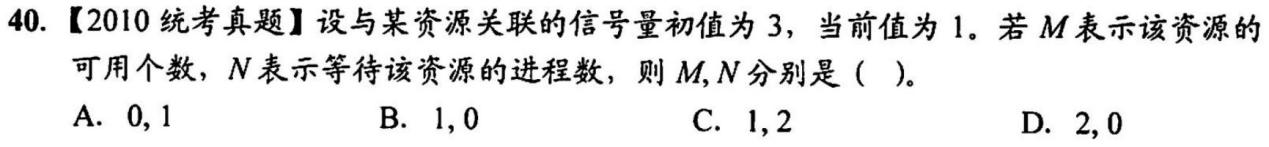
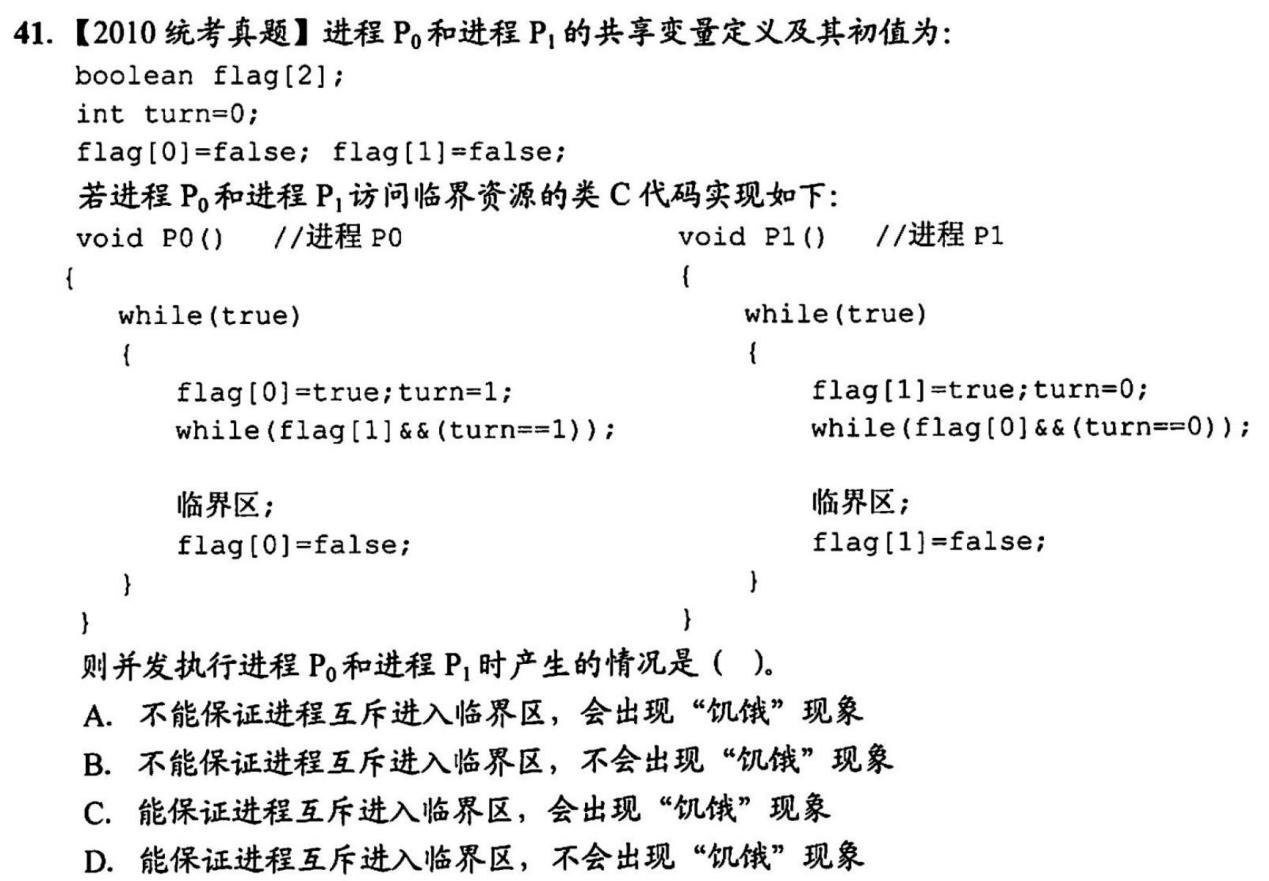
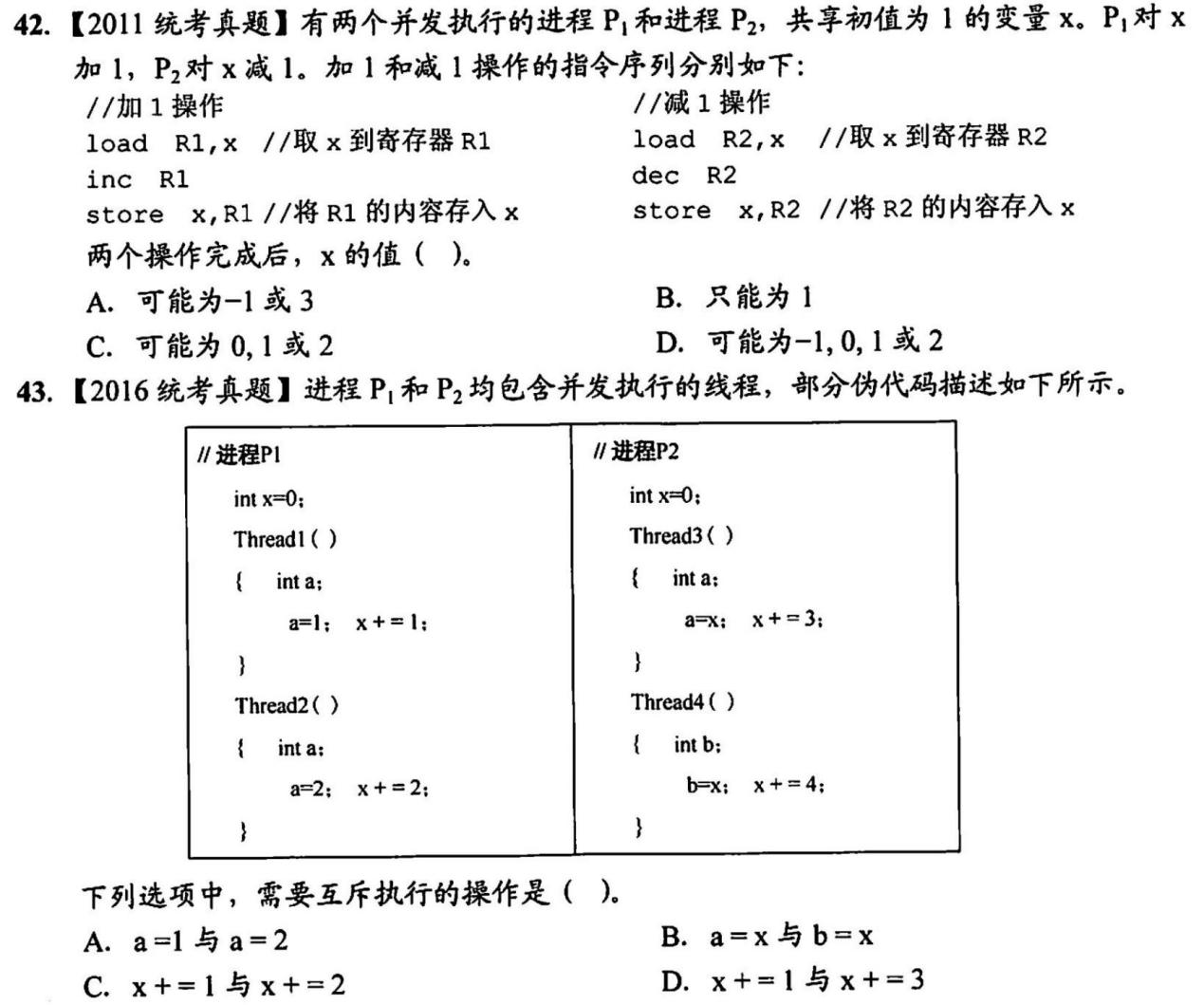
**同步与互斥**



