swiss126

目录

一、操作系统概论.....3
1.操作系统的目标：方便性、有效性 、可扩充性、开放性.....3
2.操作系统的作用：.....3
3.推动操作系统发展的主要动力.....3
4.操作系统的发展.....3
5.操作系统的基本特性：并发、共享、虚拟、异步.....3
6.操作系统的主要功能.....4
7.操作系统接口.....4
8.操作系统的结构.....4
二、进程管理.....5
1.程序执行的表示方法.....5
2.进程（可拥有资源的独立单位）.....5
3.进程控制.....5
4. 线程（调度和分派的基本单位）.....6
5.多核、多线程与超线程.....8
三、互斥与同步.....9
1.进程同步的基本概念.....9
2.互斥的实现方法.....9
3.进程通信.....12
4.死锁.....13
四、处理机调度.....16
1.处理机调度层次.....16
2.调度准则.....16
3.短程调度算法.....16
4.优先级调度算法，优先级的类型.....17
5.多级反馈队列轮换调度算法.....17
6.实时调度.....17

五、存储器管理 19
1.存储器的层次结构 19
2.存储分配形式 19
3.重定位：将逻辑地址转换称物理地址。 19
4.覆盖与交换 20
3.单道环境下的存储管理 21
4.分区存储管理 21
5.页式存储管理 23
六、文件管理 61
1.文件 61
2.文件操作： 62
3.文件结构 62
4.文件目录 65
5.文件共享与保护 69
6.文件存储空间管理 70
7.文件分配 73
8.Windows 文件管理 77
七、IO 管理 81
1.基本概念 81
2.I/O 管理目标与功能 81
3.IO 系统 82
4.I/O 控制方式 84
5.缓冲区管理 87
6.设备驱动程序 90
7.设备分配： 45

一、操作系统概论

1.操作系统的目标：方便性、有效性 、可扩充性、开放性

2.操作系统的作用：

✓ 用户与计算机硬件系统之间的接口（**系统调用、命令、图标-窗口**）

✓ 计算机系统资源的管理者

✓ 对计算机资源的抽象

3.推动操作系统发展的主要动力

✓ 不断提高计算机资源利用率

✓ 方便用户

✓ 器件的不断更新换代

✓ 计算机体系结构的不断发展

✓ 不断提出新的应用需求

4.操作系统的发展

4.1 未配置操作系统的计算机系统

4.2 单道批处理系统（缺点：资源利用不充分）

4.3 多道批处理系统

优点：资源利用率高、系统吞吐量大

缺点：平均周转时间长、无交互能力

问题：处理机争用问题、内存分配和保护问题、I/O 设备分配问题
文件的组织和管理问题、作业管理问题、用户与系统的接口问题

4.4 分时系统

满足人机需求：人机交互、共享主机

关键问题：及时接收、及时处理

特征：多路性、独立性、及时性、交互性

4.5 实时系统

用途：工业(武器)控制、信息查询、多媒体、嵌入式

任务：周期性实时、非周期性实时、硬实时、软实时

特征：多路性、独立性、及时性、交互性、 可靠性

4.6 微机操作系统

单用户单任务： CP/M 、 MS-DOS

单用户多任务：Windows

多用户多任务：UNIX、Linux

5.操作系统的基本特性：并发、共享、虚拟、异步

5.1 并发

并行性是指两个或多个事件在**同一时刻**发生。

并发性是指两个或多个事件在**同一时间**间隔内发生。

实现方法：引入进程

5.2 共享（外设互斥共享、磁盘设备同时访问）

5.3 虚拟

时分复用：虚拟处理机、虚拟设备

空分复用：提高存储空间的利用率

5.4 异步（进程以人们不可预知的先前推进，这就是进程的异步性）

6.操作系统的主要功能

处理机管理功能：进程控制、进程同步、进程通信、作业进程调度

存储器管理功能：内存分配（静态分配、动态分配）、内存保护、地址映射、内存扩充

设备管理功能：缓冲管理、设备分配、设备处理

文件管理功能：文件存储空间管理、目录管理、文件读写管理和保护

7.操作系统接口

7.1 用户接口（UI）

联机用户接口：又称交互命令接口

脱机用户接口：又称批处理命令接口

图形用户接口（GUI）

7.2 程序接口（API）

8.操作系统的结构

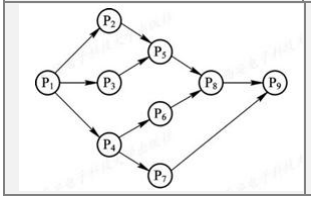
第一代：无结构的 OS
第二代：模块化的 OS
第三代：分层式结构的 OS　} 传统结构的 OS

