四. (15 points) An assembly line is to produce a product C with four part As, and three part Bs. The worker of machining(加工) A and worker of machining B will produce two part As and one part B independently each time. Then the two part As or one part B will be moved to the station(工作台), which can hold at most 12 of part As and part Bs altogether. **Two part As must be put onto the station simultaneously.** The workers must exclusively put a part on the station or get it from the station. In addition, **the worker to make C must get all part of As and Bs for one product once.**

Using semaphores to coordinate the three workers who are machining part A, part B and the product C to manufacture the product without deadlock.

It is required that

1) definition and initial value of each semaphore, and

2) the algorithm to coordinate the production process for the three workers

should be given.

一、 分析

根据题意，注意以下 2 点：

（1） worker A 生产出的 part A 后，必须一次性将 2 个 part A 同时放入工作台，此时工作台上至少应有 2 个空位。

说明：

根据此要求，只有当工作台上至少有 2 个空位时，worker A 才能通过信号量操作进入临界区，一次性向工作台放入 2 个 part A。不允许 worker A 两次进入临界区，每次向工作台放入 1 个 part A，这样做有可能导致死锁，或阻塞 worker B 的操作。

例如，假设以多元信号量 empty 表示工作台中的空位数目。当前工作台中已有 6 个 part A、5 个 part B，剩余空位数为 empty=12-(6+5)=1。此时应允许 work B 生产 1 个 part B，并放入这剩余的 1 个空位中，但不允许 work A 将其生产的 2 个 part A 同时放入工作台。

考虑下述情况，work C 完全休眠，只有 work A、work B 采用分时方式，轮流占用 CPU 进入临界区，力图生产 part A、Part B 并放入工作台。如果按照下述交替执行顺序：

worker A	worker B	empty 值	结果
		1	
wait(empty)		0	worker A 获得 empty
	wait(empty)	-1	worker B 被阻塞，无法生产并放入 1 个 part B
wait(empty)		-2	worker A 被阻塞，无法生产并放入 2 个 part A

由于 worker A 先执行 wait 操作，占用了剩余的 1 个空位，但又无法同时生产 2 个 part A 并放入工作台，从而导致 worker B 也无法进入临界区并向工作台放入一个 part B。worker A、worker B 均阻塞在信号量 empty 上。

（2） worker C 在装配一个 part C 时，**必须同时获得** 4 个 part A 和 3 个 part B，因此要求：只有当工作台上至少有 4 个 part A 和 3 个 part B，worker C 才能从工作台上一次性同时取出 4 个 part A 和 3 个 part B，并装配一个 part C。

说明：类似于哲学家就餐问题中，要求哲学家左手、右手刀叉都可用时，方可通过信号量操作同时申请/拿起左手、右手刀叉进餐。不允许先通过信号量操作申请左手刀叉，再通过信号量操作申请右手刀叉，以防造成死锁。

同样，本题要求，只有当工作台至少有 4 个 part A 和 3 个 part B 时，worker C 才能通过信号量操作进入临界区，一次性取出 4 个 part A 和 3 个 part B。

由于工作台总容量为 12，因此 part A 在工作台上最多只能放置 12-3=9 个，否则工作台上可放置的 part B 将少于 3 个。part B 在工作台上最多只能放置 12-4=8 个，否则工作台上可放置的 part A 将少于 4 个。只有这样，才能保证工作台上有足够多的 part A 和 part C 用于装配 part C。

二、 方案 1. 引入变量计数工作台上 part A、part B 的数量，用二元信号量控制对变量和工作台的访问

设置如下的整型控制变量、信号量及其初值：

```
int count_A=0, count_B=0     //分别表示当前工作台中已有的 part A 和 part B 的数量
int   empty=12               //  工作台中空位数量, empty=12 - ( count_A+ count_B)
semaphore mutexA=1, mutexB=1
                             //二元信号量，分别用于控制对 count_A、count_B 互斥访问
semaphore mutexStation=1   //二元信号量，控制对工作台和 empty 的互斥访问（取/放零件）
semaphore suspend_A=0, suspend_B=0      //控制台无足够空位时，挂起 worker A、B
          suspend_C=0                   //控制台无足够零件时，挂起 worker C
```

worker A 生产 part A：

while (true) {

wait (mutex_A); //申请互斥访问 count_A、empty，判断工作台留给 part A 的空位数
wait(mutexStation); //申请对控制台和 empty 的访问权
if **(count_A <=7) and (empty>2)** {//还能再生产放置 2 个 A，最多可放 7+2=9 个 A，
以便至少给 part B 留下 12-9=3 个空位，防止工作台中没有足够多的 part B，导致 worker C 无法同时取出 3 个 C
反例：当 count_A=10，count_B=2，C 无法同时取出所需的 4 个 A、
3 个 B
生产 2 个 part A;
同时放入 2 个 A;
count_A = count_A + 2; //修改 count_A
empty = empty - 2; //修改 empty

signal(mutexStation); //释放对控制台和 empty 的访问权
signal(mutex_A);

if C 被阻塞，且新放入 2 个 A 后，满足 C 的零件需求

then signal(suspend_C) //控制台放入新零件，解挂 C

【需自行设计此步的具体实现方式】

} else {
signal(mutexStation); //释放对控制台和 empty 的访问权
signal(mutex_A);
wait(suspend_A) //控制台无足够空位，或不允许再放入 part A，
转入 waiting 态，**挂起自身**
}