B

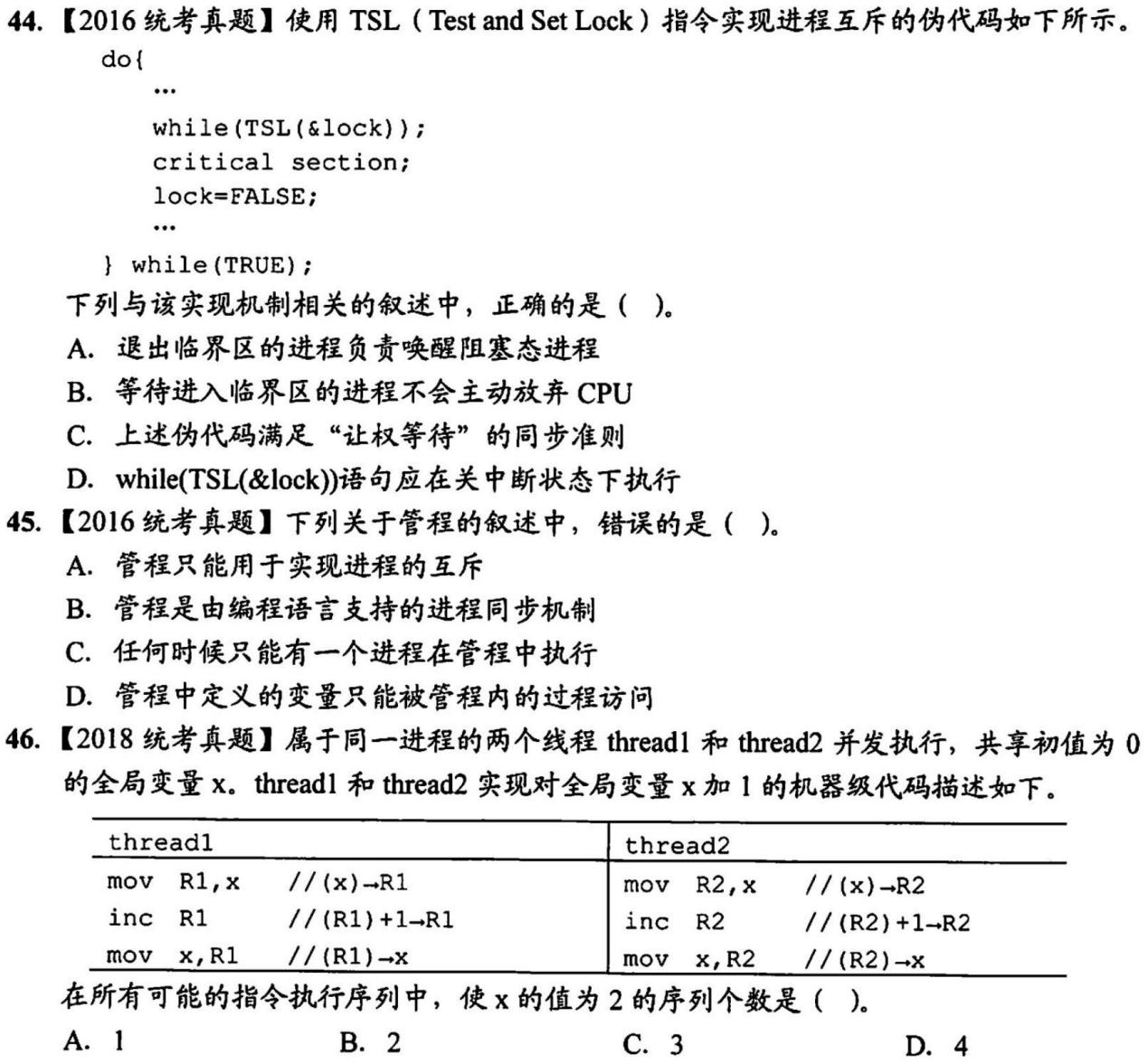


D



C

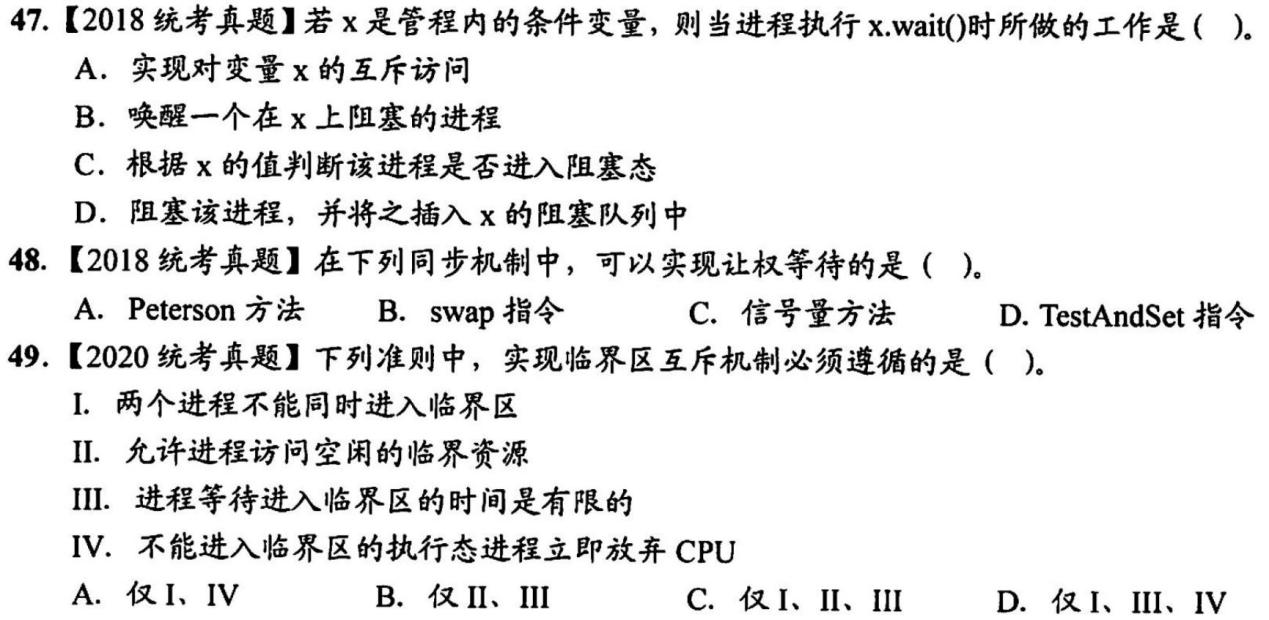
C



B

A

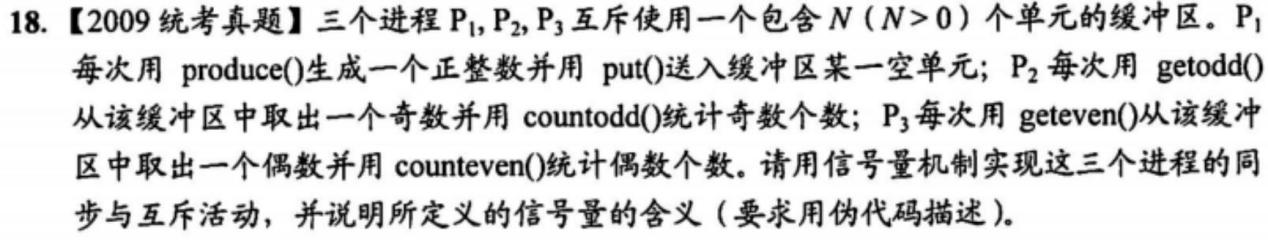
B



D

C

C



答：

semaphore mutex = 1 // 用于缓冲区的互斥访问

semaphore empty = N // 表示缓冲区的空单元数，初始为N

semaphore odd = 0 // 表示缓冲区中奇数的个数

semaphore even = 0 // 表示缓冲区中偶数的个数

int count\_odd = 0 // 统计奇数个数

int count\_even = 0 // 统计偶数个数

// P1，生产正整数并放入缓冲区

process P1 {

while (true) {

int item = produce();

P(empty);

P(mutex);

put(item);

V(mutex);

if (item % 2 == 0) {

V(even);

} else {

V(odd);

}

}

}

// P2，从缓冲区取奇数并统计

process P2 {

while (true) {

P(odd);

P(mutex)

int item = getodd();

count\_odd = countodd();

V(mutex);

V(empty);

}

}

// P3，从缓冲区取偶数并统计

process P3 {

while (true) {

P(even);

P(mutex);

int item = geteven();

