第二章+第三章作业

班级: 07112303 学号: 1120231863 姓名: 左逸龙

2-9 有五个作业正等待运行，它们估计运行时间分别为 9、6、3、5 和x。为了获得最小的平均周转时间，应按照什么顺序运行它们?（你给出的答案应是x的函数）。

答： 为了使平均周转时间最短，应采用“最短作业优先调度法”，即总是优先运行作业时间最短的作业。

对 5 个作业分别编号为 $1 \sim 5$，则运行顺序：

$$\begin{cases} 5 \to 3 \to 4 \to 2 \to 1 (0 < x \leq 3) \\ 3 \to 5 \to 4 \to 2 \to 1 (3 < x \leq 5) \\ 3 \to 4 \to 5 \to 2 \to 1 (5 < x \leq 6) \\ 3 \to 4 \to 2 \to 5 \to 1 (6 < x \leq 9) \\ 3 \to 4 \to 2 \to 1 \to 5 (x > 9) \end{cases}$$

2-12 假定系统有四道作业，它们的提交时间及估计执行时间（以小时为单位）如下表所示。在单道批处理系统中，采用先来先服务、最短作业优先和响应比高者优先的调度算法时，分别计算下表列出作业的平均周转时间。

作业号	提交时间（小时）	估计运行时间（小时）
1	8.0	2.0
2	9.0	1.2
3	9.5	0.5
4	10.2	0.3

答：① 先来先服务（FCFS）：

1. 8.0 时：作业 1 到达，开始运行
2. 9.0 时：作业 2 到达，等待调度
3. 9.5 时：作业 3 到达，等待调度
4. 10.0 时：作业 1 完成，开始运行作业 2
5. 10.2 时：作业 4 到达，等待调度
6. 11.2 时：作业 2 完成，开始运行作业 3
7. 11.7 时：作业 3 完成，开始运行作业 4
8. 12.0 时：作业 4 完成

平均周转时间 $= \frac{(10-8)+(11.2-9)+(11.7-9.5)+(12-10.2)}{4} = 2.05$（时）

② **最短作业优先（SJF）：**

1. 8.0 时：作业 1 到达，开始运行
2. 9.0 时：作业 2 到达，等待调度
3. 9.5 时：作业 3 到达，等待调度
4. 10.0 时：作业 1 完成，由于 $0.5 < 1.2$，故开始运行作业 3
5. 10.2 时：作业 4 到达，等待调度
6. 10.5 时：作业 3 完成，由于 $0.3 < 1.2$，故开始运行作业 4
7. 10.8 时：作业 4 完成，开始运行作业 2
8. 12.0 时：作业 2 完成

平均周转时间 $= \frac{(10-8)+(12-9)+(10.5-9.5)+(10.8-10.2)}{4} = 1.65$（时）

③ **响应比高者优先（HRRN）：**

1. 8.0 时：作业 1 到达，开始运行
2. 9.0 时：作业 2 到达，等待调度
3. 9.5 时：作业 3 到达，等待调度
4. 10.0 时：作业 1 完成，计算响应比：
   - $\mathrm{RP}_2 = 1 + \frac{10-9}{1.2} = \frac{11}{6}$
   - $\mathrm{RP}_3 = 1 + \frac{10-9.5}{0.5} = 2$
   - 由于 $2 > \frac{11}{6}$，故开始运行作业 3
5. 10.2 时：作业 4 到达，等待调度
6. 10.5 时：作业 3 完成，计算响应比：
   - $\mathrm{RP}_2 = 1 + \frac{10.5-9}{1.2} = 2.25$
   - $\mathrm{RP}_4 = 1 + \frac{10.5-10.2}{0.3} = 2$
   - 由于 $2.25 > 2$，故开始运行作业 2
7. 11.7 时：作业 2 完成，开始运行作业 4
8. 12.0 时：作业 4 完成

平均周转时间 $= \frac{(10-8)+(11.7-9)+(10.5-9.5)+(12-10.2)}{4} = 1.875$（时）

2-13 有五个批处理作业 A、B、C、D、E，它们几乎同时到达计算中心。它们估计运行时间分别为 10、6、2、4、8 分钟。它们的优先级分别为 3、5、2、1、4，其中 5 为最高优先级。这些作业都是纯计算型作业。一次只运行一个作业，直到它完成。对于下面列出的调度算法，忽略调度本身的开销，计算作业进程的平均周转时间。

