オペレーティングシステム・期末試験の解答例と解説

2012 年度 E・O クラス (2013 年 2 月 15 日・試験時間 90 分)

1. (a) 2 つ以上のスレッド（プロセス）がクリティカルセクションに同時に入らないこと．

(b) 安全性をみたす．

解説 in_use は共有変数，r はスレッド毎の局所変数とする．1 個以上のスレッドで

```
in_use = false;
r = true;
xchg(&in_use, &r);
```

を実行したき，in_use が **true** となる状態では r の値が **false** となるスレッドはただ 1 つである．これは xchg がアトミックに実行されることから容易に示すことができる．

プログラム 1 において，各スレッドが 8 行目の xchg(&in_use, &r); を実行する直前の状態では r の値は **true** である．したがって，do-while ループを抜けて *CS* にたどり着くスレッドが 2 個以上あると，上で述べたことに反することになる．

(c) (3) スピンロック

解説 カーネルの実装方式の中には，単一のロックを用意しておき，スレッドがカーネルに入る際（例えばシステムコールの際）と出る際にそのロックの獲得と解放を行うものがある．このような，カーネル全体にかける単一の大域的なロックをジャイアントロックと呼ぶ．ジャイアントロックにより，任意の時点でカーネル内を実行しているスレッドは高々一つであることが保証できるため，カーネル内部での資源管理の実装が容易になるが，マルチプロセッサシステムでは性能低下につながる．

他の選択肢は並行システムとは何の関係もない．アームロックは格闘技における関節技の名前で，ポストロックは音楽のジャンル[1]の一つである．

(d) カーネル内スレッドでのロック，あるいは短い期間のロックなどで用いられる．

解説 スリープロックを行うためにはスレッド（プロセス）を waiting 状態にする必要がある．そのためのオーバーヘッドは無視できない．スピンロックは CPU を無駄に消費しているとも考えられるが，ロックされる期間が短い場合は相対的なオーバーヘッドは小さくなる．

また，スリープロックを行う際にはスレッド（プロセス）スケジューリングの機構が必要であるが，カーネルスレッドではその機構を用いることができない場合がある．そのような場合はスピンロックを用いることになる．

(e) A: r, B: in_use

(f) 改良版では，in_use のチェックを通常の（バスのロックを伴わない）読み出しで行い，その値が **true** から **false** に変化したときのみ xchg での変更を試みている．これにより，バスのロック回数を大幅に減らすことができ，性能向上に寄与する．

[1] 今更と思うが Godspeed You! Black Emperor は面白いと思う．

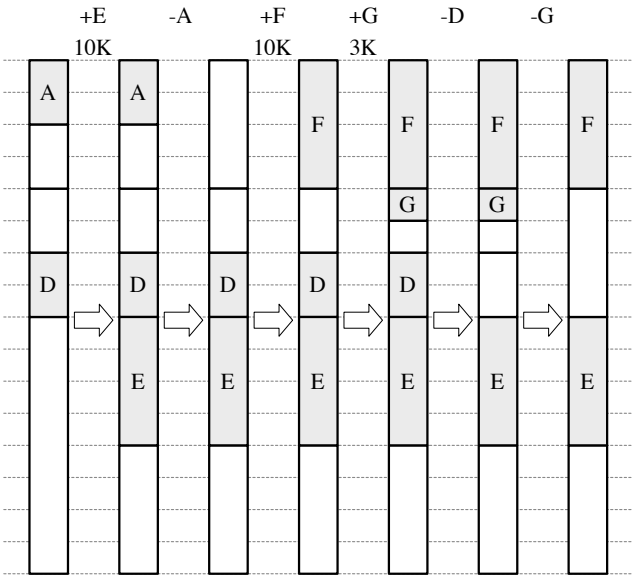

図 1: 問題 2(a) の解答

2. (a) 図 1 の通り．

解説　A を解放した後，A に割り当てられていた 8KB の部分と隣接する 8KB の部分（A のバディ）との併合が行われることに注意．次の F は E の下ではなく一番上に割り当てられる．同様に，G が解放された後に併合が 2 回行われることにも注意すること．

(b) (B) 内部断片化

解説　バディシステムで割当られるメモリの大きさは最小割当単位 $\times 2^k$（$k \geq 0$）なので，必要なメモリの大きさがこの値でない限り，割り当てたメモリの中に余分な部分が生じる．これは内部断片化である．

3. (a) 71680 バイト（70K バイト）

解説　直接参照 12 ブロックに加え，間接参照ブロックから $512 \div 4 = 128$ ブロックが参照できる．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．よって間接参照ブロックは不要で計 8 ブロックとなる．

(c) 17

解説　8000 以上で最も小さい 512 の倍数は 8192 である．$8192 = 512 \times 16$ なので 16 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．よって間接参照ブロックを入れて 17 ブロックとなる．

(d) 無駄が生じる．

解説　実際に必要となる i-node ブロックの数は $\lceil$`ninodes / IPB`$\rceil$ である．しかし C の整数除算では `ninodes / IPB` は $\lfloor$`ninodes / IPB`$\rfloor$ になるため 1 を足している．ここで `ninodes` が `IPB` で割り切れるときは $\lceil$`ninodes / IPB`$\rceil$ = $\lfloor$`ninodes / IPB`$\rfloor$ なので 1 ブロックが無駄になる．

`dinode` 構造体の大きさが 64 バイトなので `IPB` の値は $512 \div 64 = 8$ である．したがって `ninodes` が 200 のときは上に述べたように 1 ブロックの無駄が生じる．

この無駄があっても動作には問題ないが，あまり気持ちのいいものではない．割り切れるか否かで場合分けするのも面倒だが，どうすればよいか．一般に x, y が整数で $x \geq 1, y > 0$ であれば $\lfloor (x-1)/y \rfloor + 1 = \lceil x/y \rceil$ なので，i-node ブロック数を `(ninodes - 1) / IPB + 1` で計算すればよい．