第四代：微内核的 OS → 现代结构的 OS

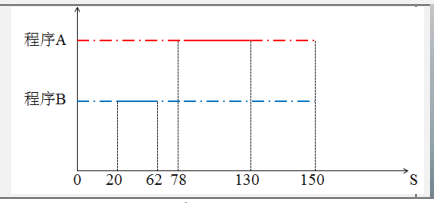
二、进程管理

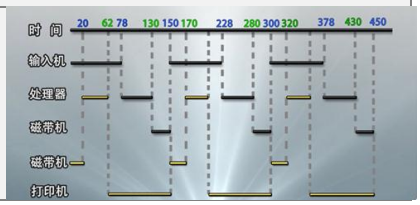
1.程序执行的表示方法

前趋图



甘特图

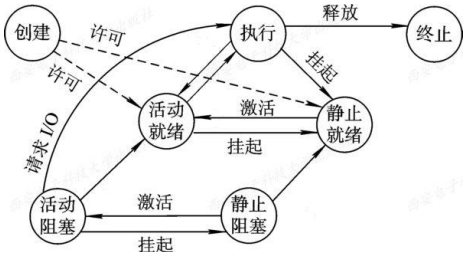




2.进程（可拥有资源的独立单位）

2.1 特征：动态性、并发性、独立性、异步性、结构性

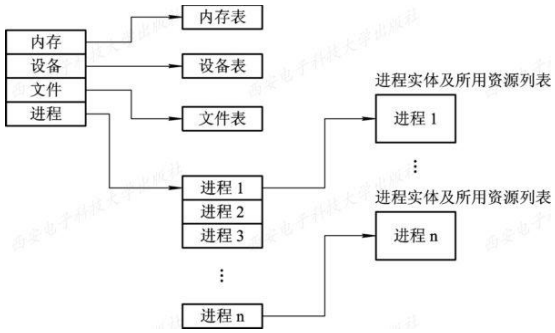
2.2 状态：创建、就绪、执行、阻塞



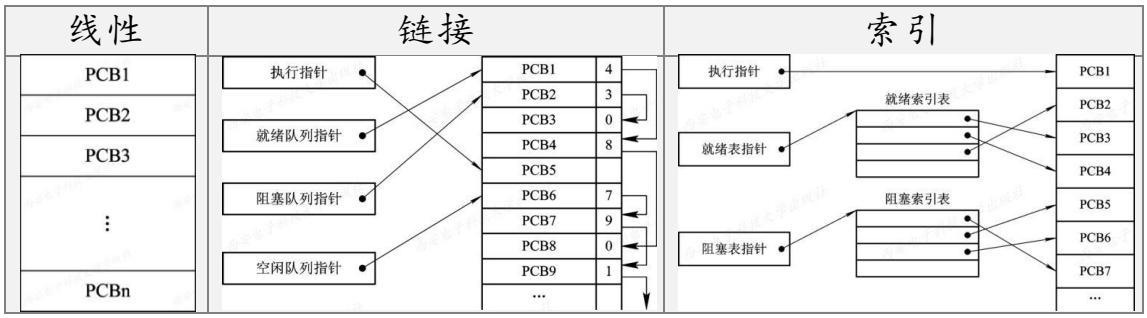
2.3 数据结构：内存表、设备表、文件表和进程表（进程控制块 PCB）

PCB 的作用：作为独立运行基本单位的标志、能实现间断性运行方式、提供进程管理所需要的信息、提供进程调度所需要的信息、实现与其它进程的同步与通信。

PCB 的结构：

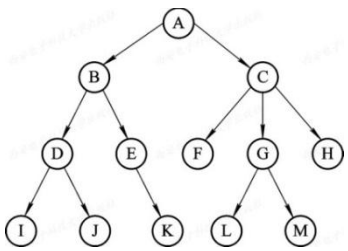


PCB 的组织方式：



3.进程控制

3.1 进程的层次结构与进程图



3.2 进程创建

引起进程创建的事件：用户登录、作业调度、提供服务、应用请求

创建过程：OS 调用进程**创建原语 Creat**

(1) 申请空白 PCB，为新进程申请获得唯一的数字标识符，并从 PCB 集合中索取一个空白 PCB。

(2) 为新进程分配其运行所需的资源，包括各种物理和逻辑资源，如内存、文件、I/O 设备和 CPU 时间等。

(3) 初始化进程控制块(PCB)。

(4) 如果进程就绪队列能够接纳新进程，便将新进程插入就绪队列。

3.3 进程终止

✓ 引起进程终止的事件：正常结束、异常结束、外界干预

✓ 终止过程：OS 调用进程**终止原语**