(1) 轮转法（时间片为 2 分钟，按 A、B、C、D、E 的顺序轮转）。

(2) 优先级调度法。

答：(1) 轮转法

① 第 1 次轮转：作业 C 完成，周转时间 = $2 \times 3 = 6$（分钟）。轮转总用时 10 分钟。

② 第 2 次轮转：作业 D 完成，周转时间 = $10 + 2 \times 3 = 16$（分钟）。轮转总用时 8 分钟。

③ 第 3 次轮转：作业 B 完成，周转时间 = $10 + 8 + 2 \times 2 = 22$（分钟）。轮转总用时 6 分钟。

④ 第 4 次轮转：作业 E 完成，周转时间 = $10 + 8 + 6 + 2 \times 2 = 28$（分钟）。轮转总用时 4 分钟。

⑤ 第 5 次轮转：作业 A 完成，周转时间 = $10 + 8 + 6 + 4 + 2 \times 1 = 30$（分钟）。

平均周转时间 = $\frac{6+16+22+28+30}{5} = 20.4$（分钟）。

(2) 优先级调度法

运行顺序：B → E → A → C → D

各作业周转时间：

A: $6 + 8 + 10 = 24$（分钟）

B: 6（分钟）

C: $6 + 8 + 10 + 2 = 26$（分钟）

D: $6 + 8 + 10 + 2 + 4 = 30$（分钟）

E: $6 + 8 = 14$（分钟）

平均周转时间 = $\frac{24+6+26+30+14}{5} = 20$（分钟）。

3-7 设系统有$n + 1$个进程，其中有n个发送进程和 1 个接收进程。A_1、A_2、…、A_n通过一个单缓冲区分别不断地向进程B发消息，B不断地取走缓冲区中的消息，而且必须取走发来的每一个消息。刚开始时，缓冲区为空。试用 P、V 操作正确实现进程之间的同步。

答：该问题可以一般化为生产者-消费者问题。n 个发送进程为生产者，接收进程 B 为消费者。它们共享单缓冲区。

（1）定义信号量与缓冲区：

```
semaphore mutex = 1, empty = 1, full = 0;
message buffer, x, y;
```

（2）定义发送进程 A_i $(i = 1, 2, ..., n)$操作：

```
while (true) {
    produce_message(x);
    P(empty);
    P(mutex);
    buffer = x;
    V(mutex);
    V(full);
}
```

（3）定义接收进程 B 操作：

```
while (true) {
    P(full);
    P(mutex);
    y = buffer;
    V(mutex);
    V(empty);
    consume_message(y);
}
```

其中 `mutex` 用于实现互斥访问，`empty` 和 `full` 用于实现同步控制。

3-8 有一容量为100的循环缓冲区，有多个并发执行进程通过该缓冲区进行通信。为了正确地管理缓冲区，系统设置了两个读写指针分别为OUT、IN。IN和OUT的值如何反映缓冲区为空还是满的情况?

答： 循环缓冲区使用 IN 指向下一个可写入位置，OUT 指向下一个可读取位置。我们可以使用以下两式来判断缓冲区为空还是满：

- **缓冲区为空**：$\text{IN} == \text{OUT}$
- **缓冲区为满**：$(\text{IN} + 1) \bmod 100 == \text{OUT}$

3-9 有一阅览室，共有100个座位。为了很好地利用它，读者进入时必须先在登记表上进行登记，该表表目设有座位号和读者姓名，离开时再将其登记项擦除。试问：

(1) 为描述读者的动作，应编写几个程序？应设几个进程？它们之间的关系是什么?