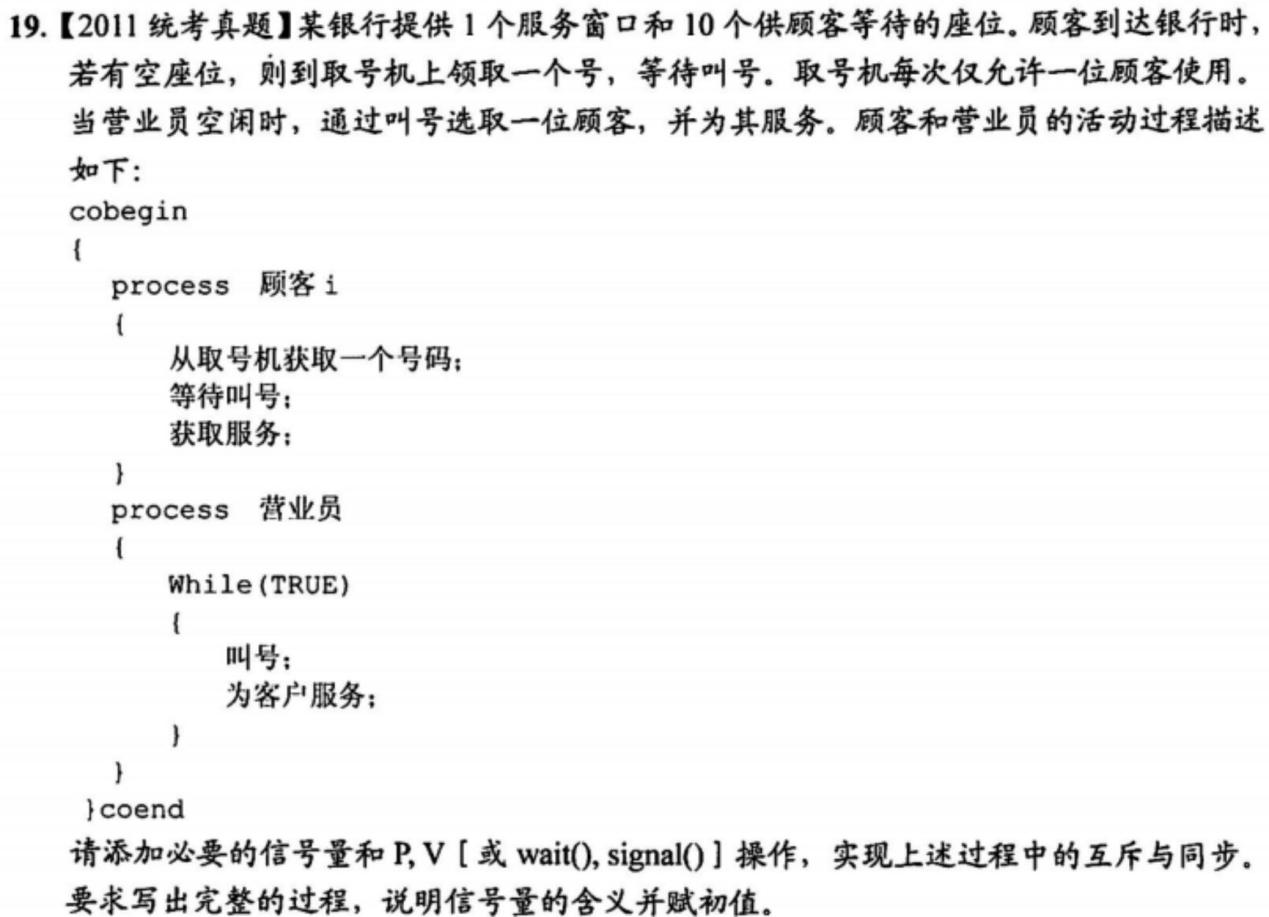
count\_even = counteven();

V(mutex);

V(empty);

