操作系统
综合题汇总

2018.12.18

duanwh@126.com

目录

1 作业（进程）调度

【例题 1】教材例题 单道系统

表 3-1 先来先服务调度算法计算结果

作　　业	进入时间	估计运行时间（分钟）	开始时间	结束时间	周转时间（分钟）	带权周转时间
JOB1	8：00	120	8：00	10：00	120	1
JOB2	8：50	50	10：00	10：50	120	2.4
JOB3	9：00	10	10：50	11：00	120	12
JOB4	9：50	20	11：00	11：20	90	4.5
作业平均周转时间 T = 112.5 分钟 作业带权平均周转时间 W = 4.975					450	19.9

表 3-2 最短作业优先作业算法计算结果

作　　业	进入时间	估计运行时间（分钟）	开始时间	结束时间	周转时间（分钟）	带权周转时间
JOB1	8：00	120	8：00	10：00	120	1
JOB2	8：50	50	10：30	11：20	150	3
JOB3	9：00	10	10：00	10：10	70	7
JOB4	9：50	20	10：10	10：30	40	2
作业平均周转时间 T = 95 分钟 作业带权平均周转时间 W = 3.25					380	13

表 3-3 最高响应比优先作业算法计算结果

作　　业	进入时间	估计运行时间（分钟）	开始时间	结束时间	周转时间（分钟）	带权周转时间
JOB1	8：00	120	8：00	10：00	120	1
JOB2	8：50	50	10：10	11：00	130	2.6
JOB3	9：00	10	10：00	10：10	70	7
JOB4	9：50	20	11：00	11：20	90	4.5
作业平均周转时间 T = 102.5 分钟 作业带权平均周转时间 W = 3.8					350	15.1

【例题 2】教材例题 多道系统

表 3-4 多道程序环境下的作业调度算法计算结果

作　　业	进 入 时 间	估计运行时间（分钟）	开 始 时 间	结束时间	周转时间（分钟）	带权周转时间
JOB1	10：00	30	10：00	11：05	65	4.167
JOB2	10：05	20	10：05	10：25	20	1
JOB3	10：10	5	10：25	10：30	20	4
JOB4	10：20	10	10：30	10：40	20	2

作业平均周转时间 T = 31.25 作业带权平均周转时间 W = 2.79	125	11.167

【例题 3】课后习题 11

11. 单道批处理环境下有 5 个作业，各个作业进入系统的时间和估计运行时间如题表 3-11（a）所示。

作　业	进入系统时间	估计运行时间/分钟
1	8:00	40
2	8:20	30
3	8:30	12
4	9:00	18
5	9:10	5

（1）如果应用先来先服务的作业调度算法，试将表 3-11（b）填写完整。

作　业	进入系统时间	估计运行时间（分钟）	结束时间	周转时间（分钟）	带权周转时间
1	8:00	40	8:40	40	1
2	8:20	30	9:10	50	1.67
3	8:30	12	9:22	52	4.33
4	9:00	18	9:40	40	2.22
5	9:10	5	9:45	35	7
作业平均周转时间 T=43.4 分钟					3.24

（2）如果应用最短作业优先的作业调度算法，试将表 3-11（c）填写完整。

作　业	进入系统时间	估计运行时间（分钟）	结束时间	周转时间（分钟）	带权周转时间
1	8:00	40	8:40	40	1
2	8:20	30	9:22	62	2.07
3	8:30	12	8:52	22	1.83
4	9:00	18	9:45	45	2.5
5	9:10	5	9:27	17	3.4
作业平均周转时间 T=37.2 分钟					2.16

（3）如果应用高相应比优先的作业调度算法，试将表 3-11（d）填写完整。

作　业	进入系统时间	估计运行时间（分钟）	结束时间	周转时间（分钟）	带权周转时间
1	8:00	40	8:40	40	1
2	8:20	30	9:22	62	2.07
3	8:30	12	8:52	22	1.83
4	9:00	18	9:45	45	2.5
5	9:10	5	9:27	17	3.4

作业平均周转时间 T=37.2 分钟	2.16

注：

（1）作业的周转时间 = 作业的完成时间 – 进入系统的时间。

（2）带权周转时间 = 作业的周转时间/作业运行时间，带权周转时间是一个比值，没有单位。

（3）高相应比优先调度算法中：

1）8:40 时，作业 2 和作业 3 到达

作业 2 响应比 = 1 + (20/30)

作业 3 响应比 = 1 + (10/12)

两者相比选择作业 3 占用 CPU

2）8:52 时，只有作业 2 到达，选择作业 2 占用 CPU

3）9:22 时，作业 4 和作业 5 到达

作业 4 响应比 = 1 + (22/18)

作业 5 响应比 = 1 + (12/5)

两者相比选择作业 5 占用 CPU

4）9:27 时，只有作业 4 了，选择作业 4 占用 CPU

【例题 4】课后习题 12

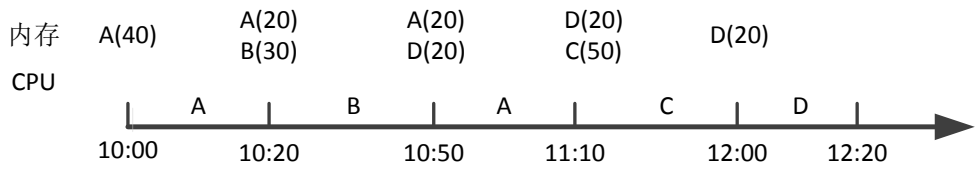
12 有一个具有两道作业的批处理系统，作业调度采用短作业优先的非抢占式调度算法，进程调度采用以优先数为基础的抢占式调度算法，在题表 3-12 所示的作业序列中，作业优先数越小优先级越高。