(2) 试用 P、V 操作描述读者之间的同步算法。

答： **(1)** 应编写**一个**程序 (所有读者行为逻辑相同)，设置**多个**进程（每个读者对应一个进程）。一方面，它们之间是**同步关系**，共享阅览室座位和登记表资源；另一方面，它们之间是**互斥关系**，需要**互斥访问**登记表。

(2) P、V 操作描述如下：

定义信号量：

```
semaphore seats = 100;
semaphore mutex = 1;
```

读者进程：

```
P(seats);
P(mutex);
在登记表上登记();
V(mutex);
阅读();
P(mutex);
擦除登记();
V(mutex);
V(seats);
```

其中 `seats` 控制座位数量，`mutex` 保证对登记表的互斥访问。

3-14 假定系统有N个进程共享M个同类资源，规定每个进程至少申请一个资源，每个进程的最大需求不超过M，所有进程的需求总和小于$M+N$。为什么在这种情况下也决不会发生死锁？试证明。

答： 使用反证法证明。

设每个进程的最大需求为 p_i，由题意知 $\sum_{i=1}^{N} p_i < M+N$ ①。

假设发生死锁，则所有进程都处于阻塞态，没有任一进程完成。因为倘若存在某个进程完成了，那么剩余的资源会更多，更不可能发生死锁。因此只有可能刚开始便发生了死锁，此时所有进程都处于阻塞态。设各进程的实际资源分配数为 q_i，则：

- $\sum_{i=1}^{N} q_i = M$ ②（所有资源都分配完了）
- $q_i < p_i\ (1 \le i \le N)$ ③（每个进程都还需要资源，所以被阻塞）

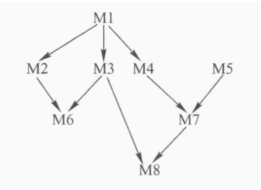
由③得：$q_i + 1 \le p_i\ (1 \le i \le N)$ ④

由④得：$\sum_{i=1}^{N} p_i \ge \sum_{i=1}^{N} (q_i + 1) = \sum_{i=1}^{N} q_i + N$ ⑤

将②代入⑤得：$\sum_{i=1}^{N} p_i \ge M+N$ ⑥，与①矛盾，假设不成立。

因此，在这种情况下决不会发生死锁。

3-15 设有8个进程M_1、M_2、…、M_8，它们有如右图所示的优先关系，试用 P、V 操作实现这些进程间的同步。



答：定义信号量：

```
semaphore a1 = 0, a2 = 0, a3 = 0, a4 = 0, a5 = 0, a7 = 0;
```

各进程操作：

```
M1() {
    执行M1();
    V(a1); V(a1); V(a1);  // 重复执行3次是因为后面有3次P(a1)，每次都会-1
}

M2() {
    P(a1);
    执行M2();
    V(a2);
}

M3() {
    P(a1);
    执行M3();
    V(a3); V(a3);
}

M4() {
    P(a1);
    执行M4();
    V(a4);
}

M5() {
    执行M5();
    V(a5);
}

M6() {
    P(a2); P(a3);
    执行M6();
}

M7() {
    P(a4); P(a5);
    执行M7();
    V(a7);
}

M8() {
    P(a3); P(a7);
    执行M8();
}
```

3-16 设有5个哲学家，共享一张放有5把椅子的圆桌。每人分得1把椅子。但是，桌子上总共只有5把叉子。哲学家们在肚子饥饿时才试图分两次从两边捡起2把叉子就餐。条件：

a. 每人只有拿到2把叉子时，哲学家才能吃饭。

b. 如果叉子已在他人手上，则该哲学家必须等到他人吃完之后才能拿到叉子。

c. 任性的哲学家在自己未拿到2把叉子吃饭之前，决不放下自己手中的叉子。

试问：

(1) 什么情况下5个哲学家全部吃不上饭?

(2) 描述一种避免有人饿死（永远拿不到2个叉子）的算法。

答： **(1)** 当每个哲学家都拿起1把叉子时，5把叉子全部被占用，但没有人能拿到2把叉子，所有哲学家都处于等待状态，形成死锁，导致全部吃不上饭。

(2) 使用限制同时就餐人数的方法避免死锁：

定义信号量：

```
semaphore fork[5] = {1, 1, 1, 1, 1};
semaphore room = 4;  // 最多允许4个哲学家同时就餐
```

哲学家i ($i = 0, 1, 2, 3, 4$)的操作：

```
philosopher_i() {
    while (true) {
        P(room);
        P(fork[i]);
        P(fork[(i+1) mod 5]);
        就餐();
        V(fork[(i+1) mod 5]);
        V(fork[i]);
        V(room);
    }
}
```

3-17 在本章前面的读者与写者问题中，是用读者优先解决问题的。试分别用读者与写者公平竞争（无写者时，读者仍遵循多个读者可以同时读）、写者优先的算法解决这个问题。