(1) 根据被终止进程的标识符，从 PCB 集合中检索出该进程的 PCB，从中读出该进程的状态；

(2) 若被终止进程正处于执行状态，应立即终止该进程的执行，并置调度标志为真，用于指示该进程被终止后应重新进行调度；

(3) 若该进程还有子孙进程，还应将其所有子孙进程也都予以终止，以防它们成为不可控的进程；

(4) 将被终止进程所拥有的全部资源或者归还给其父进程，或者归还给系统；

(5) 将被终止进程(PCB)从所在队列(或链表)中移出，等待其它程序来搜集信息。

3.4 进程阻塞与唤醒

阻塞：进程调用**阻塞原语 block** 将自己阻塞，进程阻塞是**主动行为**

唤醒：有关进程调用**唤醒原语 wakeup**，等待该事件的进程唤醒，进程唤醒是**被动行为**

3.5 进程的挂起与激活

挂起：OS 利用**挂起原语 suspend** 将指定进程或处于阻塞的进程挂起

激活：OS 将利用**激活原语 active** 将指定进程激活

3.6 原语：由若干指令组成，用于完成一定功能。具有不可分割性，执行过程中不允许被中断。

4. 线程（调度和分派的基本单位）

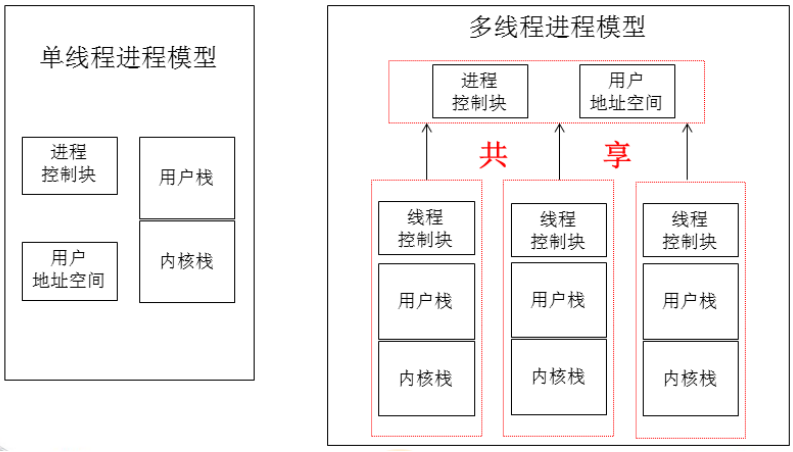
4.1 进程与线程的比较：

✓ 进程是可拥有资源的独立单位

✓ 线程是调度和分派的基本单位。

拥有并发性、独立性，支持多处理机系统

✓ 线程的引入：减少程序并发执行时所付出的时空开销，使 OS 具有更好的并发性。



4.2 线程的状态（执行、就绪、阻塞）和线程控制块（TCB）

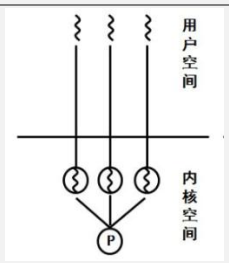
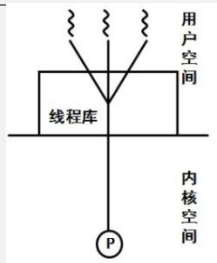
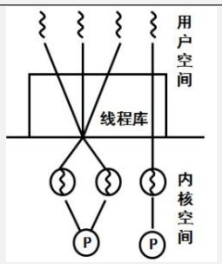
4.3 多线程 OS 中的进程属性

