作业 7

毕定钧 2021K8009906014

本次作业包含:

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	

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。

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 $T1$	线程 $T2$
$y := 1;$	$x := 1;$
$y := y + 2;$	$x := x + 1;$
$V(S1);$	$P(S1);$
$z := y + 1;$	$x := x + y;$
$P(S2);$	$V(S2);$
$y := z + y;$	$z := x + z;$

注: 请分析所有可能的情况，并给出结果与相应执行顺序。

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。

7.1

其需要的资源如下：

	Allocated	Maximum	Need	Available
processA	5 4 2 5 3	5 5 2 5 5	0 1 0 0 2	0 x 1 0 0
processB	3 1 3 2 5	3 3 4 2 5	0 2 1 0 0	
processC	3 1 3 2 5	3 3 4 2 5	4 0 3 0 0	
processD	2 0 3 4 1	6 0 6 4 1	6 0 1 1 9	

只有进程 B 满足需求，将资源分配给 B，此时剩余 0 x-2 0 0 0，完成 B 后释放，得到新表格：

	Allocated	Maximum	Need	Available
processA	5 4 2 5 3	5 5 2 5 5	0 1 0 0 2	3 x+1 4 2 5
processC	3 1 3 2 5	3 3 4 2 5	4 0 3 0 0	
processD	2 0 3 4 1	6 0 6 4 1	6 0 1 1 9	

此时进程 A 可以使用，将资源分配给 A，剩余 3 x 5 2 3，完成 A 后释放，得到新表格：

	Allocated	Maximum	Need	Available
processC	3 1 3 2 5	3 3 4 2 5	4 0 3 0 0	8 x+5 6 7 8
processD	2 0 3 4 1	6 0 6 4 1	6 0 1 1 9	

此时进程 C 满足需求，将资源分配给 C，剩余 4 x+5 3 7 8，完成 C 后释放，得到表格：

	Allocated	Maximum	Need	Available
processD	2 0 3 4 1	6 0 6 4 1	6 0 1 1 9	11 x+6 9 9 13

只需要保证中间剩余资源全为正数，即 $x \geq 2$ 即可。

7.2

A 以 1、2、3 的顺序请求资源，B 的请求顺序有 6 种：1，2，3 − 1，3，2 − 2，1，3 − 2，3，1 − 3，1，2 − 3，2，1，其中只有 1，2，3 和 1，3，2 不会出现死锁。

7.3

观察代码发现，$T1$ 与 $T2$ 的前两行不相关，会自行执行，但 $T2$ 需要等待 $T1$ 执行完第三行后才能执行第三行，$T1$ 需要等待 $T2$ 执行完第五行后执行第五行，即 $x = x + y$ 一定比 $y = z + y$ 先执行。根据执行顺序可以得到表格如下：

	x	y	z
$x=1;y=1$	1	1	
$x=x+1;y=y+2$	2	3	
$z=y+1$	2	3	4
$x=x+y$	5	3	4
$y=z+y$	5	7	4
$z=x+z$	5	7	9

	x	y	z
$x=1;y=1$	1	1	
$x=x+1;y=y+2$	2	3	
$z=y+1$	2	3	4
$x=x+y$	5	3	4
$z=x+z$	5	3	9
$y=z+y$	5	12	9

	x	y	z
$x=1;y=1$	1	1	
$x=x+1;y=y+2$	2	3	
$x=x+y$	5	3	
$z=x+z$	5	3	$(z)+5$
$z=y+1$	5	3	4
$y=z+y$	5	7	4

其他的顺序不会影响结果，故不再列出，即最终会出现三种结果。

7.4

方法一能够正常运行。方法二会导致如下情况：进程 A 写入数据 c，它请求到了 $mutex$ 但是却需要等待 $fullBuffers$ 释放；而另一个进程 B 需要删除 c，在等待 $mutex$，无法释放 $fullBuffers$。