答：(1) 读者与写者公平竞争算法：

定义信号量：

```
semaphore rmutex = 1, wmutex = 1;
semaphore queue = 1;  // 公平竞争的信号量
int readcount = 0;
```

读者进程：

```
P(queue);
P(rmutex);
if (readcount == 0) P(wmutex);
readcount++;
V(rmutex);
V(queue);
读文件();
P(rmutex);
readcount--;
if (readcount == 0) V(wmutex);
V(rmutex);
```

写者进程：

```
P(queue);
P(wmutex);
写文件();
V(wmutex);
V(queue);
```

通过 `queue` 信号量确保读者和写者按到达顺序竞争，实现公平竞争。

(2) 写者优先算法：

定义信号量：

```
semaphore rmutex = 1, wmutex = 1;
semaphore resource = 1;  // 保护文件资源
semaphore r = 1;  // 阻止新读者在有写者等待时进入
int readcount = 0, writecount = 0;
```

读者进程：

```
P(r);
P(rmutex);
if (readcount == 0) P(resource);
readcount++;
V(rmutex);
V(r);
读文件();
P(rmutex);
readcount--;
if (readcount == 0) V(resource);
V(rmutex);
```

写者进程：

```
P(wmutex);
writecount++;
if (writecount == 1) P(r);
V(wmutex);
P(resource);
写文件();
V(resource);
P(wmutex);
writecount--;
if (writecount == 0) V(r);
V(wmutex);
```

3-18 有一个理发师、一把理发椅和n把供等候理发的顾客坐的椅子。如果没有顾客，则理发师便坐在椅子上睡觉，当一个顾客到来时，必须唤醒理发师，请求理发；如果理发师正在理发时，又有顾客来到，只要有空椅子，他就坐下来等待，如果没有空椅子，他就离开。请为理发师和顾客各编写一段程序来描述他们的同步过程。

答：定义信号量：

```
semaphore customers = 0;  // 等待理发的顾客数量
semaphore barber = 0;     // 理发师是否准备好
semaphore mutex = 1;      // 保护等待室访问
int waiting = 0;          // 等待理发的顾客数量。这个变量用于与 n 比较，
                          // 和信号量作用有所不同，虽然理论上值是一样的
```

理发师进程：

```
while (true) {
    P(customers);       // 等待顾客，如果没有顾客则睡觉
    P(mutex);
    waiting--;
    V(barber);          // 通知顾客理发师准备好
    V(mutex);
    理发();
}
```

顾客进程：

```
P(mutex);
if (waiting < n) {
    waiting++;
    V(customers);       // 通知理发师有顾客
    V(mutex);
    P(barber);          // 等待理发师准备好
```

```
7       理发();
8   } else {
9       V(mutex);           // 没有空椅子，离开
10  }
```

3-19 假定系统中只有一类资源，进程一次只申请一个单位的资源，且进程申请的资源数不会超过系统拥有的资源总数。假定进程申请的资源总数为 2，且系统资源总数如下，问下列哪些情况会发生死锁?