}

}



**答：**

**semaphore waitingSeats = 10; // 空座位数量**

**semaphore ticketMachine = 1; // 取号机的互斥信号量**

**semaphore customerReady = 0; // 准备好接受服务的顾客数量**

**semaphore serve =0 ； // 服务同步信号**

**process 顾客i {**

**P(ticketMachine);**

**// 从取号机获取一个号码**

**V(ticketMachine);**

**P(waitingSeats);**

**// 等待叫号**

**V(customerReady);**

**P(serve);**

**// 获取服务**

**V(waitingSeats);**

**}**

**process 营业员 {**

**while (TRUE)**

**{**

**P(customerReady);**

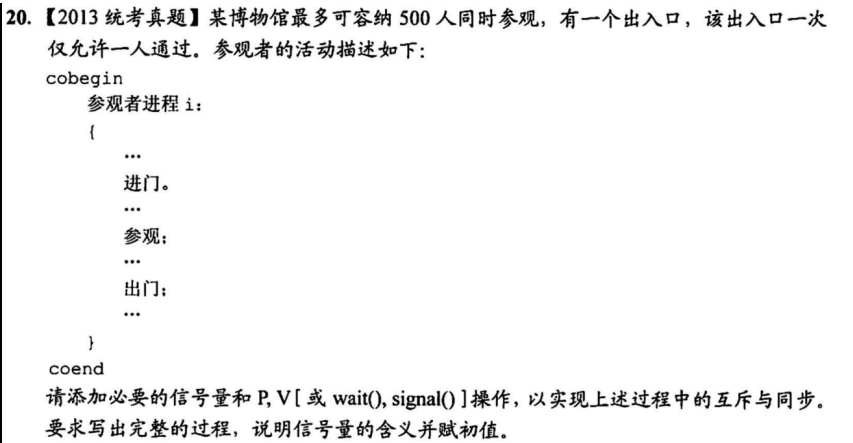
**// 叫号**

**// 为顾客服务**

**V(serve);**

**}**

**}**



**答：**

**semaphore door = 1; // 控制门的互斥信号量**

**semaphore visitors = 500; // 控制博物馆内人数的信号量**

**// 参观者进程i**

**process Visitor\_i {**

**P(visitors);**

**P(door);**

**// 进门**

**V(door);**

**// 参观**

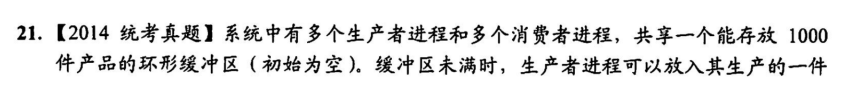
**P(door);**

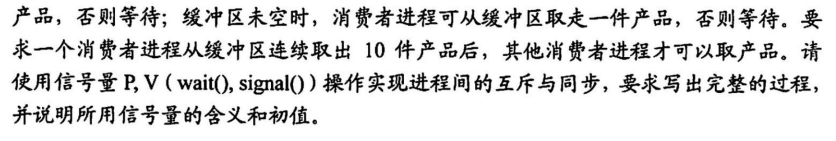