}

worker B 生产 part B：

while (true) {

wait (mutex_B); //申请互斥访问 count_B、empty，判断工作台留给 part B 的空位数
wait(mutexStation); //申请对控制台和 empty 的访问权

if **(count_B <=7) and (empty>1)** { //还能再生产并放置 1 个 B，最多可放 7+1=8 个 B，
以便至少给 part A 留下 12-8=4 个空位，防止工作台中没有足够多的 part A，导致 worker C 无法同时取出 4 个 A
反例：当 count_A=2，count_B=10，C 无法同时取出所需的 4 个 A、
3 个 B，导致 3 个进程均 suspend 挂起/死锁

生产 1 个 part B;
1 个 B 放入控制台;
count_B=count_B+1; //修改 count_B
empty = empty - 1; //修改 empty

signal(mutexStation); //释放对控制台和 empty 的访问权
signal(mutex_B);

if C 被阻塞，且新放入 1 个 B 后，满足 C 的零件需求

then signal(suspend_C); //控制台放入新零件，解挂 C

【需自行设计此步的具体实现方式】

} else {
signal(mutexStation); //释放对控制台和 empty 的访问权
signal(mutex_B)
wait(suspend_B) //控制台无足够空位，，或不允许再放入 part B，转入 waiting 态，**挂起自身**
}

}

worker C 装配：

while (true) {

wait (mutex_A); //申请互斥访问 count_A，判断 part A 数量
wait (mutex_B); //申请互斥访问 count_B，判断 part B 数量
if (count_A >=4) and (count_B >=3) { //有足够多的 part A, part B 用于装配
wait(mutexStation); //申请对控制台的访问权
同时取出 4 个 part A、3 个 part B;
count_A = count_A - 4; //修改 count_A
count_B = count_B - 3; //修改 count_B
生产 1 个 part C;
empty = empty + 7;

```
            signal(mutexStation)

            signal(mutex_A);
            signal(mutex_B);

            if A 被阻塞

            then signal(suspend_A);        //控制台新增空位，解挂 A, B

            if B 被阻塞  then signal(suspend_B)

            【需自行设计这两步的具体实现方式】

    }  else {
            signal(mutex_A);
            signal(mutex_B);
            wait(suspend_C)      //控制台无足够零件，转入 waiting 态，挂起自身
            }
}
```

三、 方案 2. 只保留方案 1 中的 mutexStation 和 suspend_A、suspend_B、suspend_C 四个二元信号量，去掉 mutex_A、mutex_B。

```
int count_A=0, count_B=0      //分别表示当前工作台中已有的 part A 和 part B 的数量
int   empty=12                //  工作台中空位数量, empty=12 - ( count_A+ count_B)
semaphore mutexStation=1   //二元信号量，控制对工作台和 count_A、count_B、empty
                              的互斥访问（取/放零件）
semaphore suspend_A=0, suspend_B=0      //控制台无足够空位时，挂起 worker A、B
          suspend_C=0                   //控制台无足够零件时，挂起 worker C
```

思路：参照哲学家就餐问题，worker A、worker B、worker C 互为邻居，mutexStation 类似于该问题中的二元信号量 mutex，suspend_A、suspend_B、suspend_C 类似于二元信号量 self[i]。

worker A 生产 part A：

while (true) {

wait(mutexStation);　　//申请对 count_A、empty 和控制台的访问权

if　**(count_A <=7) and (empty>2)**　{//还能再生产放置 2 个 A，最多可放 7+2=9 个 A，

以便至少给 part B 留下 12-9=3 个空位，防止工作台中没有足够多的 part B，导致 worker C 无法同时取出 3 个 C

反例：当 count_A=10，count_B=2，C 无法同时取出所需的 4 个 A、

3 个 B

生产 2 个 part A;

同时放入 2 个 A;

count_A = count_A + 2;　　//修改 count_A

empty = empty - 2;　　//修改 empty

signal(mutexStation);　　//释放对控制台和 empty 的访问权

signal(suspend_C)　　//控制台放入新零件，解挂/唤醒 worker C

} else {

signal(mutexStation);　　//释放对控制台和 count_A、empty 的访问权

wait(suspend_A)　//控制台无足够空位，或不允许再放入 part A，

转入 waiting 态，**挂起自身**

}

}

worker B 生产 part B：

while (true) {

wait(mutexStation);　　//申请对 count_B、empty 和控制台的访问权

if　**(count_B <=7) and (empty>1)**　{ //还能再生产并放置 1 个 B，最多可放 7+1=8 个 B，

以便至少给 part A 留下 12-8=4 个空位，防止工作台中没有足够多的 part A，导致 worker C 无法同时取出 4 个 A

反例：当 count_A=2，count_B=10，C 无法同时取出所需的 4 个 A、

3 个 B，导致 3 个进程均 suspend 挂起/死锁

生产 1 个 part B;

