

寻道总时间：316ms。

五、进程同步与互斥1

```
1 Semaphore empty = 50; // 仓库空位数
2 Semaphore itemA = 0, itemB = 0; // A 产品和 B 产品的数量
3 Semaphore mutex = 1; // 控制对仓库的互斥访问
4
5 main() {
6     cobegin
7         ProducerA();
8         ProducerB();
9         for (int i = 0; i < 5; i++) {
10             ConsumerA();
11             ConsumerB();
12         }
13     coend
14 }
15
16 ProducerA() {
17     while (true) {
18         P(empty);
19         ProductA a = produceA();
20         P(mutex);
21         put(a);
22         V(mutex);
23         V(itemA);
24     }
25 }
26
27 ProducerB() {
28     while (true) {
29         P(empty);
30         ProductB b = produceB();
31         P(mutex);
32         put(b);
33         V(mutex);
34         V(itemB);
35     }
36 }
37
38 ConsumerA() {
39     while (true) {
40         P(itemA);
41         P(mutex);
42         ProductA a = takeA();
43         V(mutex);
44         consume(a);
45         V(empty);
46     }
47 }
48
49 ConsumerB() {
50     while (true) {
51         P(itemA);
52         P(mutex);
53         ProductB b = takeB();
```

```
54     V(mutex);
55     consume(b);
56     V(empty);
57 }
58 }
```

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。

```
1  int passengers = 0;
2  Semaphore mutex = 1;
3  Semaphore multiplex = 50;
4  Semaphore bus = 0;
5  Semaphore allAboard = 0;
6
7  main() {
8      cobegin
9          Bus();
10         Passenger();
11     coend
12 }
13
14 Bus() {
15     P(mutex);
16     if (passengers > 0) {
17         V(bus); // 通知第一名乘客上车
18         P(allAboard);
19     }
20     V(mutex);
21     depart();
22 }
23
24 Passenger() {
25     P(multiplex);
26     P(mutex);
27     passengers++;
28     V(mutex);
29     P(bus);
30     V(multiplex);
31
32     boardBus();
33
34     passengers--;
35     if (passengers == 0) {
36         V(allAboard);
37     } else {
38         V(bus); // 接力棒, 上车的乘客唤醒下一个乘客上车
39     }
40 }
```

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

部分变量名有改动。

```

1  int waiting = 0;
2  Semaphore mutex = 1;
3  Semaphore bus = 0;
4  Semaphore boarded = 0;
5
6  main() {
7      cobegin
8          Bus();
9          Passenger();
10     coend
11 }
12
13 Bus() {
14     P(mutex);
15     n = min(waiting, 50);
16     for (int i = 0; i < n; i++) {
17         V(bus);
18         P(boarded);
19     }
20
21     waiting = max(waiting - 50, 0);
22     V(mutex);
23     depart();
24 }
25
26 Passenger() {
27     P(mutex);
28     waiting++;
29     V(mutex);
30
31     P(bus);
32     boardBus();
33     V(boarded);
34 }

```

Version 3 自己写的

```

1  const int MAXN = 50;
2  int waitCount = 0; // 当前排队等待的人数
3  Semaphore mutexWaitCount = 1; // 对 waitCount 的互斥访问
4  Semaphore mutex = 1; // 对上车的互斥访问
5  Semaphore wait = 0; // 乘客排队阻塞, 等待 bus 唤醒
6  Semaphore board = 0; // 乘客告知司机已上车
7
8  main() {
9      cobegin
10         Passenger();
11         Bus();
12     coend
13 }
14
15 Passenger() {
16     // 1. 先开始排队
17     P(mutexWaitCount);
18     waitCount++;
19     V(mutexWaitCount);
20

```

```

21 // 2. 等待被 Bus 唤醒
22 P(wait);
23 // 3. 开始上车
24 P(mutexWaitCount);
25 waitCount--;
26 V(mutexWaitCount);
27 P(mutex);
28 boardBus();
29 V(mutex);
30 V(board);
31 }
32
33 Bus() {
34 // 1. 如果无人等待则出发
35 P(mutexWaitCount);
36 if (waitCount == 0) {
37     V(mutexWaitCount);
38     depart();
39 } else {
40     // 2. 确定上车的人数, 依次唤醒
41     int n = waitCount <= MAXN ? waitCount : MAXN;
42     V(mutexWaitCount);
43     for (int i = 0; i < n; i++) {
44         V(wait);
45     }
46     // 3. 等待当前的全部乘客上车
47     for (int i = 0; i < n; i++) {
48         P(board);
49     }
50     // 4. 发车
51     depart();
52 }
53 }

```

Version 4 [来自网络](#)

```

1 #define N 50 //一辆车满载50人
2 Semaphore mutex = mutex2 = 1; // 用于保护waiting变量
3 Semaphore board = 1; // 用于保护上车
4 Semaphore queue = 0; // 用于排队等待一辆车, 初始队伍0人
5 int waiting = 0; // 表示等待即将到来的车人数, 初始等车的人为0
6 int should_go_this_time=0;
7 // 乘客进程
8 void Passenger(){
9     P(mutex); // 保护waiting变量
10    waiting++; // 到来的时候必须要等待
11    V(mutex);
12    P(queue); // 占用一个排队名额, queue是负数时, |queue|=排队人数
13    // 此时被下面的bus唤醒, 接触排队阻塞, 继续
14    // 上面解决了这个人应该等待
15
16    P(mutex2);
17    boardBus(); // 巴士到达, 开始上车
18    waiting--; // 等待的人减少
19    should_go_this_time--;
20    V(mutex2);
21    // 若上车动作较慢, 还不能走

```

```

22     if(should_go_this_time == 0)
23         V(ready); // 告诉司机可以走了
24     }
25
26     // 巴士进程
27     void Bus(){
28         if(waiting == 0)
29             depart();
30         else{
31             // 确定了n之后, 新来的同学也不能上车了
32             P(board); //开始保护上车进程
33             n = min(N, waiting); // 最多上50个
34             should_go_this_time = n;
35             for(int i = 0; i<n; i++){ //开始上车
36                 V(queue); // 唤醒排队的n个人继续进程
37             }
38             V(board);
39
40             P(ready); // 等待所有人上完车
41             depart(); // 如果waiting==0, n==0, 直接就走了
42         }
43     }

```

七、死锁

1

初始状态

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

P1 不再需要资源, 可正常结束, 结束后变为

1	Process	Allocated				Need				
2		R1	R2	R3	R4	R1	R2	R3	R4	
3	P2	1	0	0	0	0	8	5	0	
4	P3	1	3	5	4	1	0	0	2	
5	P4	0	2	3	2	0	0	2	0	--> OK
6	P5	0	0	1	4	0	6	4	2	
7	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