**// 出门**

**V(door);**

**V(visitors);**

**}**





**答：**

**semaphore empty = 1000; // 缓冲区空位数**

**semaphore full = 0; // 缓冲区产品数**

**semaphore mutex = 1; // 互斥信号量**

**semaphore turn = 1; // 控制消费者顺序，取10件后才能到下一个**

**// 生产者进程**

**process Producer {**

**while (TRUE) {**

**// 生产产品**

**P(empty);**

**P(mutex);**

**// 将产品放入缓冲区**

**V(mutex);**

**V(full);**

**}**

**}**

**// 消费者进程**

**process Consumer {**

**while (TRUE) {**

**P(turn);**

**for (int i = 0; i < 10; i++) {**

**P(full);**

**P(mutex);**

**// 从缓冲区取产品**

**V(mutex);**

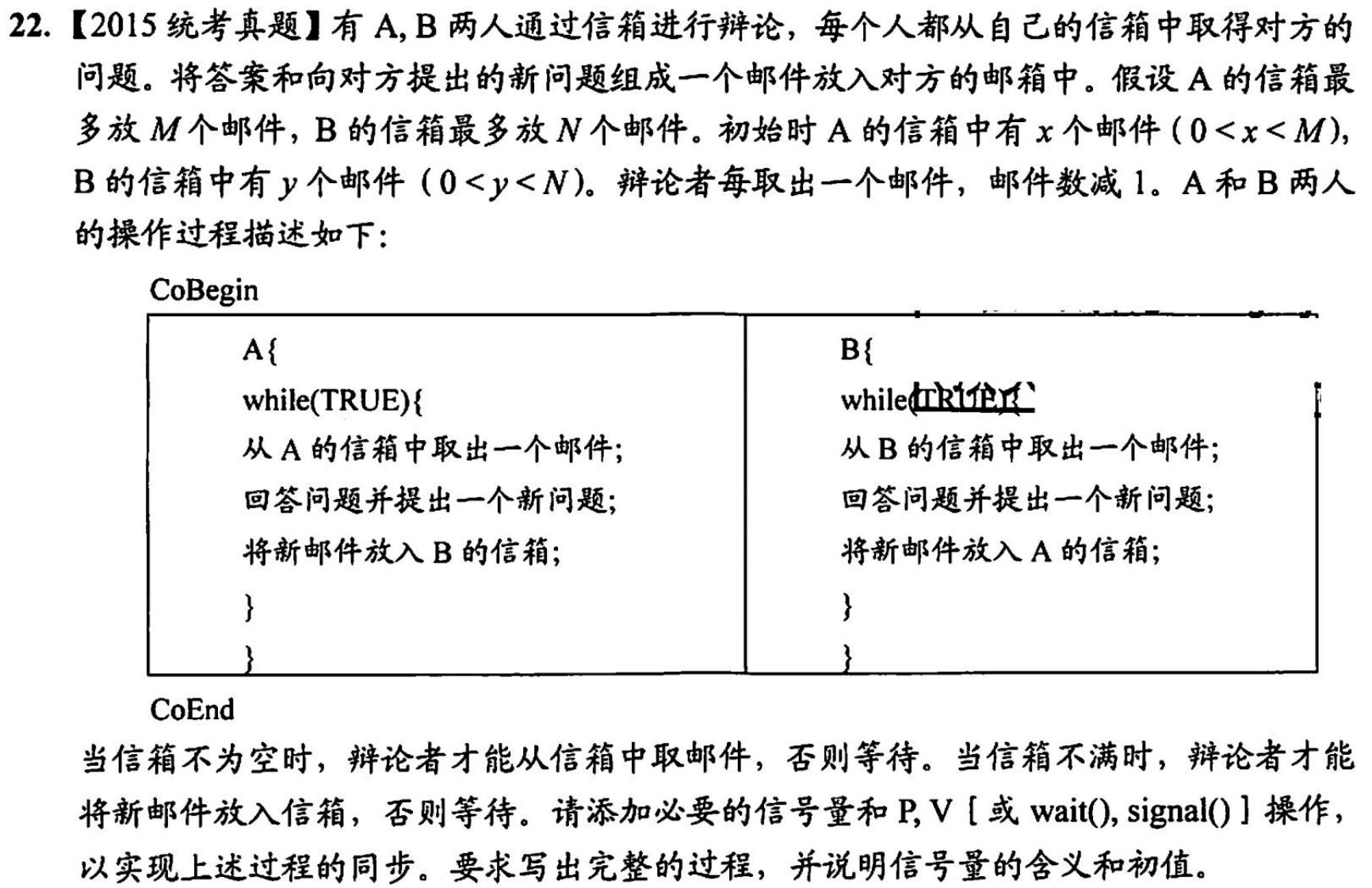
**V(empty);**

**}**

**V(turn); // 允许其他消费者取产品**

**}**

**}**



**答：**

**semaphore mailsInA = x; // A信箱中的邮件数量**

**semaphore mailsInB = y; // B信箱中的邮件数量**

**semaphore emptyInA = M - x; // A信箱中的空位数量**

**semaphore emptyInB = N - y; // B信箱中的空位数量**

**semaphore mutexA = 1; // A信箱的互斥信号量**

**semaphore mutexB = 1; // B信箱的互斥信号量**

**process A {**

**while (TRUE) {**

**P(mailsInA);**

**P(mutexA);**

**// 从A的信箱中取出一个邮件**

