

## 第一章 引论

### • 操作系统的主要功能

1. 处理机管理
2. 存储器管理
3. 设备管理
4. 文件管理
5. 接口管理

### • 操作系统的特征

1. 并发：两个或多个事件在同一时间间隔内发生。
2. 共享：系统中的资源可供内存中多个并发执行的进程共同使用。
3. 虚拟：把一个物理上的实体变为若干个逻辑上的对应物。
4. 异步：在多道程序环境下，运行走走停停，以不可知的速度向前推进。

• 操作系统在计算机系统中处于计算机硬件和用户之间的位置。

• 操作系统的基本类型有实时操作系统、批处理操作系统及分时操作系统。

#### 1. 实时操作系统：

优先处理紧急事务，实时性、可靠性

#### 2. 批处理操作系统：

多道批处理并发执行，提高资源利用率

#### 3. 分时操作系统：

允许用户直接与计算机进行交互。

## 第二章 进程的描述与控制

• 进程：动态的、程序的一次执行过程

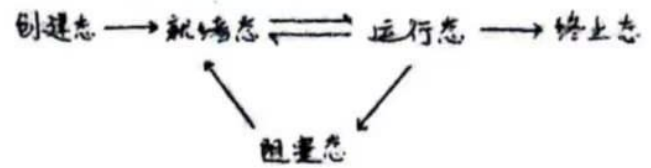
进程的组成

- PCB { 进程标识符 PID  
用户标识符 UID  
控制信息  
程序段：程序的代码  
数据段：运行时产生的数据

### • 进程的特征

动态性、并发性、独立性、异步性、结构性

### • 进程的转换

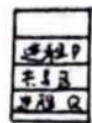


运行态 → 就绪态：时间片到，CPU被其他高优先级进程抢占。

运行态 → 阻塞态：等待系统资源分配自己，等待事件发生。

### • 进程通信的3种方式：

#### 1. 共享存储器机制



#### 2. 消息传递机制



#### 3. 管道通信



### • 线程

线程是一个基本的CPU的执行单元，也是程序执行流的最小单位。

引入线程之后，进程之间可并发，进程内的各线程之间也可以并发。

进程是资源分配的基本单位，线程是调度的基本单位。

每个线程都有一个线程ID，线程控制块(TCB)。

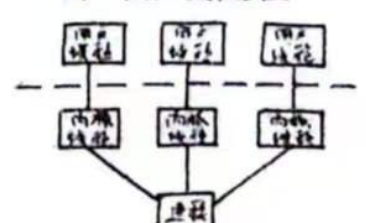
线程也有就绪态、阻塞态、运行态三种基本状态。

### 线程的实现方式

#### ★ 用户级线程



#### 内核级线程



第3章 处理机调度与死锁

• 进程调度方式:

1. 非抢占方式:

进程一直执行完,才执行下一个进程

2. 抢占式:

有更重要的进程需要处理机时,将处理机分配给重要的进程.

• 调度算法的指标

周转时间 = 完成时间 - 到达时间

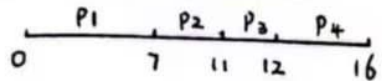
带权周转时间 =  $\frac{\text{周转时间}}{\text{实际运行时间}}$

等待时间 = 周转时间 - 实际运行时间

• 调度算法

1. 先来先服务 (FCFS)

进程	到达时间	运行时间
P1	0	7
P2	2	4
P3	4	1
P4	5	4



周转时间:  $P1 = 7 - 0 = 7$   
 $P2 = 11 - 2 = 9$   
 $P3 = 12 - 4 = 8$   
 $P4 = 16 - 5 = 11$

带权周转时间:  $P1 = 7 / 7 = 1$   
 $P2 = 9 / 4 = 2.25$   
 $P3 = 8 / 1 = 8$   
 $P4 = 11 / 4 = 2.75$

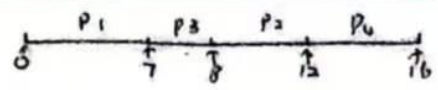
等待时间:  $P1 = 7 - 7 = 0$   
 $P2 = 9 - 4 = 5$   
 $P3 = 8 - 1 = 7$   
 $P4 = 11 - 4 = 7$