(1)拥有资源的基本单位。

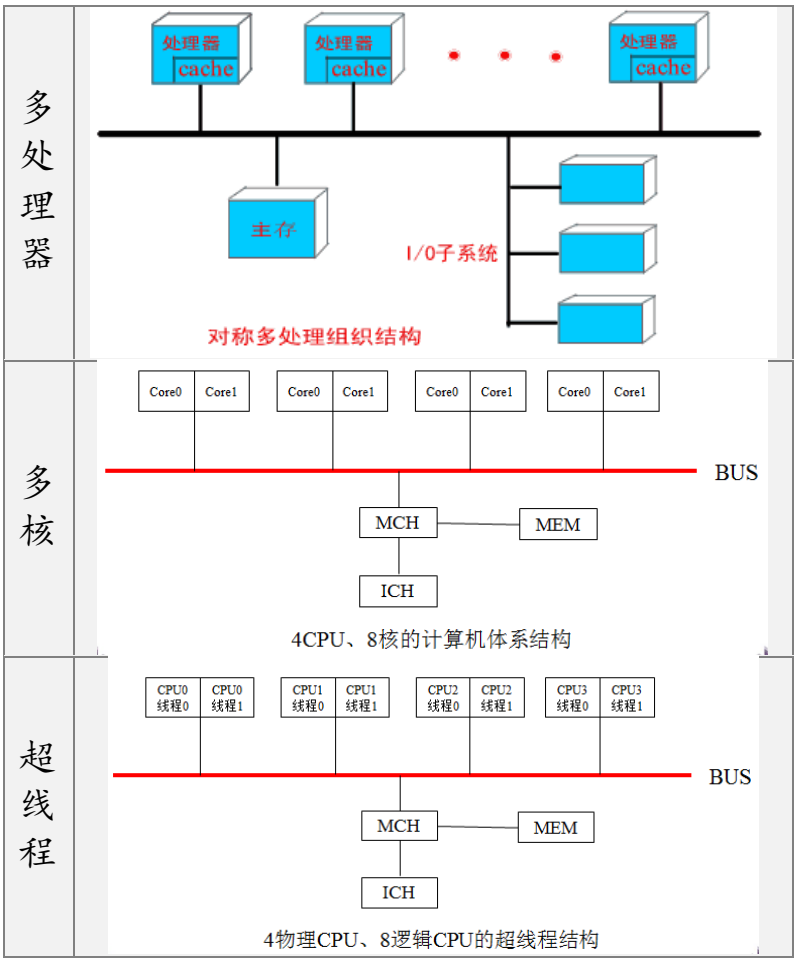
(2)多个线程可并发执行。

(3)进程已不是可执行的实体。

4.4 线程的实现

	内核支持线程 KST	用户级线程 ULT	混合式线程
			
优点	(1)内核同时调度同一进程中多个线程并行执行 (2)如果进程中的一个线程被阻塞了，内核可以调度该进程中的其它线程占有处理器运行，也可以运行其它进程中的线程 (3)线程的切换比较快 (4)内核本身也可以采用多线程技术，可以提高系统的执行速度和效率。	(1)线程切换不需要转换到内核空间。 (2)调度算法可以是进程专用的。 (3)用户级线程的实现与 OS 平台无关，因为对于线程管理的代码是属于用户程序的一部分，所有的应用程序都可以对之进行共享。	用户级线程和内核支持线程两种方式进行组合，提供了组合方式 ULT/KST 线程。在组合方式线程系统中，内核支持多个内核支持线程的建立、调度和管理，同时，也允许用户应用程序建立、调度和管理用户级线程。
缺点	对于用户的线程切换而言，其模式切换的开销较大，在同一个进程中从一个线程切换到另一个线程时，需要从用户态转到核心态进行，这是因为用户进程的线程在用户态运行，而线程调度和管理是在内核实现的，系统开销较大。	(1)系统调用的阻塞问题。在基于进程机制的 OS 中，大多数系统调用将使进程阻塞，因此，当线程执行一个系统调用时，不仅该线程被阻塞，而且，进程内的所有线程会被阻塞。而在内核支持线程方式中，则进程中的其它线程仍然可以运行。 (2)在单纯的用户级线程实现方式中，多线程应用不能利用多处理机进行多重处理的优点，内核每次分配给一个进程的仅有一个 CPU，因此，进程中仅有一个线程能执行，在该线程放弃 CPU 之前，其它线程只能等待。	

5.多核、多线程与超线程


多处理器
处理器
cache
处理器
cache
处理器
cache
主存
I/O子系统
对称多处理组织结构
多核
Core0
Core1
Core0
Core1
Core0
Core1
Core0
Core1
BUS
MCH
MEM
ICH
4CPU、8核的计算机体系结构
超线程
CPU0 线程0
CPU0 线程1
CPU1 线程0
CPU1 线程1
CPU2 线程0
CPU2 线程1
CPU3 线程0
CPU3 线程1
BUS
MCH
MEM
ICH
4物理CPU、8逻辑CPU的超线程结构

三、互斥与同步

1.进程同步的基本概念

✓ 两种形式的制约关系：**互斥与同步**

✓ **临界资源**：打印机、 磁带机、共享变量等，都属于临界资源，一次只能被一个进程使用的资源。**诸进程互斥共享临界资源。**

✓ **临界区**：每个进程中访问临界资源的那段代码称为临界区

✓ 临界区管理的三个要求：

(1) 一次最多允许一个进程停留在相关的临界区内；

(2) 一个进程不能无限止地停留在临界区；

(3) 一个进程不能无限止地等待进入在临界区。

✓ 同步机制应遵循的规则：

空闲让进、忙则等待、有限等待、让权等待。

2.互斥的实现方法

2.1 标志法

```
inside1=false;inside2=false;
```

Process P1{	Process P2{
while inside2 do {};	while inside1 do {};
insede1=true;	insede2=true;
临界区；	临界区；
inside1=false;	inside2=false;
}	}

2.2 硬件实现方法（TS、Swap、关中断）

（1）利用 Test-and-Set 指令实现互斥

```
Boolean TS（boolean *lock）{
      boolean old;
      old =*lock;
      *lock=true;
      return old;
}
      …
      while TS(&lock);    //测试 lock 并设置 lock 的值
      临界区；
      lock = false;
      ….
```

分析：当 TS(&lock)的值为假时，设置 lock 为真并结束循环。

当 TS(&lock)的值为真时，继续循环。

（2）利用 Swap 指令实现进程互斥

```
swap(boolean a,boolean b)
{
      boolean temp;
      temp=a;
      a=b;
      b=temp
}
```

```
…
boolean key;
key=true;
do
      swap(&lock,key);
while (key);
临界区;
lock=false;
…
```

（3）关中断

在进入锁测试之前关闭中断，直到完成锁测试并上锁之后才能打开中断。进程在临界区执行期间，计算机系统不响应中断，从而不会引发调度，也就不会发生进程或线程切换。保证了对锁的测试和关锁操作的连续性和完整性，有效地保证了互斥。

但是，关中断的方法存在许多缺点：

①滥用关中断权力可能导致严重后果；②关中断时间过长，会影响系统效率，限制了处理器交叉执行程序的能力；③关中断方法也不适用于多 CPU 系统，因为在一个处理器上关中断并不能防止进程在其它处理器上执行相同的临界段代码。

