オペレーティングシステム・期末試験の解答例と解説

2011 年度 E・O クラス (2012 年 2 月 13 日・試験時間 90 分)

1. (a) 空欄 A, B, C に入る式はそれぞれ以下の通り.

A : `true`
B : `xchg(&in_use, &r)`
C : `r`

解説 xchg を用いることで変数 in_use への代入とこの変数からの（代入前の）値の読み出しをアトミックに行うことができる．これにより，プログラム 1 では講義資料 3 の 46 ページで説明した test_and_set によるものと同等の相互排除を実現している．

このプログラムでは xchg の呼び出し（プログラム 1 の 9 行目）とループの条件判定（同 10 行目）の間に他のスレッドが実行される可能性があるが，相互排除は問題なく行われる．これはループの条件判定に用いられる変数 r がスレッド毎の局所変数であり，その値は他のスレッドの実行の影響を受けないためである．

プログラム 1 の動作を Promela でモデル化したものをプログラム 2 に示す．プロセス Q の定義は P と同じである．Spin を用いて正しく相互排除が実現されていることを確認できる．

参考 XV6 における xchg の定義はおおむね以下に相当する（引数と返値の型は bool ではなく int であるが）.

```
atomic bool xchg(bool *m, bool new) {
    bool old = *m;
    *m = new;
    return old;
}
```

この定義を用いた場合，プログラム 1 の 7〜10 行目は以下のようにシンプルなものになる．

```
    while (xchg(&in_use, true));
```

(b) `false`

```
 1  bool in_use = false;
 2  int ncs = 0;
 3
 4  inline xchg(m, r) {
 5    bool tmp;
 6    atomic {
 7        tmp = r; r = m; m = tmp;
 8    }
 9  }
10
11  active proctype P() {
12    bool r;
13    do :: /* NC */
14          r = true;
15          xchg(in_use, r);
16          do :: r -> xchg(in_use, r)
17             :: else -> break
18          od;
19          ncs++;
20          assert(ncs == 1); /* CS */
21          ncs--;
22          in_use = false
23    od
24  }
25
26  active proctype Q() { ... }
```

プログラム 2: xchg を用いた相互排除のモデル

(c) (2)

(d) test-and-set 命令，compare-and-swap (CAS) 命令など

(e) 以下のいずれかであればよい．

解答 **1** 必要．ロックに必要な変数の読み出しと書き込みの間に他のプロセス（スレッド）がタイマー割り込みによって running になる可能性があるため．

解答 **2** 不要．ロックに必要な変数の読み出しから書き込みの間を割り込み禁止にすればよいため．

解説 実装によって必要か不要かが決まるので，理由が正しく書かれていればどちらでもよい．

(f) スリープロックを実現するためのサービスはそもそもカーネルが提供するものであり，そのカーネル自体では使うことができない．ここでカーネル内ロックをスリープロックとして実現したとする．そのためには，カーネルスレッドのモードを（run から wait に，あるいは wait から ready に）変更するメカニズムが必要であるが，一般にカーネル内で資源をロックする時間は短いことが多く，そのためのオーバーヘッドが無視できなくなる．

2. (a) ページ 0

解説 時刻 1460 における各ページの参照ビットと汚れビット，クラスは以下のようになる．

p	0	1	2	3
R_p	0	1	0	1
D_p	0	1	1	0
クラス	0	3	1	2

クラスが一番小さいページは 0 なので，これが犠牲ページになる．

(b) 必要ない．ページ 0 はロードされてから一度も書き込みが行われていないため．これは汚れビットが 0 であることで判断できる．

(c) LRU アルゴリズムを実装するためには，メモリアクセス毎に該当するページのアクセス時刻を記録する必要がある．これは（特殊なハードウェアなしで）小さいオーバーヘッドで実現するのは困難である．

3. (a) 71680

解説 直接参照 12 ブロックに加え，間接参照ブロックから $512 \div 4 = 128$ ブロック参照できる．よって合計 $(12 + 128) \times 512 = 71680$ バイトまでのファイルを作ることができる．

(b) 8

解説 4000 以上で最も小さい 512 の倍数は 4096 である．$4096 = 512 \times 8$ なので 8 ブロック必要であるが，これらは全て直接参照可能である．よって間接ブロックは不要で，計 8 ブロックとなる．

(c) 15

解説 $7168 = 512 \times 14$ なので 14 ブロック必要であるが，直接参照できるのは 12 ブロックまでなので，間接参照ブロックが必要である．よって間接ブロックを入れて 15 ブロック必要である．