第二章　进程与线程

1. 设有一台计算机，有两条 I/O 通道，分别挂一台输入机和一台打印机。若要把输入机上的数据逐一地输入到缓冲区 B1 中，然后处理，并把结果搬到缓冲区 B2 中，最后在打印机上输出。请问：

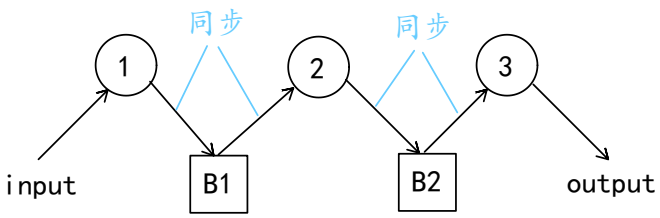
(1) 系统可设置哪些进程完成这一任务？这些进程间有什么具体制约关系？

进程一：把输入机上的数据逐一地输入到缓冲区 B1 中；
进程二：从缓冲区 B1 中取数据，处理后把结果放到缓冲区 B2 中；
进程三：从缓冲区 B2 中取数据，在打印机上输出。

这里的“逐一地”“缓冲区”等关键词表明是单个缓冲区而不是缓冲队列。那么进程一和进程二之间关于缓冲区 B1 存在同步的制约关系，进程二和进程三之间关于缓冲区 B2 存在同步的制约关系。(这里不需要互斥)

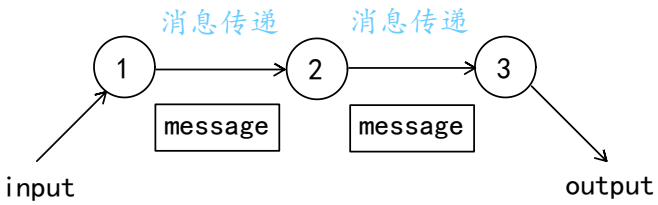
(2) 用 P-V 操作写出这些进程的同步算法。

```
struct semaphore B1_empty = 1, B1_full = 0, B2_empty = 1, B2_full = 0;
number temp1, temp2, temp3;
buffer B1, B2;
cobegin
void process1() {
    while(1) {
        temp1 = input();
        P(B1_empty);
        B1 = temp1;
        V(B1_full);
    }
}
void process2() {
    while(1) {
        P(B1_full);
        temp2 = B1_full;
        V(B1_empty);
        temp2 = processing(temp2);
        P(B2_empty);
        B2_full = temp2;
        V(B2_full);
    }
}
void process3() {
    while(1) {
        P(B2_full);
        temp3 = B2_full;
        V(B2_empty);
        output(temp3);
    }
}
coend
```



(3) 用 Send 和 Receive 原语写出这些进程的同步算法。

```
cobegin
void process1() {
    number item;
    message m;
    while (1) {
        item = input();
        m = build_message(item);
        Send(process2, &m);
    }
}
void process2() {
    number item;
    message m;
    while (1) {
        Receive(process1, &m);
        item = extract_item(&m);
        item = processing(item);
        m = build_message(item);
        Send(process3, &m);
    }
}
void process3() {
    number item;
    message m;
    while (1) {
        Receive(process2, &m);
        item = extract_item(&m);
        output(item);
    }
}
coend
```



2. “哲学家就餐问题”除课堂上介绍的方案外，有没有其他解决方案？有的话，请写出相应的解决方案。

有，比较容易想到的一种最简单的方案是，每次只允许一个哲学家就餐，也就是将所有的叉子视为一个整体的临界资源，但是缺点也很明显，会导致性能低下，因为五个叉子可以满足两个哲学家同时就餐。因此下面使用另一种方案。

大致方案是偶数序号的哲学家先拿左边的叉子，奇数序号哲学家先拿右边的叉子（释放的时候就不需要讲究先放左边还是先放后边了）。

```
struct semaphore forks[5] = {1, 1, 1, 1, 1};
void philosopher(i) {
    while (1) {
        think;
        if (i % 2) {
            P(forks[(i + 1) % 5]);
            P(forks[i])
        } else {
            P(forks[i]);
            P(forks[(i + 1) % 5]);
        }
        eat;
        V(forks[i]);
        V(forks[(i + 1) % 5]);
    }
}
```