2.3 信号量及 P、V 原语

✓ 解决问题：

1）TS 或 SWAP 指令管理临界区，采用忙式轮询，效率低；

2）关开中断管理临界区，不便交给用户程序使用。

1. 信号量的构思

一种可动态定义的软件资源：**信号量**

◆ 核心数据结构：等待进程队列

◆ 信号量声明：资源报到，建立队列

◆ 申请资源的原语：若申请不到，调用进程入队等待。

◆ 归还资源的原语：若队列中有等待进程，需释放。

◆ 信号量撤销：资源注销，撤销队列。

2. 记录型信号量

■ 其数据结构如下：

```
typedef struct semaphore{
     int value;                    //信号量值
     struct pcb *list;       //信号量等待进程队列指针
  }
```

■ 每个信号量建立一个等待进程队列

■ 每个信号量相关一个整数值

✓ 正值表示资源可用的数量

✓ 0 值表示无资源且无进程等待

✓ 负值表示等待队列中进程的个数

3. P、V 操作原语

```
Viod P(semaphore s){
  s.value=s.value-1;
  if (s.value<0) {
      将该进程插入到 s.list 的等待进程队列中;
      block(s.list);
  }
}
Viod V(semaphore s){
  s.value=s.value+1;
  if (s.value<=0) {
      从 s.list 的等待进程队列中释放一个进程 P;
      wakeup(P);
  }
}
```

2.4 用 P、V 操作原语实现进程互斥

```
semaphore mutex;
mutex=1;
…
process Pi{
  …
  P(mutex);
  进程 Pi 进入临界区
  V(mutex);
  …
}
```

2.5 用 PV 操作解决缓冲区问题

（1）单缓冲区问题

```
int B;                  //共享缓冲区
Semaphore Sput;         //可以使用的空缓冲区数
Semaphore Sget;         //缓冲区内可以使用的产品数
Sput=1;                 //缓冲区内允许放入一件产品
Sget=0;                 //缓冲区内没有产品
```

```
Process Producer{
  ……
  生产一个产品product;
  P(Sput);
  B= product;
  V(Sget);
  ……
}
```

```
Process Consumer{
  ……
  P(Sget);
  product =B;
  V(Sput);
  消费一个产品product;
  ……
}
```

（2）M 生产者 N 消费者 K 个缓冲区

```
int B[K];                    //建立K个共享缓冲区
Semaphore Sput;              //可以使用的空缓冲区数
Semaphore Sget;              //缓冲区内可以使用的产品数
Semaphore S;                 //公用信号量，保证缓冲区互斥
Sput.value=K;                //缓冲区内允许放入K件产品
Sget.value=0;                //缓冲区内没有产品
S.value=1;
int in=0;out=0;              //定义2个进/出产品的指针
```

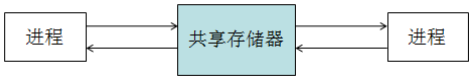
```
Process Producer_i{
  int item;
  生产1个产品暂存在item;
  P(Sput);
  P(S);
  B[in]=item;
  in=(in+1)%k;
  V(Sget);
  V(S);
}
```

```
Process Consumer_i{
  int item;
  P(Sgut);
  P(S);
  item=B[out];
  out=(out+1)%k;
  V(Spet);
  V(S);
  消费 item;
}
```

3.进程通信

3.1 进程通信的类型

1. 共享存储器系统(Shared-Memory System)

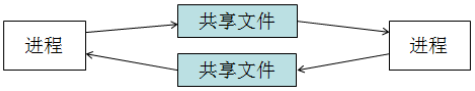


可分为以下两种类型：

①基于共享数据结构的通信方式。

②基于共享存储区的通信方式。

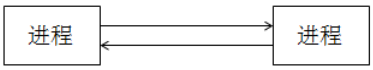
2. 管道(pipe)通信系统



为了协调双方的通信，管道机制必须提供以下三方面的协调能力：

①互斥②同步③确定对方是否存在

3. 消息传递系统(Message passing system)

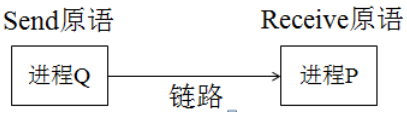


可进一步分成直接通信方式和间接通信方式

3.2 消息传递通信的实现方式

1. 直接消息传递系统

1）直接传递原语



send(P，信件)：把信件发送给进程 P

receive(Q，信件)：从进程 Q 接收信件

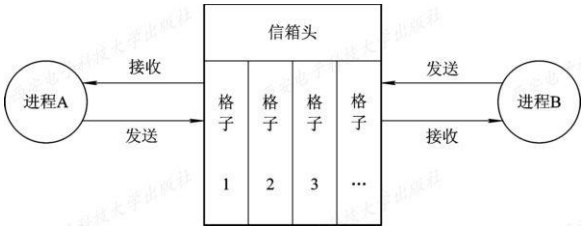
2) 进程的同步方式

3) 通信链路

2.间接消息传递系统——信箱通信

■ 发送或者接收信件通过一个信箱来进行，该信箱有唯一标识符。

■ 多个进程共享一个信箱。



信箱通信原语

Send（A，信件）：把信件传送到 A 信箱；

Receive（A，信件）：从 A 信箱中接收信件。

4.死锁

4.1 死锁的概念

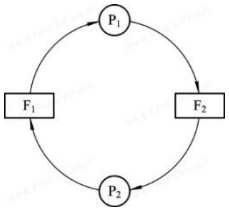
1.定义：在一组进程发生死锁的情况下，这组死锁进程中的每一个进程，都在等待另一个死锁进程所占有的资源。