题表 3-12

作业名	到达时间	运行时间	优先数
A	10：00	40 分	5
B	10：20	30 分	3
C	10：30	50 分	4
D	10：50	20 分	6

（1）列出所有作业进入内存时间及结束时间。

（2）计算平均周转时间。



作业名	到达时间	运行时间	优先数	进入内存	完成时间	周转时间（分钟）	带权周转时间
A	10：00	40 分	5	10:00	11:10	70	1.75
B	10：20	30 分	3	10:20	10:50	30	1
C	10：30	50 分	4	11:10	12:00	90	1.8
D	10：50	20 分	6	10:50	12:20	90	4.5
平均周转时间						70	2.26

注意：

（1）题目中有两级调度，分别是高级调度和低级调度。高级调度按照短作业优先的规则选择到达系统的作业进入内存。低级调度是在进入内存的作业中按照优先数小优先获得 CPU 执行。

（2）题目中的到达时间准确的说应该是到达外存的后备队列。

（3）高级调度只有在后备队列中有两个作业时才起作用。本题中，高级调度只在 10:50 时刻起作用了。（此时，作业 B 运行结束，内存中只有作业 A，后备队列中有作业 C 和作业 D 两道作业，按照短作业优先的规则将作业 D 调入内存）

低级调度是在内中有两道作业（进程）时才起作用。本题中，低级调度在 10:20，10:50，11:10 共三个时刻起作用。

（4）作业的周转时间是完成时间减去到达系统的时间。准确的讲，周转时间应该是完成时间减去到达外存后备队列的时间，而不是减去进入内存的时间。（★）

2 信号量问题

1. 解决最简单的互斥问题 P79
2. 解决最简单的同步问题 P80（缓冲区为 1 的生产者消费者问题）
3. 哲学家吃通心面问题 P81（申请两个互斥资源）
4. 苹果桔子问题 P83（两种产品的生产者消费者问题）
5. 经典的生产者消费者问题 P82（缓冲区为 n）
6. 读者写者问题 P83（互斥问题）
7. 捡黑白子问题 P110（类似最简单的同步问题）
8. 司机售票员问题 P72（类似最简单的同步问题，捡黑白子问题）
9. 窄桥问题（类似读者写者问题）
10. 作业执行先后

【例题 1】1. 解决最简单的互斥问题

```
semaphore mutex;
mutex = 1;

void Pi()
{
        …
        P(mutex);
        使用打印机;
        V(mutex);
        …
}
```

【例题 2】2. 解决最简单的同步问题

（缓冲区为 1 的生产者消费者问题）

```
semaphore empty, full;
empty = 1;
full = 0;
```

```
void Producer()
{
        while(1)
        {
            生产一个产品;
            P(empty);
            向缓冲区中放产品;
            V(full);
        }
}
```

```
void Consumer()
{
        while(1)
        {
            P(empty);
            向缓冲区中放产品;
            V(full);
            消费一个产品;
        }
}
```

【例题 3】3. 哲学家吃通心面问题

（申请两个互斥资源）

```
semaphore fork[0..4];
fork[0] = fork[1] = fork[2] = fork[3] = fork[4] = 1;

void Ph_i () // i = 0~4，五个进程
{
        思考;
        P( fork[i] );
        P( fork[(i+1) % 4] );
        吃面条;
        V( fork[(i+1) % 4] )
        V( fork[i] );
}
```

这个解法会导致死锁

解决死锁的方法 1：限定并发进程的数目，可以增加一个信号量，初始值设定为 4。

解决死锁的方法 2：规定奇数的哲学家，先申请左面的叉子；偶数的哲学家，先申请右面的筷子。

解决死锁的方法 1：限定并发进程的数目，可以增加一个信号量，初始值设定为 4。

```
semaphore fork[0..4];
fork[0] = fork[1] = fork[2] = fork[3] = fork[4] = 1;

semaphore m;
m = 4;
void Ph_i () // i = 0~4
{
        思考;
        P(m);
        P( fork[i] );
        P( fork[(i+1) % 4] );
        吃面条;
        V( fork[(i+1) % 4] )
        V( fork[i] );
        V(m);
}
```

解决死锁的方法 2：规定奇数的哲学家，先申请左面的叉子；偶数的哲学家，先申请右面的筷子。

```
semaphore fork[0..4];
fork[0] = fork[1] = fork[2] = fork[3] = fork[4] = 1;
```

```
void Ph_i () // i = 0~4
{
        思考;
        if ( i%2 == 1)          //奇数的哲学家
        {
            P( fork[i] );                       //左面的叉子
            P( fork[(i+1) % 4] );       //右面的叉子
        }
        else                    //偶数的哲学家
        {
            P( fork[(i+1) % 4] );           //右面的叉子
            P( fork[i] );                   //左面的叉子
        }
        吃面条;
        V( fork[(i+1) % 4] )
        V( fork[i] );
}
```