情况	进程数	系统资源总数	情况	进程数	系统资源总数
(1)	1	2	(4)	3	3
(2)	2	2	(5)	3	5
(3)	2	3	(6)	4	5

答： 每个进程最多需要2个资源。死锁发生的条件是：所有进程都持有部分资源但无法获得全部所需资源。

设进程数为N，系统资源总数为M。在最坏情况下，每个进程都获得了1个资源，此时：

- 已分配资源数 = N
- 剩余资源数 = $M - N$
- 每个进程还需要1个资源

若$M - N < 1$，即$M \leq N$，则剩余资源不足以满足任何进程的需求，可能发生死锁。

分析各情况：

(1) $N = 1$，$M = 2$：$M > N$，不会死锁。

(2) $N = 2$，$M = 2$：$M = N$，可能死锁。

(3) $N = 2$，$M = 3$：$M > N$，不会死锁。

(4) $N = 3$，$M = 3$：$M = N$，可能死锁。

(5) $N = 3$，$M = 5$：$M > N$，不会死锁。

(6) $N = 4$，$M = 5$：$M > N$，不会死锁。

因此，会发生死锁的情况是：**(2)**、**(4)**。

3-20 某一系统有10个同类资源，有3个进程P_1、P_2、P_3，它们对资源的最大需求量分别为8、6、7。它们的资源使用序列和数量如下图所示。

(1) 试给出采用银行家算法分配资源时，进行第 5 次分配后各进程的状态及各进程占用资源情况。

(2) 在以后的申请中，哪次的申请可以得到最先满足？给出一个进程完成序列。

次序	进程	申请量	次序	进程	申请量
1	P1	3	5	P2	2
2	P2	2	6	P1	3
3	P3	4	7	P3	4
4	P1	2	8	P2	2

答： 初始状态：可用资源 = 10，各进程已分配 = 0，各进程还需 = 最大需求。

第 1 次分配： P1 申请3

- 检查申请量：$3 \leq 10$（可用），$3 \leq 8$（最大需求）
- 假设分配后：P1 已分配3，还需5；Available = 7
- 寻找安全序列：
  - P2：Need=6 $\leq$ Available=7，可满足，完成后 Available=$7+0=7$（P2 未分配）
  - P3：Need=7 $\leq$ Available=7，可满足，完成后 Available=$7+0=7$（P3 未分配）
  - P1：Need=5 $\leq$ Available=7，可满足，完成后 Available=$7+3=10$
  - 安全序列：P2 → P3 → P1，系统安全，**允许分配**

第 2 次分配： P2 申请2

- 检查申请量：$2 \leq 7$（可用），$2 \leq 6$（最大需求）
- 假设分配后：P1 已分配3，P2 已分配2，P3 已分配0；Available = 5
- 寻找安全序列：
  - P2：Need=4 $\leq$ Available=5，可满足，完成后 Available=$5+2=7$
  - P3：Need=7 $\leq$ Available=7，可满足，完成后 Available=$7+0=7$（P3 未分配）
  - P1：Need=5 $\leq$ Available=7，可满足，完成后 Available=$7+3=10$
  - 安全序列：P2 → P3 → P1，系统安全，**允许分配**

第 3 次分配： P3 申请4

- 检查申请量：$4 \leq 5$（可用），$4 \leq 7$（最大需求）
- 假设分配后：P3 已分配4，还需3；可用资源 = 1
- 寻找安全序列：
  - P1：Need=5 $>$ Available=1，不满足
  - P2：Need=4 $>$ Available=1，不满足
  - P3：Need=3 $>$ Available=1，不满足

- 无法找到安全序列，系统不安全，**拒绝分配**

第 4 次分配： P1 申请2

- 检查申请量：$2 \leq 5$（可用），$2 \leq 5$（还需）
- 假设分配后：P1 已分配5，P2 已分配2，P3 已分配0；Available = 3
- 寻找安全序列：
  - P1：Need=3 ≤ Available=3，可满足，完成后 Available=3 + 5 = 8
  - P2：Need=4 ≤ Available=8，可满足，完成后 Available=8 + 2 = 10
  - P3：Need=7 ≤ Available=10，可满足，完成后 Available=10 + 0 = 10
  - 安全序列：P1 → P2 → P3，系统安全，**允许分配**

第 5 次分配： P2 申请2

- 检查申请量：$2 \leq 3$（可用），$2 \leq 4$（还需）
- 假设分配后：P2 已分配4，还需2；可用资源 = 1
- 寻找安全序列：
  - P1：Need=3 > Available=1，不满足
  - P2：Need=2 > Available=1，不满足
  - P3：Need=7 > Available=1，不满足
  - 无法找到安全序列，系统不安全，**拒绝分配**

(1) 第 5 次分配后各进程状态：

进程	状态	已分配	还需	最大需求
P1	未完成	5	3	8
P2	未完成	2	4	6
P3	未完成	0	7	7

可用资源 = 3。

(2) 后续申请分析：

当前状态：Available = 3，P1 已分配5还需3，P2 已分配2还需4，P3 已分配0还需7。

第 6 次申请： P1 申请3

- 检查：$3 \leq 3$（可用），$3 \leq 3$（还需）
- 假设分配后：P1 已分配8，P2 已分配2，P3 已分配0；Available = 0
- 寻找安全序列：
  - P1：Need=0 ≤ Available=0，可满足，完成后 Available=0 + 8 = 8
  - P2：Need=4 ≤ Available=8，可满足，完成后 Available=8 + 2 = 10
  - P3：Need=7 ≤ Available=10，可满足，完成后 Available=10 + 0 = 10
  - 安全序列：P1 → P2 → P3，**可以满足**