2.可重用性资源和消耗性资源

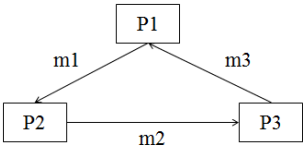
3.可抢占性资源和不可抢占性资源

4. 计算机系统中的死锁

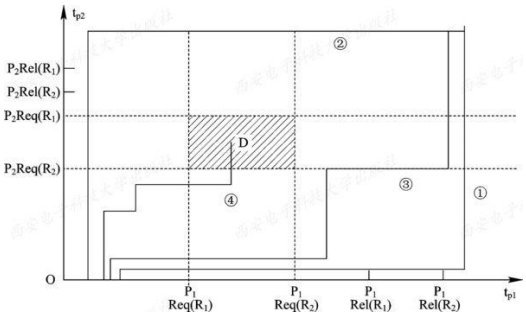
1）竞争不可抢占性资源引起死锁



2) 竞争可消耗资源引起死锁



3) 进程推进顺序不当引起死锁



4.2 死锁的必要条件（互斥、部分分配、不可抢占、循环等待）

4.3 死锁的防止：破坏产生死锁的四个必要条件中的一个或几个，以避免发生死锁。

■ 破坏互斥条件，把独占型资源改造成共享型资源

■ 采用**剥夺式调度方法**可以破坏不可抢占条件

■ 破坏部分分配条件，一次性为进程分配所有应使用的资源。

■ 破坏循环等待条件，使运行期间不存在进程循环等待现象。

***主要方法：**

1. 静态分配法

■ 开始运行之前，必须一次性地申请其在整个运行过程中所需的全部资源。破坏了部分分配条件。

2. 层次分配法

■ 在层次分配策略下，资源被分成多个层次；

■ 一个进程得到某一层的一个资源后，他只能在申请在较高层次的资源；

■ 当一个进程要释放一个资源时，必须先释放所占用的较高层次的资源；

■ 当一个进程获得了某一层的一个进程后，它想再申请改层中的另一个资源，那么，必须先释放改层中的已占有资源。

■ 不可能形成循环回路，故阻止了循环等待的出现。

4.4 死锁的避免（银行家算法：借钱给有偿还能力的客户）

1.银行家算法（单一资源）

假设系统有三个进程 P，Q，R，系统只有一类资源共 10 个，目前分配情况如下：

进程	已占资源	还需申请数
P	4	4
Q	2	2
R	2	7

安全序列：{Q，P，R}

2.银行家算法（多个资源）

假定系统中有五个进程{P_0, P_1, P_2, P_3, P_4}和三类资源{A, B, C}，各种资源的数量分别为 10、5、7，在 T0 时刻的资源分配情况如图所示，给出一个安全序列。

资源 / 情况 / 进程	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	7	5	3	0	1	0	7	4	3	3	3	2
										(2	3	0)
P_1	3	2	2	2	0	0	1	2	2			
				(3	0	2)	(0	2	0)			
P_2	9	0	2	3	0	2	6	0	0			
P_3	2	2	2	2	1	1	0	1	1			
P_4	4	3	3	0	0	2	4	3	1			

(1)从题目中，可以提取出 Max 矩阵和 Allocation，用二个矩阵相减得到 Need 矩阵：

$$\underset{\text{Max}}{\begin{pmatrix}7&5&3\\3&2&2\\9&0&2\\2&2&2\\4&3&3\end{pmatrix}} - \underset{\text{Allocation}}{\begin{pmatrix}0&1&0\\2&0&0\\3&0&2\\2&1&1\\0&0&2\end{pmatrix}} = \underset{\text{Need}}{\begin{pmatrix}7&4&3\\1&2&2\\6&0&0\\0&1&1\\4&3&1\end{pmatrix}}$$

(2)用 Avaliable 向量与 Need 矩阵各行比较，找出比 Avaliable 向量小的行：

（3　3　2）>(1　2　2)和（3　3　2）>(0　1　1)