2. 短作业优先

2.1 非抢占式

进程	到达时间	运行时间
P1	0	7
P2	2	4
P3	4	1
P4	5	4

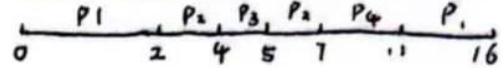
$P1 \rightarrow P3 \rightarrow P2 \rightarrow P4$



周转时间:  $P1 = 7 - 0 = 7$   $P1 = 7 / 7 = 1$   $P1 = 7 - 7 = 0$   
 $P2 = 12 - 2 = 10$   $P2 = 10 / 4 = 2.5$   $P2 = 10 - 4 = 6$   
 $P3 = 8 - 4 = 4$   $P3 = 4 / 1 = 4$   $P3 = 4 - 1 = 3$   
 $P4 = 16 - 5 = 11$   $P4 = 11 / 4 = 2.75$   $P4 = 11 - 4 = 7$

2.2 抢占式

进程	到达时间	运行时间
P1	0	7
P2	2	4
P3	4	1
P4	5	4



0: P1 (7)  
2: P1 (5), P2 (4)  
4: P1 (5), P2 (2), P3 (1)  
5: P1 (5), P2 (2), P4 (4)  
7: P1 (5), P4 (4)  
11: P1 (5)

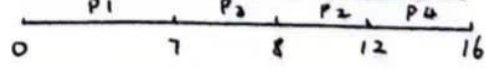
周转时间:  $P1: 16 - 0 = 16$   $P1: 16 / 7 = 2.28$   
 $P2: 7 - 2 = 5$   $P2: 5 / 4 = 1.25$   
 $P3: 5 - 4 = 1$   $P3: 1 / 1 = 1$   
 $P4: 11 - 5 = 6$   $P4: 6 / 4 = 1.5$

等待时间:  $P1: 16 - 7 = 9$   $P3: 1 - 1 = 0$   
 $P2: 5 - 4 = 1$   $P4: 6 - 4 = 2$

3. 优先级调度

3.1 非抢占式

进程	到达时间	运行时间	优先级
P1	0	7	1
P2	2	4	2
P3	4	1	3
P4	5	4	2



周转时间:  $P1: 7 - 0 = 7$   $P1: 7 / 7 = 1$   $P1: 7 - 7 = 0$   
 $P2: 12 - 2 = 10$   $P2: 10 / 4 = 2.5$   $P2: 10 - 4 = 6$   
 $P3: 8 - 4 = 4$   $P3: 4 / 1 = 4$   $P3: 4 - 1 = 3$   
 $P4: 16 - 5 = 11$   $P4: 11 / 4 = 2.75$   $P4: 11 - 4 = 7$



#### 4. 高响应比优先

非抢占式，选响应比最高的进程上机。

进程	到达时间	运行时间
P1	0	7
P2	2	4
P3	4	1
P4	5	4

响应比：  
等待时间 + 快速服务 / 原服务时间

0时刻：只有P1到达，P1上处理机

7时刻：P1运行中有P4、P2、P3。

响应比：P2:  $\frac{5+4}{4} = 2.25$

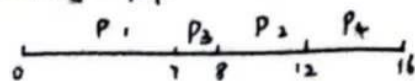
P3:  $\frac{3+1}{1} = 4$  ✓

P4:  $\frac{2+4}{4} = 1.5$

8时刻：P2:  $\frac{6+4}{4} = 2.5$  ✓

P4:  $\frac{3+4}{4} = 1.75$

12时刻：P4



#### • 银行家算法

##### 1. 数据结构

可利用资源向量 Available

最大需求矩阵 Max

分配矩阵 Allocation

需求矩阵 Need

##### 2. 安全性算法

(1) 若找到  $Finish[i] = FALSE, Need[i, j] \leq Work[j]$

(2)  $work[j] = work[j] + Allocation[i, j]$

$Finish[i] = TRUE$

(3) 所有进程满足  $Finish[i] = TRUE$ ，安全状态。

eg:

