システムソフトウェア・試験問題

2019 年度 (2019 年 11 月 25 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. RISC-V 版 xv6 のファイルシステムにおいて，inode ブロックに格納される dinode 構造体は以下のように定義されている．

```
struct dinode {
  short type;              // ファイルタイプ
  short major;             // 主デバイス番号
  short minor;             // 副デバイス番号
  short nlink;             // リンク数
  uint size;               // ファイルサイズ
  uint addrs[NDIRECT+1];   // ブロック参照
};
```

マクロ NDIRECT は 12 と定義されている．addrs[0] から addrs[NDIRECT-1] の 12 個がデータブロックへの直接参照で，addrs[NDIRECT] が間接参照である．ブロック番号を表す uint 型は 4 バイト（32 ビット）である．またブロックサイズは 1024 バイトである．

(a) RISC-V 版 xv6 では最大何バイトまでの大きさのファイルを作ることができるか．ディスクは十分大きく，ディスクサイズによる制約はないものとする．

(b) 20000 バイトのファイルが占めるデータブロックの数はいくつか．間接参照ブロックが必要な場合はそれも数えること．i-node，ビットマップ，ログのためのブロックは数えなくてもよい．

(c) dinode 構造体のフィールド nlink の値は，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す．ここで RISC-V 版 xv6 において以下のようなコマンドを実行したとする（$はシェルのプロンプトである）．このときの，ファイル foo/hello.txt およびディレクトリ foo を表す dinode 構造体の nlink の値をそれぞれ記せ．

```
$ mkdir foo
$ echo Hello > foo/hello.txt
$ mkdir foo/bar
$ ln foo/hello.txt foo/bar/hello.txt
```

(d) nlink の値が正しい値より小さい場合と大きい場合に起こりうる不具合をそれぞれ一つ挙げよ．

2. 図 1 は RISC-V 版 xv6 におけるユーザプロセスのメモリ空間である．テキスト・データ領域とスタック領域の間にガードページと呼ばれる領域がある．

(a) ガードページの役割について説明せよ．

(b) 上記 (a) を可能にする機能の名前として最も適切なものを以下の (1)〜(6) から一つ選べ．

(1)TLB (2) ページフォルト (3) ページ置換
(4) カナリア (5) トランポリン (6) セグメント

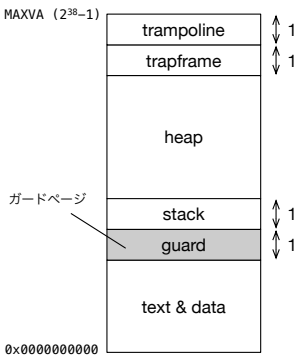

図 1: xv6 のユーザプロセスのメモリ空間

(次頁に続く)

3. RISC-V 版 xv6 のカーネルを構成する関数を用いてセマフォを実装した（図 2）．構造体 spinlock および関数 acquire, release は xv6 におけるスピンロックの実装である．関数 sleep（図 4）は適当なデータ（c とする）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状態を SLEEPING にしてその実行を中断する．この状態を c についてスリープ状態にあると呼ぶ．関数 wakeup は引数についてスリープ状態にあるカーネルスレッド全てを実行可能（RUNNABLE 状態）にする．

定義したセマフォの使用例を図 3 に示す．ここで N, K は正の整数であり，$N \geq K$ とする．K はセマフォのカウンタの初期値である．関数 initsem を 1 回実行してセマフォ sem を初期化したのち，関数 loop を実行する N 個のカーネルスレッドを起動した．

(a) 図 2 の関数 P の 7〜9 行目を以下のように変更したときに生じうる不具合を一つ述べよ．

```
7     while (s->count == 0)
8         sleep(s, &s->lock);
9     acquire(&s->lock);
```

(b) 任意の時点において CS の部分を実行できるスレッドは高々何個か．

(c) 関数 initsem を実行したのち，P(&sem); および V(&sem); の実行が完了した回数をそれぞれ N_P および N_V とする（ただし P(&sem); を実行したスレッドが図 2 の 9 行目で呼び出した sleep によってスリープ状態にある場合は，その P(&sem); の実行は完了していないものとする）．以下の等式が常に成り立つよう A に入る式（N_P と N_V の式）を答えよ．

$$\texttt{sem.count} = K + \boxed{\text{A}}$$

(d) CS の部分を実行しているスレッドの数を N_{CS} とする．以下の等式が常に成り立つよう B に入る式（N_P と N_V の式）を答えよ．

$$N_{CS} = \boxed{\text{B}}$$

(e)* 上記 **(c)**,**(d)** の結果を用いて **(b)** の結果の理由を説明せよ．

```
1  struct semaphore {
2      struct spinlock lock;
3      int count;
4  };
5
6  void P(struct semaphore *s) {
7      acquire(&s->lock);
8      while (s->count == 0)
9          sleep(s, &s->lock);
10     s->count -= 1;
11     release(&s->lock);
12 }
13
14 void V(struct semaphore *s) {
15     acquire(&s->lock);
16     s->count += 1;
17     wakeup(s);
18     release(&s->lock);
19 }
```

図 2: セマフォの定義

```
struct semaphore sem;

void initsem() {
    initlock(&sem.lock, "semaphore");
    sem.count = K;
}

void loop() {
    for (;;) {
        NC
        P(&sem);
        CS
        V(&sem);
    }
}
```

図 3: セマフォの使用例

```
void sleep(void *chan, struct spinlock *lk) {
    struct proc *p = myproc();
    if (lk != &p->lock) {
        acquire(&p->lock);
        release(lk);
    }
    p->chan = chan;
    /* SLEEPINGにしてスケジューラに制御を移す */
    p->state = SLEEPING;
    sched();
    /* スケジューラによる実行再開でここに戻る */
    p->chan = 0;
    if (lk != &p->lock) {
        release(&p->lock);
        acquire(lk);
    }
}
```

図 4: sleep の定義