寻道总时间：316ms 。

五、进程同步与互斥1

```
Semaphore empty = 50; // 仓库空位数
Semaphore itemA = 0, itemB = 0; // A 产品和 B 产品的数量
Semaphore mutex = 1; // 控制对仓库的互斥访问

main() {
    cobegin
        ProducerA();
        ProducerB();
        for (int i = 0; i < 5; i++) {
            ConsumerA();
            ConsumerB();
        }
    coend
}

ProducerA() {
    while (true) {
        P(empty);
        ProductA a = produceA();
        P(mutex);
        put(a);
        V(mutex);
        V(itemA);
    }
}

ProducerB() {
    while (true) {
        P(empty);
        ProductB b = produceB();
        P(mutex);
        put(b);
        V(mutex);
        V(itemB);
    }
}

ConsumerA() {
    while (true) {
        P(itemA);
        P(mutex);
        ProductA a = takeA();
        V(mutex);
        consume(a);
        V(empty);
    }
}

ConsumerB() {
    while (true) {
        P(itemA);
        P(mutex);
        ProductB b = takeB();
```

```
        V(mutex);
        consume(b);
        V(empty);
    }
}
```

六、进程同步与互斥2

Version 1 摘自《The Little Book of Semaphore》

部分变量名有改动。

```
int passengers = 0;
Semaphore mutex = 1;
Semaphore multiplex = 50;
Semaphore bus = 0;
Semaphore allAboard = 0;

main() {
    cobegin
        Bus();
        Passenger();
    coend
}

Bus() {
    P(mutex);
    if (passengers > 0) {
        V(bus); // 通知第一名乘客上车
        P(allAboard);
    }
    V(mutex);
    depart();
}

Passenger() {
    P(multiplex);
        P(mutex);
            passengers++;
        V(mutex);
        P(bus);
    V(multiplex);

    boardBus();

    passengers--;
    if (passengers == 0) {
        V(allAboard);
    } else {
        V(bus); // 接力棒，上车的乘客唤醒下一个乘客上车
    }
}
```

Version 2 另一方法摘自 《The Little Book of Semaphore》

部分变量名有改动。

```
int waiting = 0;
Semaphore mutex = 1;
Semaphore bus = 0;
Semaphore boarded = 0;

main() {
    cobegin
        Bus();
        Passenger();
    coend
}

Bus() {
    P(mutex);
    n = min(waiting, 50);
    for (int i = 0; i < n; i++) {
        V(bus);
        P(boarded);
    }

    waiting = max(waiting - 50, 0);
    V(mutex);
    depart();
}

Passenger() {
    P(mutex);
    waiting++;
    V(mutex);

    P(bus);
    boardBus();
    V(boarded);
}
```

Version 3 自己写的

```
const int MAXN = 50;
int waitCount = 0; // 当前排队等待的人数
Semaphore mutexWaitCount = 1; // 对 waitCount 的互斥访问
Semaphore mutex = 1; // 对上车的互斥访问
Semaphore wait = 0; // 乘客排队阻塞，等待 bus 唤醒
Semaphore board = 0; // 乘客告知司机已上车

main() {
    cobegin
        Passenger();
        Bus();
    coend
}

Passenger() {
    // 1. 先开始排队
    P(mutexWaitCount);
    waitCount++;
    V(mutexWaitCount);

```

```
    // 2. 等待被 Bus 唤醒
    P(wait);
    // 3. 开始上车
    P(mutexWaitCount);
    waitCount--;
    V(mutexWaitCount);
    P(mutex);
    boardBus();
    V(mutex);
    V(board);
}

Bus() {
    // 1. 如果无人等待则出发
    P(mutexWaitCount);
    if (waitCount == 0) {
        V(mutexWaitCount);
        depart();
    } else {
        // 2. 确定上车的人数, 依次唤醒
        int n = waitCount <= MAXN ? waitCount : MAXN;
        V(mutexWaitCount);
        for (int i = 0; i < n; i++) {
            V(wait);
        }
        // 3. 等待当前的全部乘客上车
        for (int i = 0; i < n; i++) {
            P(board);
        }
        // 4. 发车
        depart();
    }
}
```

Version 4 来自网络

```
#define N 50                //一辆车满载50人
Semaphore mutex = mutex2 = 1;// 用于保护waiting变量
Semaphore board = 1;// 用于保护上车
Semaphore queue = 0;// 用于排队等待一辆车,初始队伍0人
int waiting = 0;        // 表示等待即将到来的车人数,初始等车的人为0
int should_go_this_time=0;
// 乘客进程
void Passenger(){
  P(mutex); // 保护waiting变量
    waiting++;// 到来的时候必须要等待
  V(mutex);
  P(queue); // 占用一个排队名额, queue是负数时, |queue|=排队人数
  // 此时被下面的bus唤醒, 接触排队阻塞, 继续
    // 上面解决了这个人应该等待

  P(mutex2);
  boardBus();   // 巴士到达, 开始上车
  waiting--;        // 等待的人减少
  should_go_this_time--;
  V(mutex2);
  // 若上车动作较慢, 还不能走
```

```
  if(should_go_this_time == 0)
    V(ready); // 告诉司机可以走了
}

// 巴士进程
void Bus(){
  if(waiting == 0)
    depart();
  else{
    // 确定了n之后，新来的同学也不能上车了
    P(board);   //开始保护上车进程
    n = min(N, waiting);    // 最多上50个
    should_go_this_time = n;
    for(int i = 0; i<n; i++){ //开始上车
      V(queue);  // 唤醒排队的n个人继续进程
    }
    V(board);

    P(ready); // 等待所有人上完车
    depart();   // 如果waiting==0, n==0, 直接就走了
  }
}
```

七、死锁

1

初始状态

```
Process     Allocated       Need
            R1 R2 R3 R4     R1 R2 R3 R4
P1          0  1  2  0      0  0  0  0  --> OK
P2          1  0  0  0      0  8  5  0
P3          1  3  5  4      1  0  0  2
P4          0  2  3  2      0  0  2  0  --> OK
P5          0  0  1  4      0  6  4  2
Free                        0  3  2  0
```

P1 不再需要资源，可正常结束，结束后变为

```
Process     Allocated       Need
            R1 R2 R3 R4     R1 R2 R3 R4
P2          1  0  0  0      0  8  5  0
P3          1  3  5  4      1  0  0  2
P4          0  2  3  2      0  0  2  0  --> OK
P5          0  0  1  4      0  6  4  2
Free                        0  4  4  0
```

此时可用资源满足 P4 需求，故将资源分配给 P4，而后 P4 正常结束，系统状态变为