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.

1. **试题分析**

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。

1. 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。

1. **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);

signal(suspend\_C) //控制台放入新零件，解挂C

} 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);

signal(suspend\_C); //控制台放入新零件，解挂C

} 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);

signal(suspend\_A); //控制台新增空位，解挂A, B

signal(suspend\_B)

**}** 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**

}

**}**

1. **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**

}

**}**

1. **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

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。