注：
释放叉子的先后顺序无关紧要。

【例题 4】4. 苹果桔子问题

（两种产品的生产者消费者问题）
注意：
爸爸放苹果，女儿吃苹果
妈妈放桔子，儿子吃桔子
爸爸和妈妈是生产者
女儿和儿子是消费者
产品有两种，一个是苹果，一个是桔子。

```
semaphore empty, apple, orange;
empty = 1;
apple = orange = 0;
```

void father()
{
  while(1)
  {
  削苹果;
  P(empty);
  放苹果;
  V(apple);
  }
	void daughter()
	{
	  while(1)
	  {
	  P(apple);
	  取苹果;
	  V(empty);
	  吃苹果;
	  }

}
void mother()
{
while(1)
{
剥桔子;
P(empty);
放桔子;
V(apple);
}
}
	}
	void son()
	{
	while(1)
	{
	P(orange);
	取桔子;
	V(empty);
	吃桔子;
	}
	}

问题的修改，如果妈妈出差了，只有爸爸一个人在家，爸爸有时削苹果，有时剥桔子。

semaphore empty, apple, orange;
empty = 1;
apple = orange = 0;

void father()
{
while(1)
{
削苹果或剥桔子;
P(empty);
放苹果或放桔子;
if (放的是苹果) V(apple);
else V(orange);
}
}
	void daughter()
	{
	while(1)
	{
	P(apple);
	取苹果;
	V(empty);
	吃苹果;
	}
	}

	void son()
	{
	while(1)
	{
	P(orange);
	取桔子;
	V(empty);
	吃桔子;
	}
	}

【例题 5】5. 经典的生产者消费者问题

（缓冲区为 n）

```
semaphore mutex, empty, full;
mutex = 1;
empty = K;
full = 0;
```

```
void Producer()
{
    while (1)
    {
        生产一个产品;
        P(empty);
        P(mutex);
        放产品;
        V(mutex);
        V(full);
    }
}
```

```
void Consumer()
{
    while (1)
    {
        P(full);
        P(mutex);
        取产品;
        V(empty);
        V(full);
        消费产品;
    }
}
```

对取和放两个操作进行细化，使用到了一维数组，将一维数组看成是头尾相连的环形队列。

```
semaphore mutex, empty, full;
mutex = 1;
empty = K;
full = 0;

item B[K];
int in, out;
in = 0;
out = 0;
```

```
void Producer()
{
    item product;
    while (1)
    {
        生产一个产品 product;
        P(empty);
        P(mutex);
        B[in] = product;
        in = (in + 1) % K;
        V(mutex);
        V(full);
    }
```

```
void Consumer()
{
    item product;
    while (1)
    {
        P(full);
        P(mutex);
        product = B[out];
        out = (out + 1) % K;
        V(empty);
        V(full);
        消费产品 product;
    }
```

}	}

【例题 6】6. 读者写者问题（互斥问题）

```
int rc;                    //rc 是计数器，是整型变量，不是信号量
rc = 0;
semaphore W;      //信号量，保证文件操作的互斥
semaphore R;      //信号量，保证 rc 计数器操作的互斥

W = 1;
R = 1;
```

void Reader()
{
P(R);
rc = rc + 1;
if (rc == 1) P(W);
V(R);
 	
读文件;
 	
P(R);
rc = rc - 1;
if (rc == 0) V(W);
V(R);
}
	void Writer()
	{
	P(W);
	
	写文件;
	
	V(W);
	}

7. 捡黑白子问题（类似最简单的同步问题）

修改最简单的同步问题就可以得到解决方案。

```
int K;
K = 180;              //设定黑白子各有 180 个

semaphore black, white;
black = 1;            //设定捡黑子的进程先运行
white = 0;
```

void PickBlack()
{
int i;
i = K;
 	
while(i > 0)
{
P(black);
捡黑子;
V(white);
	void PickWhite ()
	{
	int i;
	i = K;
	
	while(i > 0)
	{
	P(white);
	捡白子;
	V(black);

i--; } }	i--; } }

【例题 8】8. 司机售票员问题

（类似最简单的同步问题，捡黑白子问题）

修改最简单的同步问题就可以得到解决方案。

semaphore start, open;

start = 1; //设定初始时车门是关着的

open = 0;

void Driver()
{
while(1)
{
P(start);
启动车辆;
行驶中
到站停车;
V(open);
}
}
	void Conductor ()
	{
	while(1)
	{
	P(open);
	开门;
	上乘客，售票;
	关门;
	V(start);
	}
	}

【例题 9】9. 窄桥问题（类似读者写者问题）

修改的是读者进程。

int scount; //南下车辆计数

int ncount; //北上车辆计数

scount = ncount = 0;

semaphore mutex; //南下北上互斥

semaphore smutex; //对 scount 操作互斥

semaphore nmutex; //对 ncount 操作互斥

mutex = smutex = nmutex = 1;

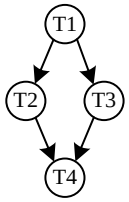
void South()
{
P(smutex);
scount = scount + 1;
if (scount == 1) P(mutex);
V(smutex);
 	
