システムソフトウェア・試験問題

2022 年度 (2022 年 12 月 1 日・試験時間 100 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 のカーネルを構成する関数を用いてセマフォを実装した（コード 1）．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 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（コード 3）．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の状態を RUNNABLE にする（コード 3）．

```
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 }
```

コード 1: セマフォの定義

こうして定義したセマフォを用いて，生産者・消費者問題の解を実装した（コード 2）．関数 setup を実行した後，関数 producer および consumer をそれぞれ別のカーネルスレッド（以下スレッド）で実行するものとする．ここでは生産者（producer を実行しているスレッド）と消費者（consumer を実行しているスレッド）の間でデータを受け渡す代わりに共有変数 nitems の値を増減させている．また N は正の整数とする．

(a) コード 2 の空欄 [A], [B], [C], [D] に入る式をそれぞれ答えよ．

(b) スピンロック mtx は何のためにあるのか．その役割を説明せよ．

(c) 生産者のスレッドが 3 個，消費者のスレッドが 2 個あり，N は 5 より大きい整数であるとする．以下の (A)〜(H) のうち不変条件であるものを一つ選べ．

(A) `0 <= nitems && nitems <= 1`

(B) `0 <= nitems && nitems <= 3`

(C) `2 <= nitems && nitems <= 5`

(D) `0 <= nitems && nitems <= N`

(E) `3 <= nitems && nitems <= N + 2`

(F) `0 <= nitems && nitems <= N - 3`

(G) `2 <= nitems && nitems <= N - 3`

(H) `N - 2 <= nitems && nitems <= N + 3`

(d) コード 1 の 17 行目で wakeup(s) を実行したときの s->countの値は 0 より大きいため，8〜9 行目を以下のように変更してもよさそうに思える．

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

このような変更を行ったときに発生し得る不具合を一つ挙げ，不具合となる理由を説明せよ．

```
struct semaphore ne, nf;
struct spinlock mtx;
int nitems = 0;

void setup() {
    initlock(&ne.lock, "ne");
    ne.count = 0;
    initlock(&nf.lock, "nf");
    nf.count = N;            // Nは正の整数
    initlock(&mtx, "mtx");
}

void producer() {
    for (;;) {
        wait_random_period    // 任意の時間待つ
        P( A );
        acquire(&mtx);
        nitems++;
        release(&mtx);
        V( B );
    }
}

void consumer() {
    for (;;) {
        P( C );
        acquire(&mtx);
        nitems--;
        release(&mtx);
        V( D );
        wait_random_period    // 任意の時間待つ
    }
}
```

コード 2: セマフォによる生産者・消費者問題の解

2. コード 4 に示すプログラム foo を xv6 で実行した．実行前にはディレクトリ D は存在していなかったとする．

(a) プログラム foo を実行するとファイル D/F および D/G が作られる（以下単に F および G とする）．エラーがなく実行できたとして，以下の (A)〜(H) から正しい記述をすべて選べ．

(A) F のサイズは 1024 バイト，G のサイズは 2048 バイトである．

(B) F と G のサイズはどちらも 1024 バイトである．

(C) F と G のサイズはどちらも 2048 バイトである．

(D) F と G の内容は 1 バイト目から異なる．

```
void sleep(void *chan, struct spinlock *lk) {
    struct proc *p = myproc();
    acquire(&p->lock);
    release(lk);
    p->chan = chan;
    p->state = SLEEPING;
    sched();
    p->chan = 0;
    release(&p->lock);
    acquire(lk);
}

void wakeup(void *chan) {
    struct proc *p;
    for (p = proc; p < &proc[NPROC]; p++) {
        if (p != myproc()) {
            acquire(&p->lock);
            if (p->state == SLEEPING &&
                p->chan == chan) {
                p->state = RUNNABLE;
            }
            release(&p->lock);
        }
    }
}
```

コード 3: sleep と wakeup の定義

(E) F と G の内容は 1 バイト目から 1024 バイト目までは一致する．

(F) F と G の内容はすべて一致する．

(G) F と G の inode 番号は異なる．

(H) F と G の inode 番号は一致する．

(b) プログラム foo の実行中にシステムがクラッシュした．再起動時に起こり得る状況として正しいものを以下の (A)〜(H) からすべて選べ．ただし xv6 のファイルシステムの 1 ブロックの大きさは 1024 バイトであり，バッファキャッシュ1 個の大きさも同じである．

(A) F も G も存在しない．

(B) F は存在するが G は存在しない．

(C) F は存在しないが G は存在する．

(D) F と G は両方存在し，いずれもサイズが 0 バイトである．

```c
char a[2048], b[1024];

int main() {
    int fd;
    for (int i = 0; i < 2048; i++)
        a[i] = 'a' + i % 26;
    for (int i = 0; i < 1024; i++)
        a[i] = 'A' + i % 26;
    mkdir("D");
    chdir("D");
    fd = open("F", O_WRONLY | O_CREATE);
    write(fd, a, 2048);
    close(fd);
    link("F", "G");
    fd = open("G", O_WRONLY);
    write(fd, b, 1024);
    close(fd);
    exit(0);
}
```

コード 4: プログラム foo

(E) F と G は両方存在し，いずれもサイズが 512 バイトである．

(F) F と G は両方存在し，いずれもサイズが 1024 バイトである．

(G) F と G は両方存在し，いずれもサイズが 2048 バイトである．

(H) F と G は両方存在し，片方のサイズが 0 バイトでもう片方のサイズが 512 バイトである．

(I) F と G は両方存在し，片方のサイズが 0 バイトでもう片方のサイズが 1024 バイトである．

(J) F と G は両方存在し，片方のサイズが 0 バイトでもう片方のサイズが 2048 バイトである．

```c
uint64 sys_freemem(void) {
    uint n = 0;
    acquire(&kmem.lock);
    for (struct run *r = kmem.freelist; r != 0;
         r = r->next)
        n++;
    release(&kmem.lock);
    return n * PGSIZE;
}
```

コード 5: システムコール freemem の実装

```c
int main() {
    printf("%l\n", freemem());
    sbrk(4096);
    printf("%l\n", freemem());
    sbrk(1024);
    printf("%l\n", freemem());
    sbrk(1024);
    printf("%l\n", freemem());
    sbrk(4096);
    printf("%l\n", freemem());
    sbrk(2048);
    printf("%l\n", freemem());
    exit(0);
}
```

コード 6: freemem の利用例

3. xv6 では，カーネル内で実行時にメモリを確保するために関数 kalloc が用意されている．関数 kalloc で確保可能な空きメモリのバイト数を与えるシステムコール freemem をコード 5 のように実装した．

(a) 1 回の kalloc の呼び出しで確保されるメモリのバイト数を答えよ．

(b) システムコール freemem を使ったプログラム例をコード 6 に示す．実行したところ，2 行目の printf の実行で 133382144 が表示された．4, 6, 8, 10, 12 行目の printf で表示される値をそれぞれ答えよ．