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

P1: Available: 5 3 2

P1 → P3: Available: 7 4 3

P1 → P3 → P4: Available: 7 4 5

P1 → P3 → P4 → P0: Available: 7 5 5

P1 → P3 → P4 → P0 → P2: Available: 10 5 7

##### 3. 银行家算法

(1)  $Request[j] \leq Need[i, j]$

(2)  $Request[j] \leq Available[j]$

(3)  $Available[j] = Available[j] - Request[j]$

$Allocation[i, j] = Allocation[i, j] + Request[j]$

$Need[i, j] = Need[i, j] - Request[j]$

(4) 检查是否处于安全状态。

eg: P1发出请求向量  $Request(1, 0, 2)$ ，系统按银行家算法进行检查。

(1)  $Request(1, 0, 2) \leq Need(1, 2, 2)$

(2)  $Request(1, 0, 2) \leq Available(3, 3, 2)$

(3) Available(2, 3, 0) (4) 安全性算法

Allocation(3, 0, 2)

need(0, 2, 0)

#### • 死锁

各进程互相等待对方手里的资源，导致进程都阻塞，无法向前推进的现象。

##### • 死锁产生的必要条件

1. 互斥条件
2. 不剥夺条件
3. 请求和保持条件
4. 循环条件

##### • 死锁的处理策略

1. 预防死锁：破坏死锁产生四个必要条件中的一个。
2. 避免死锁：用某种方式防止进入不安全状态。
3. 检测死锁：允许死锁发生，检测出发生后采取措施。
4. 解除死锁。

##### • 处理策略

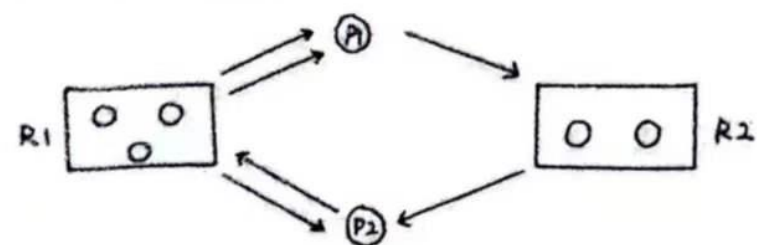
预防死锁：对资源采用按序分配策略。

避免死锁：银行家算法。

系统处于安全状态，就一定不会发生死锁

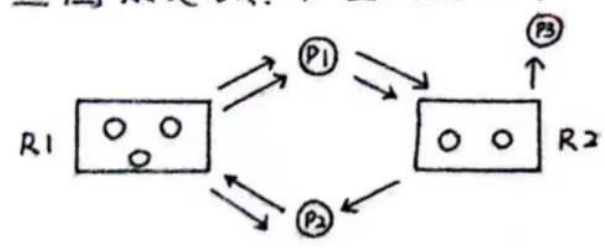
系统处于不安全状态，就可能发生死锁。

## • 死锁的检测



找出一条有向边与它相连,且该有向边对应资源的申请数量小于等于系统中已有空闲资源数量。(从资源向进程的边减去,还有资源).

上图未死锁,下图为死锁.



## 第四章 进程同步

### • 进程同步与进程互斥

进程同步:两个或多个进程在它们的工作次序产生的制约关系.

进程互斥:一个进程访问临界资源时,另一个想要访问该资源时必须等待.

### • 信号量机制

wait(s)原语 P操作  
signal(s)原语 V操作  
记录型信号量

```
/*记录型信号量的定义*/
typedef struct {
    int value; // 剩余资源数
    struct process *L; // 等待队列
} semaphore;
```

```
void wait(semaphore s){
    S.value--;
    if (S.value < 0) block(S.L);
} // 可能进入阻塞
```

```
void signal(semaphore s){
    S.value++;
    if (S.value <= 0) wakeup(S.L);
} // 可能进行唤醒
```

S.value ++ 之后  $\leq 0$ , 有进程等待该资源.  
S.value ++ 之后  $> 0$ , 已没有进程等待.  
S.value 初值为某资源的数目.  
S.value = -2, 有2个进程在等待.

### • 用信号量实现进程互斥, 同步

#### 1. 进程互斥

(互斥信号量 mutex, 初值为1)  
在进入区 P(mutex) —— 申请资源  
在退出区 V(mutex) —— 释放资源

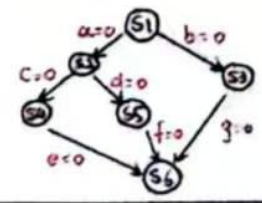
```
semaphore mutex = 1;
P1(){
    ...
    P(mutex);
    代码段
    V(mutex);
    ...
}
```