南下通过大桥;
	void North()
	{
	P(nmutex);
	ncount = ncount + 1;
	if (ncount == 1) P(mutex);
	V(nmutex);
	
	北上通过大桥;

P(smutex);
scount = scount - 1;
if (scount == 0) V(mutex);
V(smutex);
}
	P(nmutex);
	ncount = ncount - 1;
	if (ncount == 0) V(mutex);
	V(nmutex);
	}

【例题 10】10 作业执行先后

（1）每个先后关系设定一个型号量 S，初始值设定为 0
（2）先执行的程序在程序的最后执行 V 操作
（3）后执行的程序在程序的最前执行 P 操作。

设有一个作业由 4 个进程 T1, T2, T3, T4 组成，这 4 个进程必须按下图的次序运行，试用信号量操作表达 4 个进程的同步关系。



semaphore a, b, c, d; a = b = c = d = 0;	

void T1()
{
...
V(a);
V(b);
}
	void T2()
	{
	P(a);
	...
	V(c);
	}

void T3()
{
P(b);
...
V(d);
}
	void T4()
	{
	P(c);
	P(d);
	...
	}

3 银行家算法

【例题 1】课后习题 14.

14. 假定具有 5 个进程的进程集合 P＝｛P0,P1,P2,P3,P4｝，系统中有 3 类资源 A，B 和 C。其中 A 类资源有 10 个，B 类资源有 5 个，C 类资源有 7 个。假定在某时刻有如题表 4-14 所示的状态。

题表 4-14

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

试给出 Need，并说明当前系统是否处于安全状态，如果是，给出安全序列。如果不是，说明理由。

答：

	Allocation			Max			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	7	4	3	3	3	2
P1	2	0	0	3	2	2	1	2	2			
P2	3	0	2	9	0	2	6	0	0			
P3	2	1	1	2	2	2	0	1	1			
P4	0	0	2	4	3	3	4	3	1			

系统处于安全状态，可以找到一个安全序列 P1, P3, P0, P2, P4。

附加问题：如果此时 P1 发出一个请求 Request1(1, 1, 0)系统能否满足？为什么？

解答：可以分配，因为至少可以找到一个安全序列{P1, P3, P0, P2, P4}，分配后系统仍然处于安全状态。

过程如下：

（1）Request1(1, 1, 0) <= Available(3, 3, 2)

（2）Request1(1, 1, 0) <= Need(1, 2, 2)

（3）试探性分配

	Allocation			Max			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	7	4	3	~~3~~ 2	~~3~~ 2	~~2~~ 2
P1	~~2~~ 3	~~0~~ 1	~~0~~ 0	3	2	2	~~1~~ 0	~~1~~ 0	~~2~~ 2			
P2	3	0	2	9	0	2	6	0	0			
P3	2	1	1	2	2	2	0	1	1			

P4	0	0	2	4	3	3	4	3	1			

（4）找安全序列，可以找到安全序列{P1, P3, P0, P2, P4}

（5）找到一个安全序列，可以保证系统处于安全状态，所以可以分配。

Available(2, 2, 2)可以满足 P1（Available(2, 2, 2) >= Need1(0, 0, 2)）

P1 运行结束，Available = Available + Allocation1 = (2, 2, 2) + (3, 1, 0) = (5, 3, 2)

Available(5, 3, 2)可以满足 P3（Available(5, 3, 2) >= Need3(0, 1, 1)）

P3 运行结束，Available = Available + Allocation3 = (5, 3, 2) + (2, 1, 1) = (7, 4, 3)

Available(7, 4, 3)可以满足 P0（Available(7, 4, 3) >= Need0(7, 4, 3)）

P0 运行结束，Available = Available + Allocation0 = (7, 4, 3) + (0, 1, 0) = (7, 5, 3)

Available(7, 5, 3)可以满足 P2（Available(7, 5, 3) >= Need2(6, 0, 0)）

P2 运行结束，Available = Available + Allocation2 = (7, 5, 3) + (3, 0, 2) = (10, 5, 5)

Available(10, 5, 5)可以满足 P4（Available(10, 5, 5) >= Need4(4, 3, 1)）

P4 运行结束，Available = Available + Allocation4 = (10, 5, 5) + (0, 0, 2) = (10, 5, 7)

【注意】

（1）矩阵和向量

Allocation 是已分配矩阵

Max 是初始时向系统提出的最大需求矩阵

Need 是还能向系统提出的最大的需求矩阵（Need = Max – Allocation）

Available 是系统中剩余的资源向量。

（2）系统如果处于安全状态，可以给出的安全序列不一定是唯一的。只要找到一个安全序列就可以证明系统处于安全状态了。

（3）查找安全序列的方法：

Available （3 3 2）可以满足 P1（Available(3 3 2) >= Need1(1 2 2)）或者 P3（Available (3 3 2) >= Need3(0 1 1)）。

如果满足 P1，Available = Available（3 3 2） + Allocation1（2 0 0） = (5 3 2)

……

【例题 2】课后习题 18.