**V(mutexA);**

**V(emptyInA);**

**// 回答问题并提出一个新问题**

**P(emptyInB);**

**P(mutexB);**

**// 将新邮件放入B的信箱**

**V(mutexB);**

**V(mailsInB);**

**}**

**}**

**// 辩论者B的过程**

**process B {**

**while (TRUE) {**

**P(mailsInB);**

**P(mutexB);**

**// 从B的信箱中取出一个邮件**

**V(mutexB);**

**V(emptyInB);**

**// 回答问题并提出一个新问题**

**P(emptyInA);**

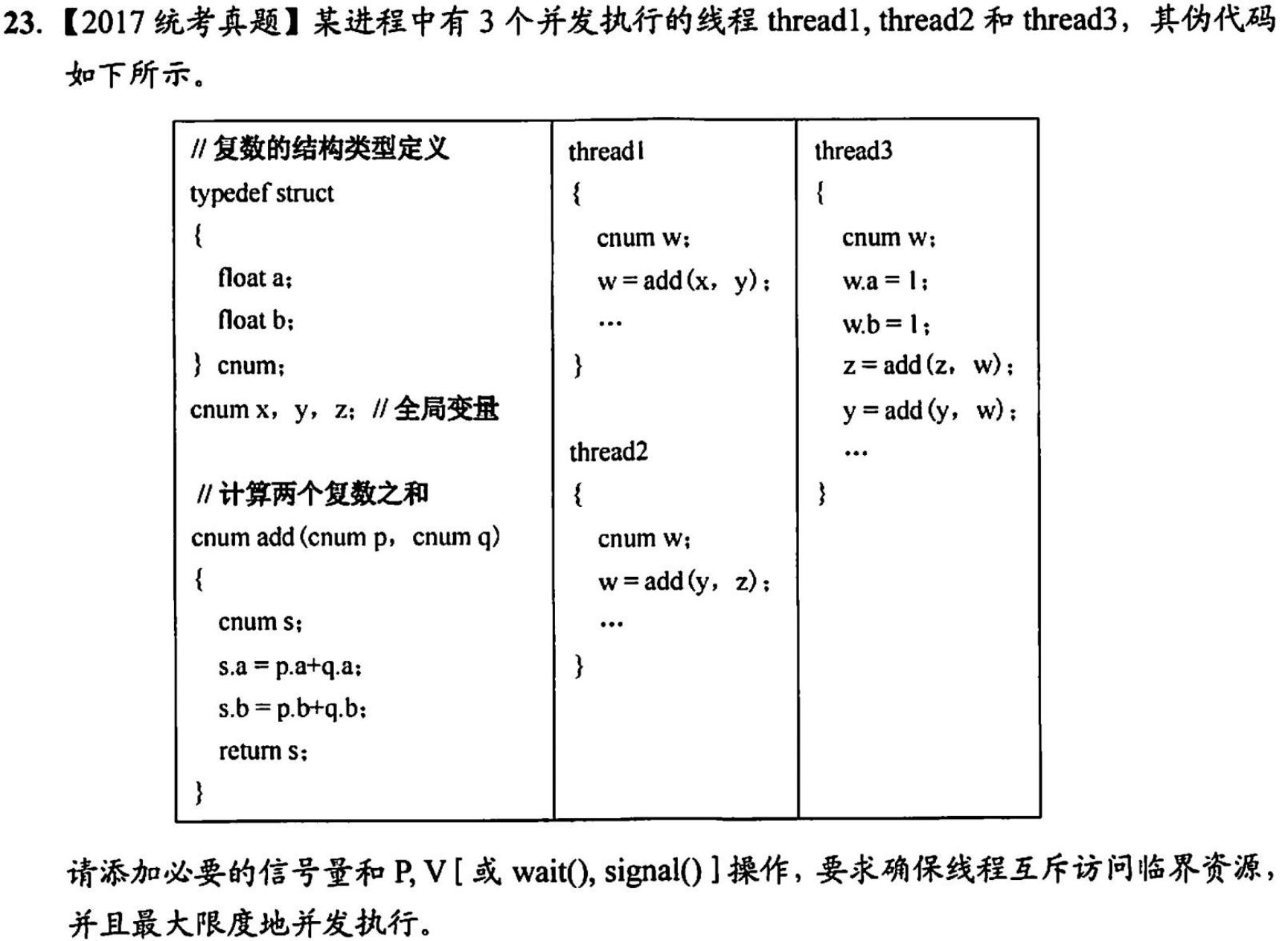
**P(mutexA);**

**// 将新邮件放入A的信箱**

**V(mutexA);**

**V(mailsInA);**

**}**



**答：**

**semaphore mutex\_y1; // 用于thread1和thread3对变量y的互斥访问**

**semaphore mutex\_y2; // 用于thread2和thread3对变量y的互斥访问**

**smeaphore mutex\_z; //用于变量z的互斥访问**

**//复数的结构类型定义**

**typedef struct**

**{**

**float a;**

**float b;**

**}cnum;**

**cnum x,y,z; //全局变量**

**//计算两个复数之和**

**cnum add(cnum p, cnum q)**

**{**

**cnum s;**

**s.a=p.a+q.a;**

**s.b=p.b+q.b;**

**return s;**

**}**

**thread1{**

**cnum w；**

**P（mutex\_y1）；**

**w=add（x，y）；**

**V（mutex\_y1）；**