对应的二个进程分别为 P1 和 P3，我们选择 P1 加入安全序列。

(3)释放 P1 所占有的资源，即把 P1 进程对应的 Allocation 矩阵中的一行与 Avaliable 向量相加：

（3　3　2）+（2　0　0）=（5　3　2）= *Avaliable*

(4)在 Need 矩阵中去掉 P1 对应的行

P0	7	4	3
P2	6	0	0
P3	0	1	1
P4	4	3	1

(5)重复第（2）步，最后得到一个安全序列：{P1，P3，P4，P2，P0}

故系统是安全的。

4.5 死锁的检测与恢复

如果在系统中，既不采取死锁预防措施，也未配有死锁避免算法，系统很可能会发生死锁。在这种情况下，系统应当提供两个算法：

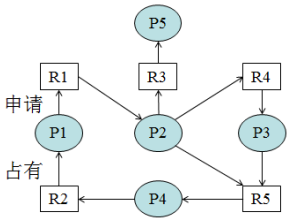
① 死锁检测算法。该方法用于检测系统状态，以确定系统中是否发生了死锁。

② 死锁解除算法。当认定系统中已发生了死锁，利用该算法可将系统从死锁状态中解脱出来。

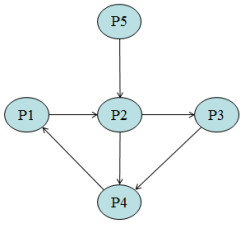
1. 死锁的检测

为了能对系统中是否已发生了死锁进行检测，在系统中必须：① 保存有关资源的请求和分配信息；② 提供一种算法，它利用这些信息来检测系统是否已进入死锁状态。

1) 资源分配图



2) 等待图：



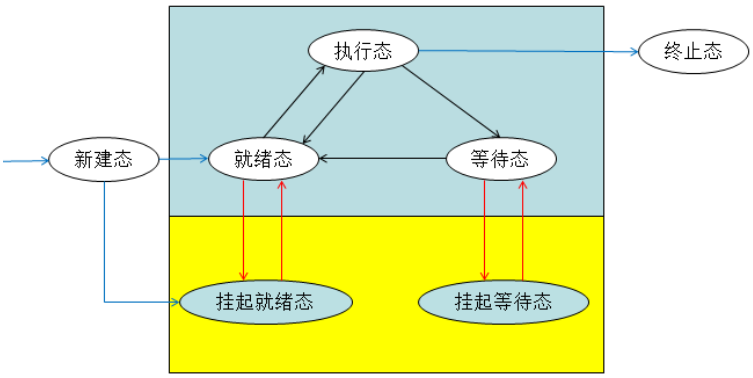
2. 死锁的解除

1)终止进程的方法：终止所有死锁进程、逐个终止进程

2)校验点：执行过程中定时设置校验点。从校验点开始重新执行。

四、处理机调度

1.处理机调度层次



黑色：低级调度 蓝色：高级调度 红色：中级调度

①高级调度(High Level Scheduling)：又称长程调度、作业调度，决定能否加入到执行的进程池中

②低级调度(Low Level Scheduling)：又称短程调度、进程调度，决定哪个可用进程占用处理器

③中级调度(Intermediate Scheduling)：又称为平衡负载调度，决定主存中的可用进程集合

2.调度准则

(1) 资源利用率。

$$\text{CPU 的利用率} = \frac{\text{CPU有效工作时间}}{\text{CPU有效工作时间}+\text{CPU空闲等待时间}}$$

(2) 公平性，诸进程获得合理的 CPU 时间，不发生进程饥饿现象

(3) 平衡性，系统中的 CPU 和各种外部设备都能经常处于忙碌状态

(4) 响应时间短 (5)周转时间小 (6)吞吐量多

3.短程调度算法

■ 先来先服务(first-come first-served，FCFS)调度算法

非抢夺式，平均等待时间比较长，不适合分时和实时系统。

■ 短作业优先(short job first，SJF)调度算法

缺点：

(1)须预知作业运行时间(2)对长作业非常不利(3)无法实现人机交互

(4)完全未考虑作业紧迫程度,不能保证紧迫性作业能得到及时处理。

■ 轮转调度算法(RR)

高响应比优先调度算法(Highest Response Ratio Next，HRRN)

$$\text{优先权} = \frac{\text{等待时间} + \text{要求服务时间}}{\text{要求服务时间}}$$

$$R_P = \frac{\text{等待时间}+\text{要求服务时间}}{\text{要求服务时间}} = \frac{\text{响应时间}}{\text{要求服务时间}}$$

4.优先级调度算法，优先级的类型

（1）静态优先级：静态优先级是在创建进程时确定的，在进程的整个运行期间保持不变。优先级是利用某一范围内的一个整数来表示的，例如 0～255 中的某一整数，又把该整数称为优先数。确定进程优先级大小的依据有如下三个：进程类型、进程对资源的需求、用户要求

（2）动态优先级：动态优先级是指在创建进程之初，先赋予其一个优先级，然后其值随进程的推进或等待时间的增加而改变，以便获得更好的调度性能。动态优先级根据进程占用 CPU 的长短、进程等待 CPU 的长短来确定。