18. 系统有 A，B，C 和 D 共 4 种资源，在某时刻进程 P0，P1，P2，P3 和 P4 对资源的占有和需求情况如题表 4-18 所示，试解答下列问题：

题表 4-18

Process	Allocation	Claim	Available
	A B C D	A B C D	A B C D
P0	0 0 3 2	0 0 4 4	1 6 2 2
P1	1 0 0 0	2 7 5 0	
P2	1 3 5 4	3 6 10 10	
P3	0 3 3 2	0 9 8 4	
P4	0 0 1 4	0 6 6 10	

（1）系统此时处于安全状态吗？为什么？

（2）若此时 P2 发出 request1(1，2，2，2)，系统能分配资源给它吗？为什么？

答：

（1）系统处于安全状态，因为可以找到一个安全序列。{P0, P3, P1, P2, P4}

Process	Allocation	Claim	Need	Available
	A B C D	A B C D	A B C D	A B C D
P0	0 0 3 2	0 0 4 4	0 0 1 2	1 6 2 2
P1	1 0 0 0	2 7 5 0	1 7 5 0	
P2	1 3 5 4	3 6 10 10	2 3 5 6	
P3	0 3 3 2	0 9 8 4	0 6 5 2	
P4	0 0 1 4	0 6 6 10	0 6 5 6	

（2）P2 进程发出请求 Request2(1, 2, 2, 2)，系统不能分配资源给它，因为如果分配给该进程，系统会进入不安全状态。

查找安全序列过程：

a)请求要小于等于当前所能提出的最大请求，要小于等于 Available。

Reques2(1 2 2 2) <= Need2(2 3 5 6)

Reques2(1 2 2 2) <= Available(1 6 2 2)

b)试探性分配

Available = Available（1 6 2 2） – Request2（1 2 2 2） = （0 4 0 0）

Allocation2 = Allocation2(1 3 5 4) + Request2（1 2 2 2）= （2 5 7 6）

Need2 = Need2（2 3 5 6） - Request2（1 2 2 2） = （1 1 3 4）

Process	Allocation	Claim	Need	Available
	A B C D	A B C D	A B C D	A B C D
P0	0 0 3 2	0 0 4 4	0 0 1 2	~~1 6 2 2~~ 0 4 0 0
P1	1 0 0 0	2 7 5 0	1 7 5 0	
P2	~~1 3 5 4~~ 2 5 7 6	3 6 10 10	~~2 3 5 6~~ 1 1 3 4	
P3	0 3 3 2	0 9 8 4	0 6 5 2	
P4	0 0 1 4	0 6 6 10	0 6 5 6	

c）找安全序列

Available（0 4 0 0）不能满足如何一个进程的可能提出的最大请求。找不到安全序列。

注：

（1）矩阵和向量

Allocation 是已分配矩阵

Claim 是初始时向系统提出的最大需求矩阵

Need 是还能向系统提出的最大的需求矩阵（Need = Claim – Allocation）

Available 是系统中剩余的资源向量。

（2）系统如果处于安全状态，可以给出的安全序列不一定是唯一的。只要找到一个安全序列就可以证明系统处于安全状态了。

4 进程资源分配图

4.1 图表

（1）圆圈代表进程

（2）方框代表资源

（3）方框中的点代表资源数

（4）圆圈指向方框的有向边表示请求资源

（5）方框指向圆圈的有向边表示占有资源

4.2 判断是否存在死锁

（1） 不存在回路一定不死锁

（2）存在回路且资源数目均为 1，一定死锁

（3）其它情况，简化后进行分析

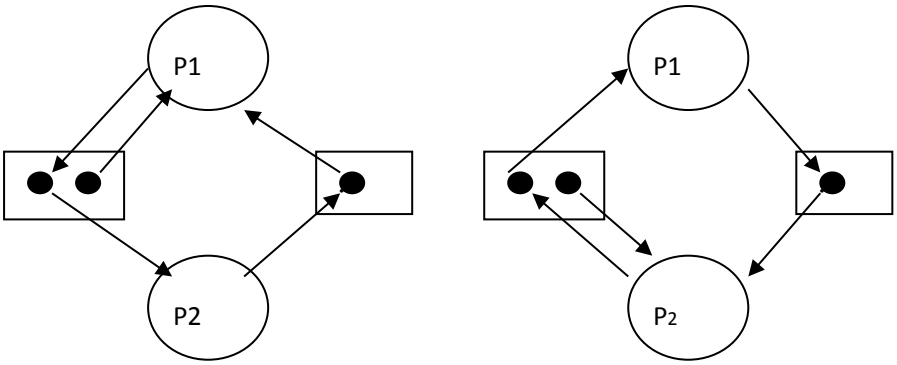
【例题 1】例题

假定某计算机系统有 R1 和 R2 两类可再使用资源（其中 R1 有两个单位，R2 有一个单位），它们被进程 P1，P2 所共享，且已知两个进程均以下列顺序使用两类资源。

→申请 R1→申请 R2→申请 R1→释放 R1→释放 R2→释放 R1→

试求出系统运行过程中可能到达的死锁点，并画出死锁点的资源分配图（或称进程－资源图）。

[解答]



5 分页式机制

5.1 无快表
（1）时间 = 访问页表的时间 + 访问数据的时间
（2）共两次访问内存的时间
5.2 有快表
（1）快表命中
访问快表的时间+访问数据的时间
（2）快表没有命中
访问页表的时间+访问数据的时间
5.3 访问页表的时间=访问数据的时间= 访问内存的时间