#### 2. 进程同步

(同步信号量 S, 初值为0)  
在前操作"之后执行 V(S)  
在后操作"之前执行 P(S)  
设代码2在代码4之前执行

```
semaphore S = 0;
P1(){
    代码1;
    代码2;
    V(S);
}
P2(){
    代码3;
    P(S);
    代码4;
}
```

#### 3. 前驱关系



<pre>P1(){     ...     S1;     V(a);     V(b);     ... }</pre>	<pre>P2(){     ...     P(a);     S2;     V(c);     V(d);     ... }</pre>	<pre>P3(){     ...     P(b);     S3;     V(g);     ... }</pre>
<pre>P4(){     P(c);     S4;     V(e);     ... }</pre>	<pre>P5(){     P(d);     S5;     V(f);     ... }</pre>	<pre>P6(){     ...     P(e);     P(f);     P(g);     S6; }</pre>



## 第五章 存储器管理

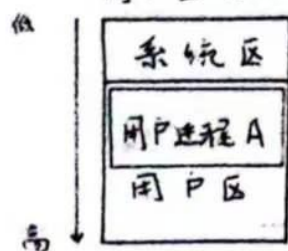
- 解决“程序大小超过物理内存总和”问题，使用覆盖技术与交换技术。
- 连续分配存储管理方式

### 1. 单一连续分配

分为系统区与用户区。

系统区位于内存的低地址部分。

用户区存放用户进程的数据。



### 2. 固定分区分配

用户区划分固定大小的分区，每个分区只装一进程。

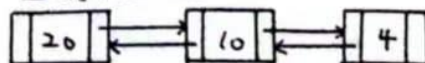
系统区	系统区
分区1	分区1
分区2	分区2
分区3	分区3
分区4	分区4

### 3. 动态分区分配

空闲分区表

分区号	分区地址(MB)	起始地址	状态
1	20	8	空闲
2	10	32	空闲
3	4	60	空闲

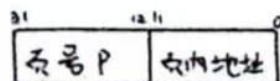
空闲分区链表



动态分区分配算法

- 首次适应算法：找到第一个能满足大小的块
- 最佳适应算法：优先使用更小的空闲区
- 最坏适应算法：优先使用更大的空闲区
- 邻近适应算法：从上次查找结束位置开始查找第一个空闲分区。

- 分页存储管理



逻辑地址

0 页
1 页
...
n 页

页号 块号

0	3
1	6
...	...
n	5



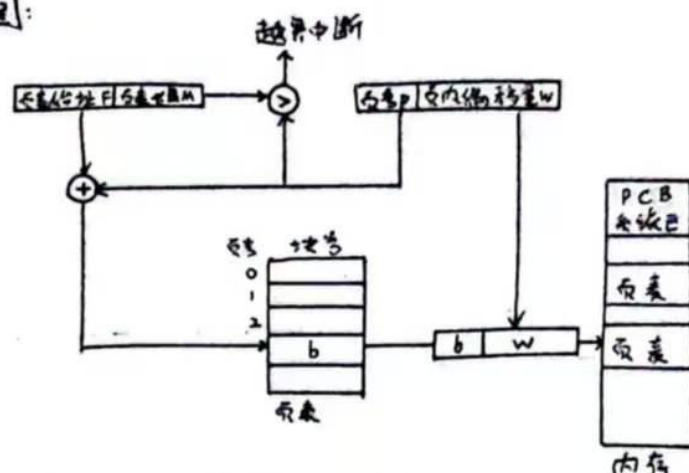
计算：

页号 = 逻辑地址 / 页面长度

页内偏移量 = 逻辑地址 % 页面长度

物理地址 = 页面地址 + 页内偏移量

图：



例：某作业的逻辑地址空间为4页（每页2048字节），页面映像表如右图，求出逻辑地址4865所对应的物理地址。

页号	块号
0	2
1	4
2	6
3	8

页号 =  $4865 / 2048 = 2$

页内偏移量 =  $4865 \% 2048 = 769$

物理地址 =  $2 \times 2048 + 769 = 4865$

## 第6章 虚拟存储器

- 页面置换算法

### 1. 最佳置换算法 (OPT)

淘汰页面：将是以后最长时间不再被访问的页面。

例：分配3个内存块

访问页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
内存1	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
内存2		0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
内存3			1	1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
是否缺页	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√

发生9次缺页，页面置换发生6次

缺页率 =  $9 / 20 = 45\%$