**……**

**}**

**thread2{**

**cnum w；**

**P（mutex\_y2）；**

**P（mutex\_z）；**

**w=add（y，z）；**

**V（mutex\_y2）；**

**V（mutex\_z）；**

**……**

**}**

**thread3{**

**cnum w；**

**w.a=1；**

**w.b=1；**

**P（mutex\_z）；**

**z=add（z，w）；**

**V（mutex\_z）；**

**P（mutex\_y1）；**

**P（mutex\_y2）；**

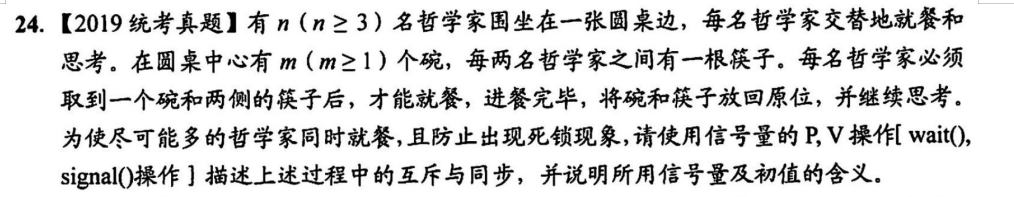
**y=add（y，w）；**

**V（mutex\_y1）；**

**V（mutex\_y2）；**

**……**

**}**



**答：**

**semaphore chopsticks[n]; // 假设有n根筷子，每根筷子一个信号量**

**for (int i = 0; i < n; i++) {**

**chopsticks[i] = 1; // 每根筷子的初始值为1**

**}**

**semaphore bowls = m; // 假设有m个碗**

**semaphore mutex = 1; // 避免死锁的互斥信号量**

**process 哲学家i {**

**while (TRUE) {**

**// 哲学家思考**

**P(mutex);**

**P(bowls);**

**P(chopsticks[i]);**

**P(chopsticks[(i + 1) % n]);**

**V(mutex);**

**// 哲学家就餐**

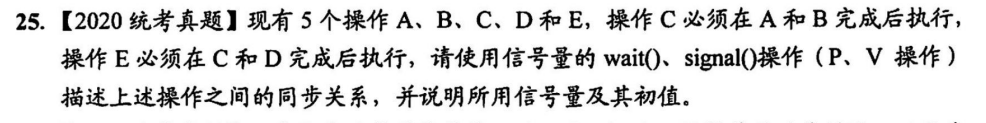