【例题 1】教材例题

假定访问内存的时间为 100ns，访问相联存储器的时间为 20ns，相联存储器为 32 个单元时查快表的命中率可达 90%，于是按逻辑地址进行存取的平均时间为：

$$(100+20)\times 90\%+(100+100)\times(1-90\%)=128\text{ns}$$

比两次访问内存的时间 200ns 下降了近四成。

6 地址转换

6.1 分页式地址转换
（1）逻辑地址（LA）分为两个部分
页号 p，p=LA/L
页内位移 d，d=LA%L
（2）页的大小为 L
（3）根据 p 查找页表得到对应的块号 b
（4）物理地址 PA，PA = b*L + d
6.2 分段式地址转换

【例题 1】分页式地址转换

在一分页式存储系统中，逻辑地址长度为 16 位，页面大小为 4096B，且第 0、1、2 页依次存放在物理块 5、10、11 中，分别将逻辑地址 2F6A(H)和 164F(H)转换为相应的物理地址。（物理地址写成十六进制格式）
[解答]
2F6A(H)和 164F(H)转换为相应的物理地址为 BF6A(H)和 A64F(H)。
页的大小是 4096B（2 的 12 次方），所以逻辑地址前 4 位是页号，后 12 位是页内位移。页号为 2，对应的块号是 11，页内位移保持不变，形成的物理地址 BF6A(H)

2	F	6	A
0010	1111	0110	1010
页号 2	页内位移		
块号 11			
1011	1111	0110	1010
B	F	6	A

1	6	4	F
0001	0110	0100	1111
页号 1	页内位移		
块号 10			
1010	0110	0100	1111
A	6	4	F

【例题 2】课后习题 15.

15. 某虚存的用户空间共有 24 个页面，每页 1KB，内存 16KB。假定某时刻系统为用户的第 0，1，2 和 3 页分别分配的物理块号为 5，10，4 和 7，试将虚拟地址 0A5C 和 093C 变换为物理地址。
答：

逻辑地址（H）	0A5C	093C

逻辑地址（B）	0000 1010 0101 1100	0000 1001 0101 1100
1K = 2 的 10 次方，前 6 位是页号，后 10 位是页内位移。 转换成物理地址时，页内位移不变，页号转换成块号		
页号	0000 10（10 进制是 2）	0000 10（10 进制是 2）
块号	0001 00（10 进制是 4）	0001 00（10 进制是 4）
物理地址（B）	0001 0010 0101 1100	0001 0001 0101 1100
物理地址（H）	125C	113C

【例题 3】课后习题 14.

14. 给定段表如题表 5-13 所示。

题表 5-13

段　　号	段　首　址	段　　长
0	400	600
1	1300	400
2	100	200

给定以下地址为段号和位移数：

（1）[0，430]　（2）[1，200]　（3）[2，400]　（4）[3，100]

答

（1）物理地址 = 400 + 430 = 830

（2）物理地址 = 1300 + 200 = 1500

（3）400 > 200 （段内位移大于段长），系统产生越界中断。

（4）3 >= 3（段号大于等于段表长度），系统产生越界中断。

7 页面置换算法

（1）先进先出页面置换算法
（2）最佳页面置换算法
（3）最近最久页面置换算法

【例题 1】教材例题

一个页式存储管理系统使用 FIFO，OPT 和 LRU 页面替换算法，如果一个作业的页面走向为：2，3，2，1，5，2，4，5，3，2，5 和 2。当分配给该作业的物理页框块数为 3 时，试计算访问过程中发生的缺页中断次数和缺页中断率。

最佳页面置换算法（OPT 算法）

	2	3	2	1	5	2	4	5	3	2	5	2
页 1	2	2		2	2		4			4		
页 2		3		3	3		3			2		
页 3				1	5		5			5		
是否缺页	√	√		√	√		√			√		
置换了					1		2			3		
										注 1		

缺页 6 次，缺页率为 6/12 * 100% =50%。

发生了 3 次置换，依次置换的页面是 1、2、3。

【注 1】要将 2 换入，需要换出的页面可以是 3 或者 4。如果加上限定，同等情况下优先置换出先进入内存的页面，此时置换出的页面是 3。

先进先出页面置换算法（FIFO 算法）

	2	3	2	1	5	2	4	5	3	2	5	2
页 1	2	2		2	5	5	5		3		3	3
页 2		3		3	3	2	2		2		5	5
页 3				1	1	1	4		4		4	2
是否缺页	√	√		√	√	√	√		√		√	√
置换了					2	3	1		5		2	4

缺页 9 次，缺页率为 9/12 * 100% =75%

发生了 6 次置换，依次置换的页面是 2、3、1、5、2、4。

最近最久未使用页面置换算法（LRU 算法）

	2	3	2	1	5	2	4	5	3	2	5	2
页 1	2	2		2	2		2		3	3		
页 2		3		3	5		5		5	5		
页 3				1	1		4		4	2		
是否缺页	√	√		√	√		√		√	√		
置换了					3		1		2	4		

缺页 7 次，缺页率为 7/12 * 100% =58.3%

发生了 4 次置换，依次置换的页面是 3、1、2、4。

【例题 2】习题

一个页式存储管理系统使用 FIFO，OPT 和 LRU 页面替换算法，如果一个作业的页面走向为：4，3，2，1，4，3，5，4，3，2，1 和 5。当分配给该作业的物理页框块数为 3 时，试计算访问过程中发生的缺页中断次数和缺页中断率。

