システムソフトウェア・試験問題

2018 年度 (2018 年 11 月 26 日・試験時間 90 分)

書籍，配布資料およびノート等は参照してはならない．ただし，最大一枚までのメモ（手書きに限る．A4 両面使用可）を参照できるものとする．

1. 仮想記憶機構に関する以下の問 **(a)〜(c)** に答えよ

(a) ページアウトが必要になった時点で最も最初にメモリに読み込まれていたページを犠牲ページとするページ置換アルゴリズム（FIFO アルゴリズム）を考える．5 個のページ 0〜4 が以下 (1) に示す順でアクセスされるとする．

$$0, 1, 2, 3, 0, 1, 4, 4, 4, 2, 3, 3 \qquad (1)$$

物理ページフレーム数が 3 で全て空である（割り当てがない）状態から上記のアクセスを実行したとき，ページフォルトは何回発生するか．

(b) 物理ページフレーム数が 4 で全て空である（割り当てがない）状態から上記 (1) に示すアクセスを実行したとき，ページフォルトは何回発生するか．

(c) 最近もっとも使われていないページを犠牲ページとするアルゴリズム（LRU アルゴリズム）をページ置換アルゴリズムとして用いることが難しい理由を述べよ．

2. xv6 のファイルシステムに関する以下の問 **(a)〜(c)** に答えよ．

(a) ファイルシステムのビットマップブロックには，ブロックの使用・未使用を表すビット列が格納されている．このビット列が実際の使用・未使用状況と異なる場合に何が起こるか．

(b) dinode 構造体のフィールド nlink は当該構造体が表すファイルがディレクトリから参照されている数を表している．xv6 の起動直後にルートディレクトリにおいて以下のようなコマンドを実行した（$はシェルのプロンプトである）．この時点における，ファイル foo/baz，ディレクトリ bar，およびルートディレクトリを表す dinode 構造体の nlink の値をそれぞれ記せ．

```
$ mkdir foo
$ mkdir bar
$ echo Hello > foo/baz
$ ln foo/baz bar/boo
```

(c) プロセス P がディレクトリ D にあるファイル F に書き込みを行なっている最中にシステムがクラッシュした．xv6 のログ機構によって実現されるものを以下の選択肢 1–5 から 1 つまたは 2 つ選べ．

1. P が F に書き込もうとしていたメモリ中のデータを復帰できる
2. P の実行を開始する前の F の内容を復帰できる
3. F が使用しているデータブロック（間接ブロックを含む）がビットマップブロック内で使用済みと正しくマークされている
4. P の実行を中断した時点から再開できる
5. P の実行を開始する前の D の内容を復帰できる

3. ソースコード 1 は xv6 におけるスピンロックの実装[1]である．関数 xchg は第 1 引数（uint 型へのポインタ）が指す先の変数に第 2 引数の値を代入して第 1 引数に元々入っていた値を返す．ただしこれらの動作は

[1] エラー処理やデバッグ情報の管理，およびアウトオブオーダー実行への対応を省いて簡略化している．

```
1  struct spinlock { uint locked; };
2
3  void acquire(struct spinlock *lk) {
4    pushcli();
5    while (xchg(&lk->locked, 1) != 0);
6  }
7
8  void release(struct spinlock *lk) {
9    lk->locked = 0;
10   popcli();
11 }
```

ソースコード 1: 関数 aqcuire と release

x86 の xchg 命令を用いてアトミックに行なわれる．関数 pushcli を実行した CPU コアは割り込み禁止状態になる．その CPU コアが pushcli の実行回数と同じ回数 popcli を実行することで割り込み禁止は解除される．以下の問 **(a)** および **(b)** に答えよ．

(a) 関数 xchg が（xchg 命令を用いずに）以下のように定義されていたとする．このとき，CPU コア数が 1 の場合と 1 より大きい場合のそれぞれについて，acquire と release で正しく相互排除は行えるか否かを述べよ．

```
uint xchg(uint *var, uint newval) {
  uint oldval = *var;
  *var = newval;
  return oldval;
}
```

(b) ソースコード関数 1 の acquire および release の定義から pushcli および popcli を削除したとする．このとき，CPU コア数が 1 の場合と 1 より大きい場合のそれぞれについて，起こり得る現象（ヒント：1 単語）を述べよ．

4. ソースコード 2 は xv6 におけるスリープロックの実装[2]である．関数 acquiresleep と releasesleep は構造体 sleeplock で表されるロックの確保と解放を行う．acquire と release と異なり，ロックの確保に関して待っているプロセス（を管理しているカーネルスレッド）は SLEEP 状態にあり，CPU 時間を消費しない．実装にもちいられている関数 sleep は適当な資源

[2]エラー処理やデバッグ情報の管理を省いて簡略化している．

```
1  struct sleeplock {
2    uint locked;
3    struct spinlock lk;
4  };
5
6  void acquiresleep(struct sleeplock *lk) {
7    acquire(&lk->lk);
8    while (lk->locked)
9      sleep(lk, &lk->lk);
10   lk->locked = 1;
11   release(&lk->lk);
12 }
13
14 void releasesleep(struct sleeplock *lk) {
15   acquire(&lk->lk);
16   lk->locked = 0;
17   wakeup(lk);
18   release(&lk->lk);
19 }
```

ソースコード 2: 関数 acquiresleep と releasesleep

を指し示すポインタ（チャネルと呼ばれる）とそれに付随するスピンロックを引数とし，呼び出したカーネルスレッドの実行を SLEEP 状態にして中断する．この状態を当該チャネルについてスリープ状態にあると呼ぶ．関数 wakeup は引数であるチャネルについてスリープ状態にあるカーネルスレッド全ての実行を再開する．以下の問 **(a)** および **(b)** に答えよ．

(a) xv6 では，バッファキャッシュへのブロックの読み書きや inode 構造体のディスクからの読み書きに関数 acquiresleep と releasesleep を用いている．acquire と release によるスピンロックでなくこれらを用いる理由を述べよ．

(b) 関数 acquiresleep の定義（8 行目）において，**if**でなく **while**を用いている理由を述べよ．
ヒント：wakeup は同じチャネルについてスリープ状態にある全てのカーネルスレッドを RUNNABLE にする．