**V(chopsticks[i]);**

**V(chopsticks[(i + 1) % n]);**

**V(bowls);**

**}**

**}**



**答：**

**semaphore semA = 0; // A操作未完成**

**semaphore semB = 0; // B操作未完成**

**semaphore semC = 0; // C操作未完成**

**semaphore semD = 0; // D操作未完成**

**process A {**

**... // 执行操作A的相关任务**

**V(semA);**

**}**

**process B {**

**... // 执行操作B的相关任务**

**V(semB);**

**}**

**process C {**

**P(semA);**

**P(semB);**

**... // 执行操作C的相关任务**

**V(semC);**

**}**

**process D {**

**... // 执行操作D的相关任务**

**V(semD);**

**}**

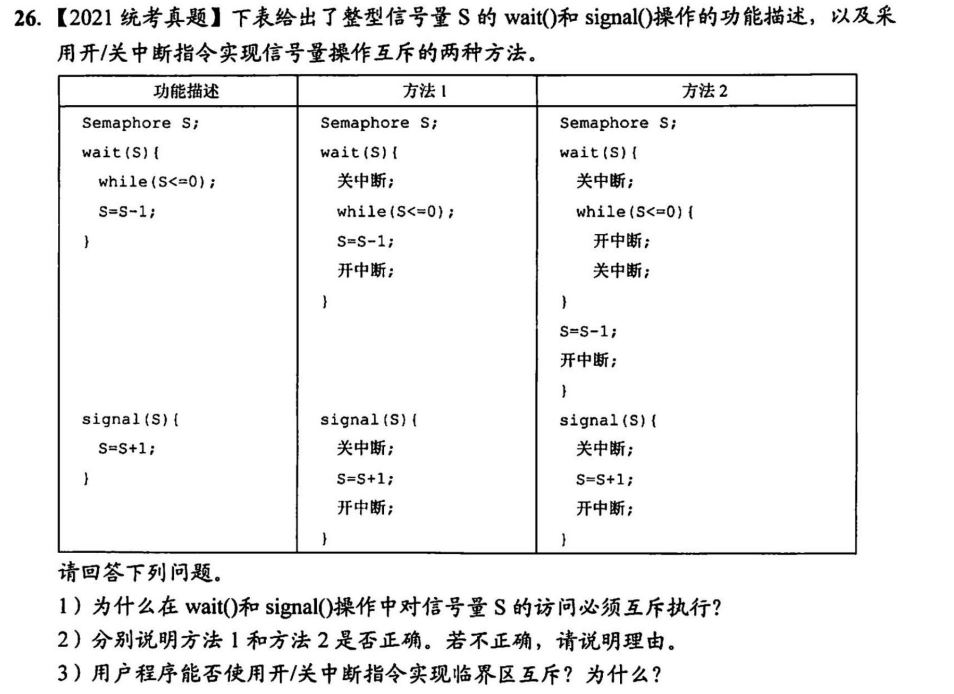
**process E {**

**P(semC);**

**P(semD);**

**... // 执行操作E的相关任务**

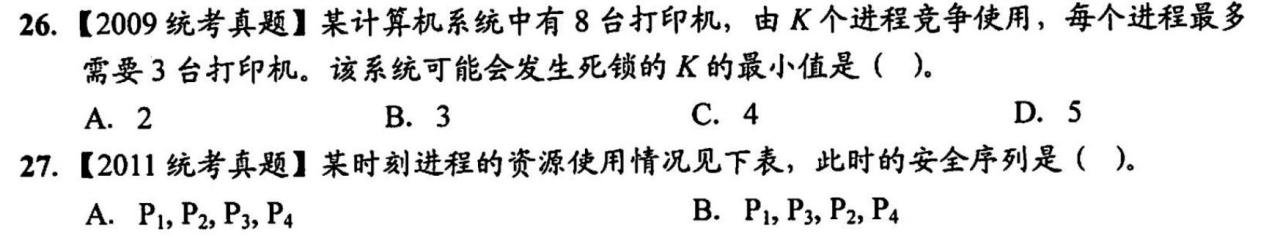
**}**



**答：**

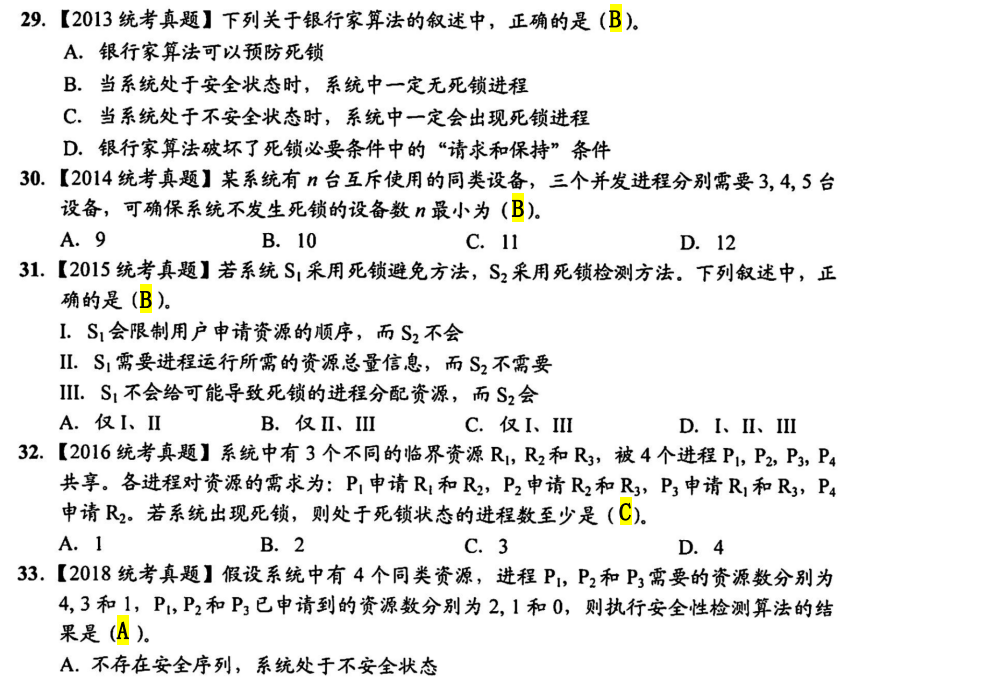
1. **信号量S是多个进程共享的变量，它们都能对S进行读写操作，为了避免数据不一致，对S的访问需要是互斥的。**
2. **方法1是错误的，在wait()中，当S<=0时，关中断后，其他进程都无法修改S的值，while语句会陷入死循环。方法2是正确的，方法2在每次循环中都开中断，使得其他进程可以修改S的值，从而避免while语句陷入死循环。**
3. **用户程序不能使用开/关中断指令实现临界区互斥，因为这两个指令都是特权指令，不能在用户态下执行，只能在内核态下执行。**

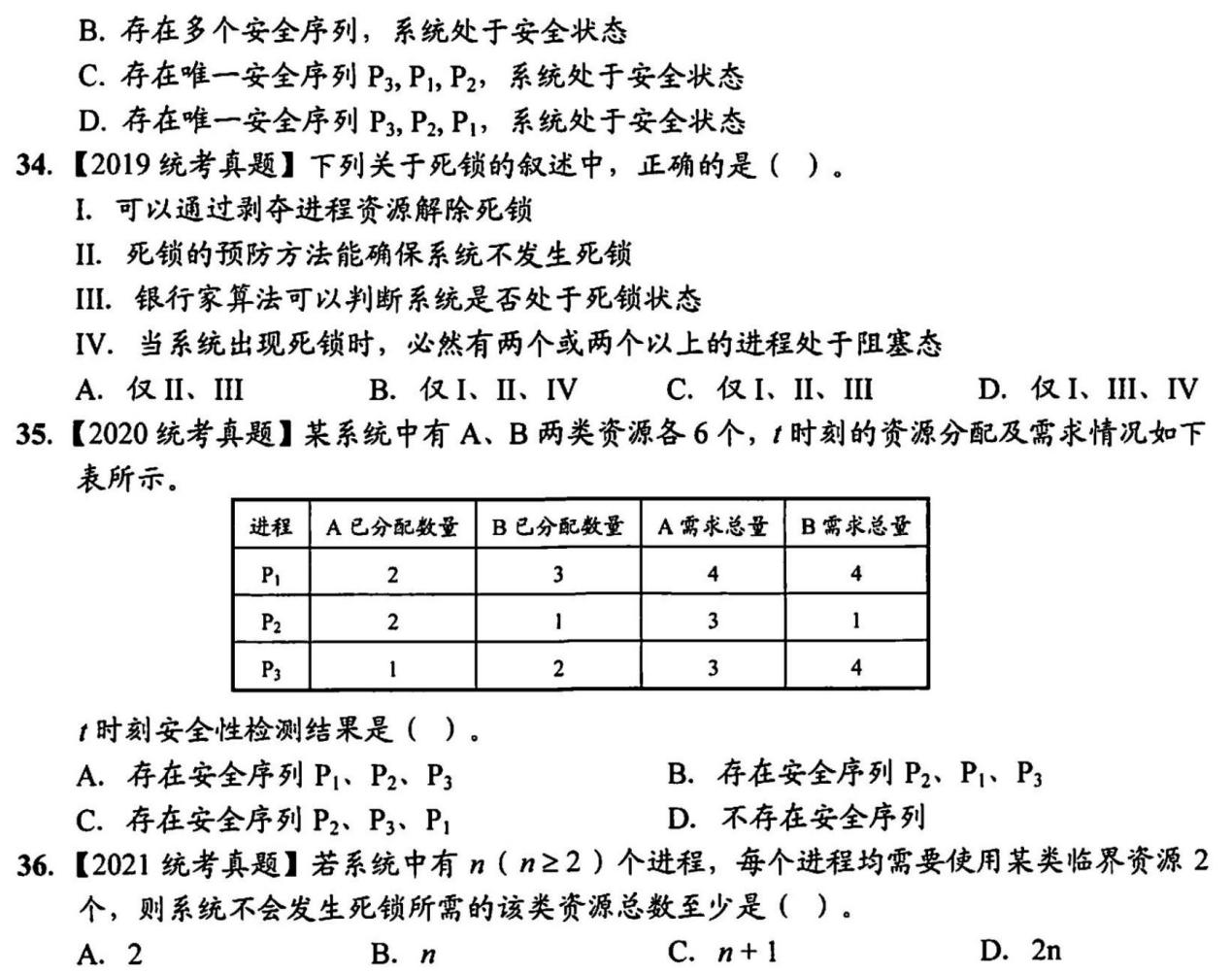
**死锁**



C

D





B

B

C