因此，**第 6 次申请（P1 申请3）** 可以最先得到满足，此时完成序列为：P1 → P2 → P3。

3-21 考虑某一系统，它有4类资源R_1、R_2、R_3、R_4，有5个并发进程P_0、P_1、P_2、P_3、P_4。请按照银行家算法回答下列问题：

(1) 各进程的最大资源请求和已分配的资源矩阵及系统当前的剩余资源向量如下图所示，计算各进程的剩余资源请求向量组成的矩阵。

(2) 系统当前处于安全状态吗?

(3) 当进程P_2申请的资源分别为$(1,0,0,1)$时，系统能立即满足吗?

进程	分配向量				最大需求量			
	R1	R2	R3	R4	R1	R2	R3	R4
P0	0	0	1	2	0	0	1	2
P1	1	0	0	0	1	7	5	0
P2	1	3	5	4	2	3	5	6
P3	0	6	3	2	0	6	5	2
P4	0	0	1	4	0	6	5	6

R1	R2	R3	R4
1	5	0	2

当前剩余资源向量

答：(1) 计算剩余资源请求向量（Need 矩阵）：

Need = Max - Allocation

进程	R1	R2	R3	R4
P0	$0-0=0$	$0-0=0$	$1-1=0$	$2-2=0$
P1	$1-1=0$	$7-0=7$	$5-0=5$	$0-0=0$
P2	$2-1=1$	$3-3=0$	$5-5=0$	$6-4=2$
P3	$0-0=0$	$6-6=0$	$5-3=2$	$2-2=0$
P4	$0-0=0$	$6-0=6$	$5-1=4$	$6-4=2$

(2) 检查系统是否处于安全状态：

当前可用资源向量：Available = $(1,5,0,2)$

寻找安全序列：

- P0：Need$(0,0,0,0) \leq$ Available$(1,5,0,2)$，可满足。完成后释放$(0,0,1,2)$，Available = $(1,5,1,4)$
- P2：Need$(1,0,0,2) \leq$ Available$(1,5,1,4)$，可满足。完成后释放$(1,3,5,4)$，Available = $(2,8,6,8)$
- P3：Need$(0,0,2,0) \leq$ Available$(2,8,6,8)$，可满足。完成后释放$(0,6,3,2)$，Available = $(2,14,9,10)$
- P1：Need$(0,7,5,0) \leq$ Available$(2,14,9,10)$，可满足。完成后释放$(1,0,0,0)$，Available = $(3,14,9,10)$
- P4：Need$(0,6,4,2) \leq$ Available$(3,14,9,10)$，可满足。

存在安全序列：P0 → P2 → P3 → P1 → P4，系统处于**安全状态**。

(3) 当 P2 申请$(1,0,0,1)$时：

① 检查申请是否超过 Need：$(1,0,0,1) \leq (1,0,0,2)$，满足。

② 检查申请是否超过 Available：$(1,0,0,1) \leq (1,5,0,2)$，满足。

③ 假设分配，更新状态：

- Available = $(1,5,0,2) - (1,0,0,1) = (0,5,0,1)$
- P2 Allocation = $(1,3,5,4) + (1,0,0,1) = (2,3,5,5)$
- P2 Need = $(1,0,0,2) - (1,0,0,1) = (0,0,0,1)$

④ 检查分配后是否安全：

- P0：Need$(0,0,0,0) \leq$ Available$(0,5,0,1)$，可满足。完成后 Available = $(0,5,1,3)$
- P2：Need$(0,0,0,1) \leq$ Available$(0,5,1,3)$，可满足。完成后 Available = $(2,8,6,8)$
- P3：Need$(0,0,2,0) \leq$ Available$(2,8,6,8)$，可满足。完成后 Available = $(2,14,9,10)$
- P1：Need$(0,7,5,0) \leq$ Available$(2,14,9,10)$，可满足。完成后 Available = $(3,14,9,10)$
- P4：Need$(0,6,4,2) \leq$ Available$(3,14,9,10)$，可满足。

存在安全序列：P0 → P2 → P3 → P1 → P4，分配后仍处于安全状态。

因此，**系统能立即满足 P2 的申请**。