最佳页面置换算法（OPT 算法）

	4	3	2	1	4	3	5	4	3	2	1	5
页 1	4	4	4	4	4	4	4	4	4	2	2	2
页 2		3	3	3	3	3	3	3	3	3	1	1
页 3			2	1	1	1	5	5	5	5	5	5
是否缺页	√	√	√	√			√			√	√	
置换了				2			1			4	3	
											注 1	

缺页 7 次，缺页率为 7/12 * 100% =58.3%。

发生了 4 次置换，依次置换的页面是 2、1、4、3。

【注 1】要将 1 换入，需要换出的页面可以是 2 或者 3。如果加上限定，同等情况下优先置换出先进入内存的页面，此时置换出的页面是 3。

先进先出页面置换算法（FIFO 算法）

	4	3	2	1	4	3	5	4	3	2	1	5
页 1	4	4	4	1	1	1	5	5	5	5	5	5
页 2		3	3	3	4	4	4	4	4	2	2	2
页 3			2	2	2	3	3	3	3	3	1	1
是否缺页	√	√	√	√	√	√	√			√	√	
置换了				4	3	2	1			4	3	

缺页 9 次，缺页率为 9/12 * 100% =75%

发生了 6 次置换，依次置换的页面是 4、3、2、1、4、3。

最近最久未使用页面置换算法（LRU 算法）

	4	3	2	1	4	3	5	4	3	2	1	5
页 1	4	4	4	1	1	1	5	5	5	2	2	2
页 2		3	3	3	4	4	4	4	4	4	1	1
页 3			2	2	2	3	3	3	3	3	3	5
是否缺页	√	√	√	√	√	√	√			√	√	√
置换了				4	3	2	1			5	4	3

缺页 10 次，缺页率为 10/12 * 100% =83.3%

发生了 7 次置换，依次置换的页面是 4、3、2、1、5、4、3。

分配 4 个物理块时：

采用先进先出页面置换算法：

	4	3	2	1	4	3	5	4	3	2	1	5
页 1	4	4	4	4			5	5	5	5	1	1
页 2		3	3	3			3	4	4	4	4	5
页 3			2	2			2	2	3	3	3	3
页 4				1			1	1	1	2	2	2
是否缺页	√	√	√	√			√	√	√	√	√	√
置换了							4	3	2	1	5	4

产生 10 次缺页。缺页率是 10/12*100% = 83.3%

发生了 6 次置换，依次置换的页面是 4、3、2、1、5、4。

【Belady 异常】同样采用先进先出页面置换算法，如果分配 3 个物理块缺页次数是 9 次，如果分配 4 个物理块缺页次数是 10 次，分配的物理块数增加了，缺页次数不降反升，这个就是 Belady 异常。

【例题 3】课后习题 12.

12. 在一个采用页式虚拟存储管理的系统中，有一用户作业，它依次要访问的字地址序列是：115，228，120，88，446，102，321，432，260 和 167，若该作业的第 0 页已经装入内存，现分配给该作业的内存共 300B，页的大小为 100B，请回答下列问题：

（1）按 FIFO 调度算法将产生____次缺页中断，依次淘汰的页号为____，缺页中断率为____；

（2）按 LRU 调度算法将产生____次缺页中断，依次淘汰的页号为____，缺页中断率为____。

【解答】

地址	115	228	120	88	446	102	321	432	260	167
页号	1	2	1	0	4	1	3	4	2	1

（1）先进先出页面置换算法

	1	2	1	0	4	1	3	4	2	1
块 1	0	0			4		4			4
块 2	1	1			1		3			3
块 3		2			2		2			1
是否缺页	√	√			√		√			√
置换的					0		1			2

按 FIFO 调度算法将产生 5 次缺页中断，依次淘汰的页号为 0、1、2，缺页中断率为 50%；

（2）最近最久未使用页面置换算法

	1	2	1	0	4	1	3	4	2	1
块 1	0	0			0		3		3	1
块 2	1	1			1		1		2	2
块 3		2			4		4		4	4
是否缺页	√	√			√		√		√	√
置换的					2		0		1	3

按 LRU 调度算法将产生 6 次缺页中断，依次淘汰的页号为 2、0、1、3，缺页中断率为 60%；

8 磁盘调度算法

（1）先来先服务
（2）最短寻道时间优先
（3）扫描算法
（4）循环扫描算法

【例题 1】教材例题

【例题 2】课后习题 8.

