进程

20373980 林子杰

首先说明，本次引用的PV操作指的是记录型信号量，代表了`wait()`和`signal()`操作。

Q1.读写者问题（写者优先）

使用伪代码描述解决之一问题的方法。为了保证**读写互斥**、**写写互斥**，首先需要两个互斥变量。同时，为了保证写者的优先级，以及保证写者在正在进行读操作的所有读者都完成任务后才进行读，还要设置两个共享的计数变量，并且为他们分别分配互斥变量。

```
/* 互斥变量 */
  readLock = 1
  writeLock = 1
  readCountLock = 1
  writeCountLock = 1
/* 计数变量 */
  readCount = 0
  writeCount = 0

writer() {

  /* 对writeCountLock的互斥操作 */
  P(writeCountLock)
    /* 当前没有其它写者，则本写者可以阻塞读操作 */
    if(writeCountLock == 0)
      P(readLock)
    writeCount++
  V(writeCountLock)

  /* 互斥的写操作 */
  p(writeLock)
    write()
  V(writeLock)

  /* 对writeCountLock的互斥操作 */
  P(writeCountLock)
  /* 当前写者是最后一个写者，则释放读操作锁。上锁者和释放者未必是同一个人 */
    writeCount--
    if(writeCountLock == 0)
      V(readLock)
  V(writeCountLock)
}

reader() {
```

```
  /* 首先判断锁是否可用。拿取锁保证读写互斥；立即释放可以保证共享读 */
  P(readLock)
  v(readLock)

  /* 对readCountLock的互斥操作 */
  P(readCountLock)
    if(readCount == 0)
      P(writeLock)
    readCount++
  V(readCountLock)

  /* 共享的读操作 */
  read()

  /* 对readCountLock的互斥操作 */
  P(readCountLock)
    /* 一组read操作执行完后，才释放写者锁允许写行为 */
    readCount--
    if(readCount == 0)
      V(writeLock)
  V(readCountLock)
}
```

- 写写互斥是通过`writeLock`的PV操作嵌套`write()`实现的
- 读读共享是因为`read()`没有用PV操作嵌套
- 读写互斥是通过`writeCountLock`和`readCountLock`的PV操作实现的（某一方进入共享区则抢占对方的锁，直至某一方的最后一个人离开共享区）
- 写优先的根本原因是在`writer()`中，只要存在写者，就先抢对方的锁，即不让读者进入

"写者优先"指的是**仅在当前共享区内存在写者时，后面等待区的写者可以先进入共享区**，而不是指写者可以随意打断读者；共享区内存在读者，或是空时，写者和读者是公平竞争的。

Q2.寿司店问题

同样分析共享变量。我们需要一个`eating`记录当前用餐的人数，还需要一个`waiting`记录当前排队的人数。对于这两个变量，我们分别需要一个互斥变量。

```
/* 互斥变量 */
  eatingLock = 1
  waitingLock = 0
/* 计数变量 */
  eating = 0
  waiting = 0
  wait = false

client() {

  /* 拿取eating的锁 */
  P(eatingLock)
```

```
    /* 当前有五个人，需要等待。释放eating锁并拿取waiting锁进入等待 */
    if(wait)
      waiting++
      V(eatingLock)
      P(waitingLock)
    else
    /* 当前不足五个人，可以落座。修改eating和落座人数，释放eating锁 */
      eating++
      if (eating == 5)
        wait = true
      else
        wait = false
        V(eatingLock)

  /* 本题eat()未要求互斥 */
  eat()

  /* 用餐完毕，再次拿取eating的锁。每次要修改eating都要拿锁 */
  P(eatingLock)
    eating--
    /* 同一桌人走完了，队列中的至多五个人可以eat */
    if(eating == 0)
      int cnt = MIN(waiting,5)
      waiting -= cnt
      eating += cnt
      /* 下一桌又坐满了，后面的人只能等待 */
      if(eating == 5)
        wait = true
      else
        wait = false
      /* 按照本桌可容纳人数，逐个唤醒乘客 */
      while(n--)
        V(waitingLock)
  V(eatingLock)
}
```

Q3.缓冲区

这是一个比较典型的生产者-消费者模型。需要注意，我们需要分别统计缓冲区奇数、偶数和整数的数量。本次实现的思路和面向对象作业中的一些思路很相似。

```
/* 同步变量 */
  shareLock = 0
/* 互斥变量 */
  intLock = 1
  oddLock = 1
  evenLock = 1
/* 计数变量 */
  intNum = 0
```

```
  oddNum = 0
  evenNum = 0

P1() {
  while(true) {
    int p = produce()
    if(intNum < n)
      P(intLock)
        intNum++
        put(n)
      V(intLock)
      if(isOdd(p))
        P(oddLock)
          oddNum++
        V(oddLock)
      else if(isEven(p))
        P(evenLock)
          evenNum++
        V(evenLock)
      V(shareLock)
      V(shareLock)
    else if(intNum >= n)
      P(shareLock)
  }
}

P2() {
  while(true) {
    if(oddNum > 0) {
      P(intLock)
        intNum--
        getOdd()
      V(intLock)
      V(shareLock)
      P(oddLock)
        oddNum--
      V(oddLock)
      countOdd()
    } else if(oddNum == 0){
      P(shareLock)
    }
  }
}

P3() {
  while(true) {
    if(evenNum > 0) {
      P(intLock)
        intNum--
```

```
        getEvem()
      V(intLock)
      V(shareLock)
      P(evemLock)
        evenNum--
      V(evenLock)
      countEven()
    } else if(evenNum == 0){
      P(shareLock)
    }
  }
}
```

Q4.搜索-插入-删除

首先明确存在互斥关系的线程：插入和插入、删除和搜索、删除和插入、删除和删除。由此可知，我们需要四个互斥变量来描述问题。然后，需要考虑的是删除线程时的计数问题。

```
/* 互斥变量 */
  insert_insert_lock = 1
  delete_search_lock = 1
  delete_insert_lock = 1
  delete_delete_lock = 1
  searcher_num_lock = 1
/* 计数变量 */
  searcher_num = 0
searcher() {
  while(true) {
    P(search_num_lock)
        searcher_num++
        P(search_delate)
    V(search_num_lock)
    P(delete_search_lock)
      search()
    V(delete_search_lock)
    P(search_num_locks)
        searcher_num--
    if (searcher == 0)
        V(search_delate);
    V(search_num_lock);
  }
}

inserter() {
  while(true) {
    P(insert_insert_lock)
    P(delete_insert_lock)
      insert()
    V(detele_insert_lock)
```

```
    V(insert_insert_lock)
  }
}

deleter() {
  while(true) {
    P(delete_delete_lock)
    P(delete_insert_lock)
    P(delete_search_lock)
      insert()
    V(delete_search_lock)
    V(detele_insert_lock)
    V(delete_delete_lock)
  }
}
```