数についての制限はない.

一回の買い物では, 求める 2 種類の商品の在庫数が, ともに求める以上にあればそれらを買い, 対応する商品の在庫数を買った分だけ減少させる. どちらかでも在庫数が足りなければ, **どちらも一個も**

買わずにその買い物すべてを諦める. 片方の商品だけ買ったり, 在庫にある数だけ買うようなことはしない.

例えば $M = 5$ だったとして, 商品の在庫が商品 0 から順に

$$46, 35, 24, 57, 13$$

だったとして, ある買い物が「商品 1 を 30 個, 商品 3 を 50 個」を求めているとすると, (他の買い物がなければ) その買い物は成立し, 買い物後の在庫の状態は

$$46, 5, 24, 7, 13$$

となる. これがもし, 「商品 0 を 30 個, 商品 4 を 50 個」だったらその買い物は成立せず, 在庫は変化しない.

一回の買い物の行動を C 言語で書けば以下の関数 buy2 の様になる.

```
int buy2(counter_t * C, long M, long p, long q, long x, long y) {
  if (C[p].v < x || C[q].v < y) {
    return 0;
  } else {
    C[p].v -= x;
    C[q].v -= y;
    return 1;
  }
}
```

関数呼び出し buy2(C, M, p, q, x, y) は, M 種類の商品の中から商品 p を x 個, 商品 q を y 個求め, 買い物が成立したら C にそれを反映して 1 を返す. 成立しなければ C を変更せずに 0 を返す.

$0 \leq p < M$, $0 \leq q < M$, $p \neq q$ を仮定して良い.

buy2 をスレッドセーフにする, つまり, buy2 を複数のスレッドが並行して呼び出しても正しく動作するようにするために, 以下のアプローチを考える.

方法 A: counter_dec を, 商品 p, q それぞれについて呼び出して実現する. ただし以下の counter_dec は (1) で作ったものとする.

```
int buy2(counter_t * C, long M, long p, long q, long x, long y) {
  if (!counter_dec(&C[p], x)) return 0;
  if (!counter_dec(&C[q], y)) {
    counter_dec(&C[p], -x);
    return 0;
  }
  return 1;
}
```

方法 B: 商品 p, 商品 q の mutex をこの順番でロックしてから, その状態で在庫確認し, 在庫が十分にあれば商品 p, 商品 q の在庫数を減らす. 疑似コードで書けば以下.

```
int buy2(counter_t * C, long M, long p, long q, long x, long y) {
  C[p]をロック;
  C[q]をロック;
  if (C[p].v < x || C[q].v < y) {
    C[p]をアンロック;
    C[q]をアンロック;
    return 0;
  } else {
    C[p].v -= x;
    C[q].v -= y;
    C[p]をアンロック;
    C[q]をアンロック;
    return 1;
  }
}
```

方法 C: 商品 0 の mutex をロックしてから, その状態で在庫確認し, 在庫が十分にあれば商品 p, 商品 q の在庫数を減らす. 疑似コードで書けば以下.

```
int buy2(counter_t * C, long M, long p, long q, long x, long y) {
  C[0]をロック;
  if (C[p].v < x || C[q].v < y) {
    C[0]をアンロック;
    return 0;
  } else {
    C[p].v -= x;
    C[q].v -= y;
    C[0]をアンロック;
    return 1;
  }
}
```

さてここで buy2 関数を複数のスレッドが並行して呼び出しても**正しく動作する**, ということをきちんと定義しておこう. 商品 p ($p = 0, 1, \cdots, M$) の在庫の初期値を C_p とする. 複数のスレッドが buy2 を様々なパラメータで合計 T 回呼び出したとする. それらの呼び出しを $B_0, B_1, \cdots, B_{T-1}$ と呼ぶことにし, 呼び出し B_t が buy2 のパラメータ p, q, x, y に渡した値をそれぞれ, p_t, q_t, x_t, y_t と呼び, B_t の返り値 (0 または 1) を s_t と呼ぶことにする (s は, success の頭文字). これらの呼び出しが**正しく動作する**とは, 以下の条件が満たされることを言う.

条件 1: すべての呼び出し B_t ($t = 0, 1, \cdots, T-1$) が終了する.