8. 假定磁盘有 200 个柱面，编号 0～199，当前存取臂的位置在 143 号柱面上，并刚刚完成了 125 号柱面的服务请求，如果请求队列的先后顺序是：86，147，91，177，94，150，102，175，130；试问：为完成上述请求，下列算法存取臂移动的总量是多少？并算出存取臂移动的顺序。

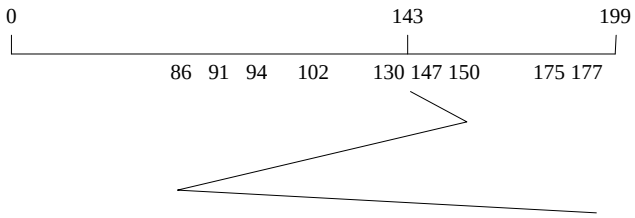
（1）先来先服务算法 FCFS；
（2）最短查找时间优先算法 SSTF；
（3）扫描算法 SCAN。

答：

（1）先来先服务算法 FCFS；

移动总量 = 565

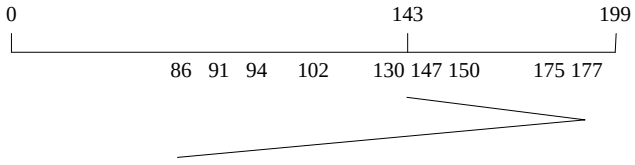
（2）最短查找时间优先算法 SSTF



移动总量 = (150 – 143) + (150 – 86) + (177 – 86) = 7 + 64 + 91 = 162

处理磁道序列是：143 147 150 130 102 94 91 86 175 177

（3）扫描算法 SCAN



移动总量 = (177-143) + (177 – 86) = 34 + 91 = 125

处理磁道序列是：143 147 150 175 177 130 102 94 91 86

注：

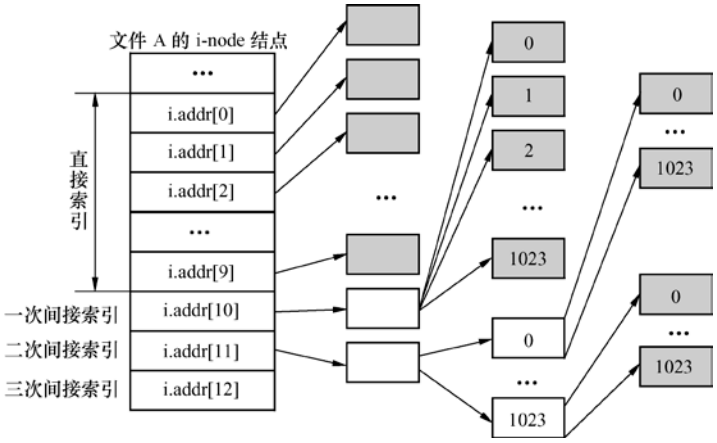
（1）在计算最短查找时间优先算法和扫描算法时，将所有的请求从小到大排列。
（2）最短查找时间优先算法，每次都要查看与当前磁头的最近的两个请求的距离，从而判

断磁头向左还是向右移动。（切勿第一次左右查看，后面的请求不查看）
（3）扫描算法不同的教材讲解会有不同，有的教材上磁头每次都要移动到磁盘最大的磁道（199）和磁盘最小的磁道（0），我们教材上讲解的是移动到请求最大的磁道（177）和请求最小的磁道（86）。
（4）扫描算法要注意磁头移动方向，此题中当前磁头是 143，刚处理完的是 125 号磁道，表示磁头向磁道号大的方向移动。

9 混合索引

【例题 1】教材例题 混合索引

Linux 操作系统采用的是混合索引方式，如图 7-6 所示。每个文件的索引表规定为 13 个索引项，每项 4 个字节，登记一个存放文件信息的物理块号。前面 10 项存放文件信息的物理块号，叫直接寻址。假定物理块的大小是 4KB，直接寻址可以表示 40KB（4KB×10）的文件。如果文件大于 10 块，则利用第 11 项指向一个物理块，该块中最多可放 1K 个存放文件信息的物理块的块号。这种方式是一次间接寻址，可以表示 4MB（4KB×1K）的文件。每个大型文件还可以利用第 12 和 13 项作二次和三次间接寻址，二次间接寻址可以表示 4GB 的文件，三次间接寻址可以表示 4TB 的文件。



类别	物理块数（数据）	表示的文件	索引块数
直接索引	10 个	4K*10 = 40KB	0
一次间接索引	(4K/4) = 1024	4K*1024 = 4MB	1
二次间接索引	1024*1024	4K*1024*1024 = 4GB	1024
三次间接索引	1024*1024*1024	4K*1024*1024*1024 = 4TB	1024*1024

【例题 2】课后习题 12.

12. 如果一个索引节点为 128B，指针长 4B，状态信息占用 68B，而每块大小为 8KB。问在索引节点中有多大空间给指针？使用直接、间接、二次间接和三次间接指针分别可表示多大的文件？

【解答】

索引结点中有 60B 给指针。

直接索引 96KB,一次间接索引 16MB，二次间接索引 32GB，一次间接索引 64TB。

索引点 128B － 68B = 60B，60B 用来存放文件的存储信息（盘块指针）。
60B/4B = 15 个，可以存放 15 个盘块地址（盘块号）。
15 个盘块号中

1 个用于一次间接索引
1 个用于二次间接索引
1 个用于三次间接索引
剩下的 12 个用于直接索引

每个盘块中可以存放 8KB/4B = 2K 个盘块地址。

直接索引：8KB * 12 = 96KB
一次间接索引：8KB *（8KB/4B） = 8KB * 2K = 16 MB
二次间接索引：8KB *（8KB/4B）*（8KB/4B） = 8KB * 2K * 2K = 32 GB
三次间接索引：8KB *（8KB/4B）*（8KB/4B）*（8KB/4B）
= 8KB * 2K *2K *2K = 64 TB