作业 7

7.1 某系统存在 4 个进程和 5 份可分配资源，当前的资源分配情况和最大需求如下表所示。求满足安全状态下 X 的最小值。请写出解题分析过程。

| | Allocated | | | | | Maximum | | | | | Available | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| process A | 5 | 4 | 2 | 5 | 3 | 5 | 5 | 2 | 5 | 5 | 0 | x | 1 | 0 | 0 |
| process B | 3 | 1 | 3 | 2 | 5 | 3 | 3 | 4 | 2 | 5 | | | | | |
| process C | 2 | 0 | 3 | 4 | 1 | 6 | 0 | 6 | 4 | 1 | | | | | |
| process D | 4 | 2 | 3 | 5 | 2 | 10 | 2 | 4 | 6 | 11 | | | | | |

设五种资源分别为 R1, R2, R3, R4, R5。

则当前资源请求矩阵为:

| | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| A | 0 | 1 | 0 | 0 | 2 |
| B | 0 | 2 | 1 | 0 | 0 |
| C | 4 | 0 | 3 | 0 | 0 |
| D | 6 | 0 | 1 | 1 | 9 |

显然，如果 X 为 0 、1 均不能让任一线程得到资源请求的满足。

假设 X 为 2，则现在线程 B 可以得到所请求资源，将资源分配给 B 后的请求矩阵为:

| | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| A | 0 | 1 | 0 | 0 | 2 |
| B | 0 | 0 | 0 | 0 | 0 |
| C | 4 | 0 | 3 | 0 | 0 |
| D | 6 | 0 | 1 | 1 | 9 |

线程 B 退出后的可用资源为

| R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|
| 3 | 3 | 4 | 2 | 5 |

则现在可以把资源分配给线程 A，请求矩阵变为:

| | R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|---|
| A | 0 | 0 | 0 | 0 | 0 |
| C | 4 | 0 | 3 | 0 | 0 |
| D | 6 | 0 | 1 | 1 | 9 |

线程 A 退后的可用资源为:

| R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|
| 8 | 7 | 6 | 7 | 8 |

同理，将资源分配给线程 C，线程 C 退出后的可用资源为:

| R1 | R2 | R3 | R4 | R5 |
|---|---|---|---|---|
| 10 | 7 | 9 | 11 | 9 |

最后可用将资源分配给 D ，此时时为安全状态，x 的最小值为 2。

7.2 两进程 A 和 B 各需要数据库中的 3 份记录 1、2、3，若进程 A 以 1、2、3 的顺序请求这些资源， 进程 B 也以同样的顺序请求这些资源，则将不会产生死锁。但若进程 B 以 3、2、1 的顺序请求这些资源，则可能会产生死锁。这 3 份资源存在 6 种可能的请求顺序，其中哪些请求顺序能保证无死锁产生？请写出解题分析过程。

分析问题，死锁可能发生的情况是：进程 A 已经获取了资源 1，而进程 B 已经获取了资源 3，然后 A 和 B 分别请求资源 3 和资源 1，由于每个进程都持有对方需要的资源并且等待对方持有的资源，所以就产生了死锁。现在假设 A 的请求顺序为 1、2、3，对 B 的请求顺序可能为：123、132、213、231、312 和 321。那么首先按照同样

的顺序请求资源不会发生死锁，也就是按照 123 的顺序请求资源，这种策略被称为资源排序法，是一种常见的避免死锁的策略。通常让所有进程按照相同顺序来请求资源就可以避免发生死锁。

而按照 132 的请求顺序也不会发生死锁，原因是假设 A 抢到资源 1，则 B 申请资源 1 时会被阻塞而挂起，此后 A 可以继续申请资源 2，3，不会产生死锁 。假设 B 抢到资源 1，则 A 申请资源 1 时会被阻塞挂起，此后 B 可以继续申请资源 3，2，也不会产生死锁 。

（以上考虑的调度策略即为，如果一个进程申请不到资源，则会被阻塞直到占用资源的进程运行完毕。）

7.3 设有两个优先级相同的进程 T1，T2 如下。令信号量 S1，S2 的初值为 0，已知 z=2，试 问 T1，T2 并发运行结束后 x=?y=?z=?