条件 2: 各商品の在庫の値が, 買われた分だけ正しく減っている. つまり, すべての呼び出しが終了したあとの, 商品 p に対するカウンタの値 (C[p].v) を F_p として, すべての商品 $p = 0, 1, \cdots, M$ に対し,

$$F_p \geq 0$$

および,

$$C_p - F_p = \sum_{\{t \in [0,M) | s_t = 1, p_t = p\}} x_t + \sum_{\{t \in [0,M) | s_t = 1, q_t = p\}} y_t$$

が成り立つ.

条件 3: 買い物が無駄に失敗していない. 言い換えると, 成立しなかった買い物は, すべての呼び出しが終了した状態でもやはり成立しない. つまり, すべての $t = 0, 1, \cdots, M-1$ に対し,

$$s_t = 0 \to (F_{p_t} < x_t \text{ または } F_{q_t} < y_t)$$

が成り立っている.

この条件が必要な理由は一見わかりにくいかもしれないが, この条件を課さなければ, すべての買い物を無条件で失敗させるような解 (何もせずに 0 を返す `buy2`) も, 「正しい」ということになってしまうことに注意.

(3) 方法 A が正しいか答えよ. 条件 1〜3 を満たしているか, 各条件について答えよ. ある条件を満たしていないと答える場合, 条件が満たされない実行系列を具体的に示すこと. 満たしていると答える場合は, そのことの証明は不要.

(4) 方法 B について同様の問いに答えよ.

(5) 方法 C について同様の問いに答えよ.

(6) 3 条件を満たす, (方法 A, B, C とは異なる) 方法を答えよ. 50-100 字程度の概略と, 実際に動くコードを書け. 書いたコードは, Jupyter 環境上で解答用紙 (ipynb ファイル) を fetch したときにともに配布されているプログラムを使ってテストするとよい. テストの仕方は解答用紙にある. その際に必要な, 復号のためのパスワードは

`Eew9Ee`

である. テストは解答に対する自信を深めるための方法であり, 必須ではない.

3

以下の問いに答えよ.

(1) 未来のアクセス系列が与えられている (オフラインアルゴリズム) という仮定のもとで, 最適なページ置換アルゴリズムとはどのようなものか, 70 字以内で簡潔に答えよ

(2) P ページ分の物理メモリを持つ計算機で L 個の論理ページ ($0, 1, \cdots, L-1$ の番号をつける) をアクセスするプログラムを考える. ただし $L > P$ であり, 初期状態ではどの論理ページも物理メモリ上にないものとする. プログラムのアクセスパターンと, ページ置換アルゴリズムが以下の組み合わせであるとき, **平均ページフォルト間隔**を, 理由の説明とともにそれぞれ答えよ.

ここで, 平均ページフォルト間隔とは, プログラムを十分長い時間走らせたとき, あるページフォルトから次のページフォルトまでの間隔の平均値である. 例えば 1000 番目のアクセスでページフォルトが起き, 次のページフォルトが 1234 番目で起きたとすると両者の間隔は 234 であると考える. 例えば, ページフォルトがすべてのアクセスで起きているとき, ページフォルト間隔は 1 であり, ページフォルトを起こすアクセスと起こさないアクセスが交互に起きているのであれば, ページフォルト間隔は 2 となる.

(a) プログラムは, 論理ページ 0 から $L-1$ までを一回ずつこの順にアクセスすることを繰り返す (つまりアクセス系列は $0, 1, \cdots, L-1, 0, 1, \cdots, L-1, \cdots$). ページ置換アルゴリズムは LRU.

(b) プログラムは (a) と同じ. ページ置換アルゴリズムは (1) で述べた最適なオフラインアルゴリズム.

(c) プログラムは (a) と同じ. ページ置換アルゴリズムは乱択アルゴリズムで, ページ置換が必要になったとき, 物理メモリ上にある P 個のページから等確率 $(1/P)$ でページアウトするページを選ぶ. ただし簡単のためここでは, $L = P + 1$ とする.

(d) プログラムはすべてのページを等確率 $(1/L)$ でランダムにアクセスすることを繰り返す. ページ置換アルゴリズムは LRU.

(3) ページ置換アルゴリズムの目標はページ置換数を減らすことだが, 実際の OS はこれ以外の実践的な要素も考慮してページアウトするページを決めている. どのような要素を考慮しているか, 理由とともに 150 字程度述べよ.

問題は以上である.