3. a，b 两点之间是一段东西向的单行车道，现要设计一个自动管理系统，管理的规则如下：当 a，b 之间有车辆在行驶时同方向的车可以同时驶入 a，b 段，但另一个方向的车必须在 a，b 段以外等待；当 a，b 之间无车辆在行驶时，到达 a 点（或 b 点）的车辆可以进入 a，b 段，

但不能同时驶入；当某方向在 a，b 段行驶的车辆驶出了 a，b 段且暂无车辆进入 a，b 段时，应让另一方向等待的车辆进入 a，b 段行驶。请用 PV 操作作为工具，对 a，b 段实现正确管理以确保行驶安全。

可能会存在一个问题，就是一个方向一直有车辆，导致另一个方向无法进入车道，从而产生很长的排队。

还有个问题，是否要考虑车辆排队的过程，即先入先出，或先排队先进入。

这里先不考虑这两个问题。

这种问题大多数做法都是使用两个互斥量和两个计数值，这里尝试使用一个互斥量和一个计数值，计数值为正表示从 a 到 b 有车辆，计数值为负表示从 b 到 a 有车辆。但是缺点是发生任何一个事件（a 或 b 方向进入或离开车辆）都需要访问临界资源（使用同一个 mutex 信号量），可能会导致性能较低。

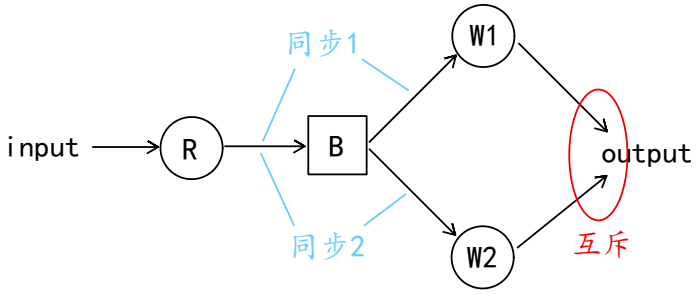
```
struct semaphore mutex = 1, s = 1;
// 车道内的车辆数
int count = 0; // 正数为从 a 到 b，负数为从 b 到 a
cobegin
void a_to_b() {
    P(mutex);
    if (count <= 0) {
        P(s);
    }
    count++;
    V(mutex);
    通过 a -> b;
    P(mutex);
    count--;
    assert(count >= 0);
    if (count == 0) {
        V(s);
    }
    V(mutex);
}
```

```
void b_to_a() {
    P(mutex);
    if (count >= 0) {
        P(s);
    }
    count--;
    V(mutex);
    通过 b -> a;
    P(mutex);
    count++;
    assert(count <= 0);
    if (count == 0) {
        V(s);
    }
    V(mutex);
}
coend
```

4. 假定有三个进程 R、W1、W2 共享一个缓冲区 B，B 中每次只能存放一个整数。进程 R 每次启动输入设备读一个整数且把它存放在缓冲区 B 中，若存放到缓冲区 B 中的是奇数，则由进程 W1 将其取出打印，否则由进程 W2 将其取出打印。要求用 PV 操作管理这 3 个并发进程，使它们能够正确同步工作。

这里使用了三个信号量实现 R 与 W1 的同步、R 与 W2 的同步，一个信号量实现了打印的互斥。

```
struct semaphore odd = 0, even = 0, empty = 1;
struct semaphore output_mutex;
buffer B;
cobegin
void R() {
    int temp;
    while (1) {
        temp = input();
        P(empty);
        B = temp;
        if (temp % 2) {
            V(odd);
        } else {
            V(even);
        }
    }
}
void W1() {
    int temp;
    while (1) {
        P(odd);
        temp = B;
        V(empty);
        P(output_mutex);
        output(temp);
        V(output_mutex);
    }
}
void W2() {
    int temp;
    while (1) {
        P(even);
        temp = B;
        V(empty);
        P(output_mutex);
        output(temp);
        V(output_mutex);
    }
}
coend
```



5. 假定在一个 CPU 上执行以下 5 个作业：（优先数小者优先级高）

作业号	1	2	3	4	5
到达时间	0	2	4	6	8
优先数	4	3	5	2	1
运行时间	3	6	4	5	2

当分别采用 FCFS、RR(时间片为 1)、非抢占式 SJF、抢占式 SJF、优先级调度 5 种调度算法时，试（1）画出调度图（2）计算每个作业的周转时间（3）计算平均周转时间。