线程 T1

y:=1;

y:=y+2;

V(S1);

z:=y+1;

P(S2);

y:=z+y;

线程 T2

x:=1;

x:=x+1;

P(S1);

x:=x+y;

V(S2);

z:=x+z;

注:请分析所有可能的情况，并给出结果与相应执行顺序。

P 操作是一个原子操作，用于测试信号量的值。如果信号量的值大于 0，那么就将其减 1；如果信号量的值为 0，那么就阻塞当前线程，直到信号量的值大于 0。V 操作也是一个原子操作，用于增加信号量的值。如果有其他线程在等待这个信号量，那么相当于唤醒一个等待线程。

因此：线程 T1 和 T2 都会在一开始就执行一些赋值操作，然后执行 V(S1)或 P(S1)操作。由于 S1 的初值为 0，所以 T2 会在 P(S1)操作处阻塞，直到 T1 执行 V(S1)操作。

此时 x 的值为 2.y 的值为 3.

接下来，由于 T2 在 P(S1)操作后不再被阻塞，所以它可以继续执行 x:=x+y;和 V(S2);而 T1 则会在 T2 执行 V(S2)之前阻塞，这之前的 x,y,z 的值是确定的为：5，3，2

往后开始出现分歧：

T1 可能会阻塞，而 T2 不会再被阻塞，所以当 T2 执行完 V(S2);后

如果 T2 先执行 z:=x+z;，后 T1 执行 z:=y+1;那么最终 x,y,z 的值为 5，7，4

如果 T1 先执行 z:=y+1;P(S2); y:=z+y;后，T2 再执行 z:=x+z;

那么最终 x,y,z 的值为：5，7，9

如果 T1 先执行 z:=y+1，T2 再执行 z:=x+z;，然后 T1 再执行 P(S2); y:=z+y;

那么最终 x,y,z 的值为：5，12，9

7.4 在生产者-消费者问题中，假设缓冲区大小为 5，生产者和消费者在写入和读取数据时都会更新写入/读取的位置 offset。现有以下两种基于信号量的实现方法，

方法一

```
Class BoundedBuffer {
    mutex = new Semaphore(1);
    fullBuffers = new Semaphore(0);
    emptyBuffers = new Semaphore(n);
}
BoundedBuffer::Deposit(c) {
    emptyBuffers->P();
    mutex->P();
    Add c to the buffer;
    offset++;
    mutex->V();
    fullBuffers->V();
}
BoundedBuffer::Remove(c) {
    fullBuffers->P();
    mutex->P();
    Remove c from buffer;
    offset--;
    mutex->V();
    emptyBuffers->V();
}
```

方法二：

```
Class BoundedBuffer {
    mutex = new Semaphore(1);
    fullBuffers = new Semaphore(0);
    emptyBuffers = new Semaphore(n);
}
BoundedBuffer::Deposit(c) {
    mutex->P();
    emptyBuffers->P();
    Add c to the buffer;
    offset++;
    fullBuffers->V();
    mutex->V();
}
BoundedBuffer::Remove(c) {
    mutex->P();
    fullBuffers->P();
    Remove c from buffer;
    offset--;
    emptyBuffers->V();
    mutex->V();
}
```

请对比分析上述方法一和方法二，哪种方法能让生产者、消费者进程正常运行，并说明分析原因。

在方法一中，生产者和消费者在尝试获取或释放缓冲区之前先检查缓冲区是否为空或满。这是通过在调用 P()操作之前检查 emptyBuffers 或 fullBuffers 信号量来实现的。如果缓冲区已满，生产者会等待直到有可用的空缓冲区；如果缓冲区为空，消费者会

等待直到有可用的满缓冲区。这种方法可以确保生产者和消费者都不会在尝试读写缓冲区时阻塞。

在方法二中，生产者和消费者在尝试获取或释放缓冲区之前不会检查缓冲区是否为空或满。相反，它们会直接尝试获取互斥锁，并在成功获取后才检查 emptyBuffers 或 fullBuffers 信号量。这可能会导致生产者和消费者在尝试读写缓冲区时阻塞，因为它们必须等待互斥锁可用。例如，如果缓冲区为空，那么消费者（Remove）会首先获取互斥锁，然后尝试执行 fullBuffers->P()。但是，由于 fullBuffers 的值为 0，消费者会在这里阻塞。同时，由于消费者已经获取了互斥锁，所以生产者（Deposit）无法获取互斥锁，也就无法向缓冲区添加数据。这就导致了死锁：消费者在等待生产者生产数据，而生产者在等待消费者释放互斥锁。

所以只有方法一可用使消费者和生产者都正确运行。