1 个 B 放入控制台;

count_B=count_B+1;　　//修改 count_B

empty = empty - 1;　　//修改 empty

signal(mutexStation);　　//释放对控制台和 count_B、empty 的访问权

```
            signal(suspend_C);      //控制台放入新零件，解挂/唤醒 worker C
        }    else   {
                    signal(mutexStation);   //释放对控制台和 count_B、empty 的访问权
                    wait(suspend_B)   //控制台无足够空位，，或不允许再放入 part B，转入
                                      waiting 态，挂起自身
                }
}
```

worker C 装配：

```
while (true) {
    wait(mutexStation);              //申请对 count_A、count_B 和控制台的访问权
    if   (count_A >=4) and (count_B >=3) { //有足够多的 part A, part B 用于装配

            同时取出 4 个 part A、3 个 part B;
            count_A = count_A - 4;        //修改 count_A
            count_B = count_B - 3;        //修改 count_B
            生产 1 个 part C;
            empty = empty + 7;
            signal(mutexStation)

            signal(suspend_A);       //控制台新增空位，解挂 A, B
            signal(suspend_B)
    }   else {
            signal(mutexStation);    //释放对控制台、count_A、count_B 的控制权
            wait(suspend_C)      //控制台无足够零件，转入 waiting 态，挂起自身
            }
}
```

四、 方案 3. 【参考书提供，可能存在问题】

```
Semaphore stsem = 12;                        //  工作台上的 12 个工位
Semaphore Asem = 12-3=9,     Asem1 = 0;   // part A
Semaphore Bsem=10-4=8,       Bsem1=0;     // part B
Semaphore mutex(1);                          //  互斥
```

【如果扩展此题，允许多个 worker A、多个 worker B，且 worker A 一次生产需要 m1>1 个空位，worker B 一次生产也需要 m2>1 个空位，大概率会出现这两类进程相互阻塞，严重影响执行效率。

】

```
worker A
{
   while(1){
      Generate_two_partAs
      P(Asem);        //判断是否还允许生产 Part A，以防止造成与 part B 数目不匹配
      P(Asem);
      P(stsem);       //判断工作台是否有空位容纳 part A
      P(stsem)

      P(mutex);      //获取对工作台的控制权，放置 2 个 part A
      Put_twoAs();t
      V(mutex);

      V(Asem1);    //通知、唤醒 Work C，有 2 个 part A 放入工作台
      V(Asem1);
   }
}

worker B
{
   while(1){
      generate_one_ partB();
      P(Bsem);              //判断是否还允许生产 Part B,，以防止造成与 part A 数目不匹配
      P(stsem);             //判断工作台是否有空位容纳 part B

      P(mutex);            //获取对工作台的控制权，放置 1 个 part B
      Put_oneB();
      V(mutex);

      V(Bsem1);           //通知、唤醒 Work C，有 1 个 part B 放入工作台
    }
}
```

```
Worker C
{
while(1){
    for( int i=0;i<4;i++)          //判断工作台上是否有 4 个 part A
        P(Asem1);
    for( int i=0;i<3;i++)          //判断工作台上是否有 3 个 part B
        P(Bsem1);

    P(mutex)                       //从工作台上取 4 个 A、3 个 B，组装 1 个 C
    Get_four_As_and_three_Bs();
    V(mutex);

    for( int i=0;i<4;i++)          //唤醒因不允许再生产 A 而被阻塞的 worker A
        V(Asem);
    for( int i=0;i<3;i++)          //唤醒因不允许再生产 B 而被阻塞的 worker B
        V(Bsem);

    for(int i=0;i<7;i++)           //工作台上空出 4+3=7 个空位，唤醒因无空位被阻塞
                                     的 worker A、worker B
    V(stsem);

  }
}
```

可能存在的问题：

假设当前工作台中已有 6 个 part A、5 个 part B，Asem=9-6=3, Bsem=8-5=3, 剩余空位数为 stsem=12-(6+5)=1，此时应允许 work B 生产 1 个 part B，并放入这剩余的 1 个空位中。

但是，如果 worker A、B 按照下述方式交替执行：

worker A	worker B	Asem	Bsem	stsem
		3	3	1
P(Asem);		2		
P(Asem);		1		
P(stsem);				0
	P(Bsem);		2	
	P(stsem)			-1
P(stsem);				-2

worker A、worker B 均被阻塞，明显不合理。

原因：没有将 worker A 需要的 2 个零件 A 的空位当作一个整体去一次性申请，而是多次申请，类似于哲学家就餐问题中先申请左手刀叉，再申请右手刀叉。