■ 优先级调度算法缺点与解决办法

缺点：无穷阻塞或饥饿现象，某个低优先级进程无穷等待 CPU。

解决方法：老化技术，逐渐增加等待很长时间的进程的优先级。

5.多级反馈队列轮换调度算法

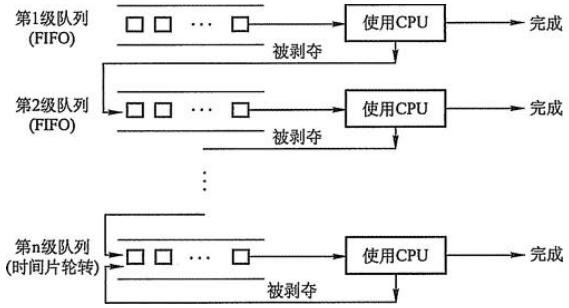
■ 调度机制

多级反馈队列调度算法的调度机制可描述如下：

(1) 设置多个就绪队列。

(2) 每个队列都采用 FCFS 算法。

(3) 按队列优先级调度。



■ 算法性能

如果规定第一个队列的时间片略大于多数人机交互所需之处理时间时，便能较好地满足各种类型用户的需要。

6.实时调度

■ 实现实时调度的基本条件：提供必要的信息、系统处理能力强、采用抢占式调度机制、具有快速切换机制

■ 实时调度算法的分类：① 根据实时任务性质，可将实时调度的算法分为**硬实时调度算法**和**软实时调度算法**；② 按调度方式，则可分为**非抢占调度算法**和**抢占调度算法**。

■ 非抢占式调度算法有非抢占式轮转调度算法和优先调度算法

■ 抢占式调度算法

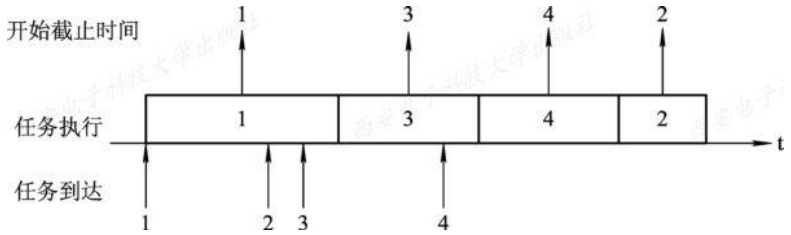
可根据抢占发生时间的不同而进一步分成以下两种调度算法：

(1) 基于时钟中断的抢占式优先级调度算法。

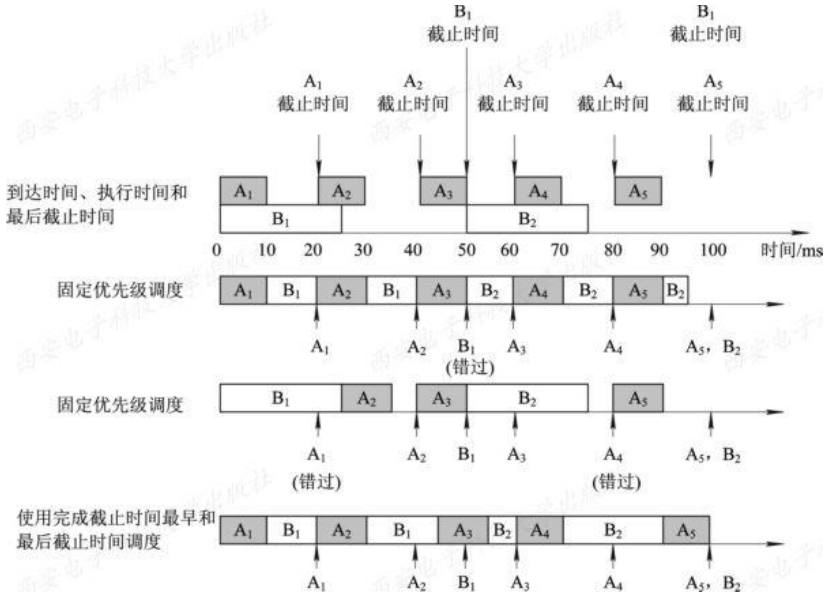
(2) 立即抢占(Immediate Preemption)的优先级调度算法。

6.1 最早截止时间优先 EDF(Earliest Deadline First)算法

(1)非抢占式调度方式用于非周期实时任务



(2) 抢占式调度方式用于周期实时任务



6.2 最低松弛度优先 LLF(Least Laxity First)算法

该算法在确定任务的优先级时，根据的是任务的紧急(或松弛)程度。任务紧急程度愈高，赋予该任务的优先级就愈高，以使之优先执行。

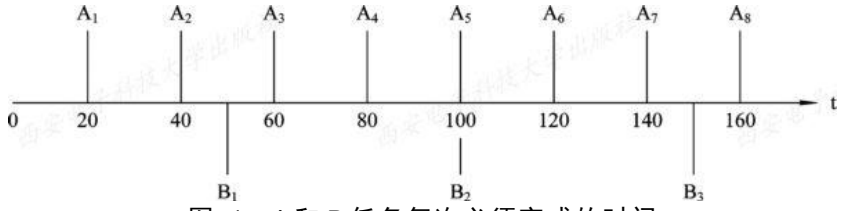


图 1 A 和 B 任务每次必须完成的时间

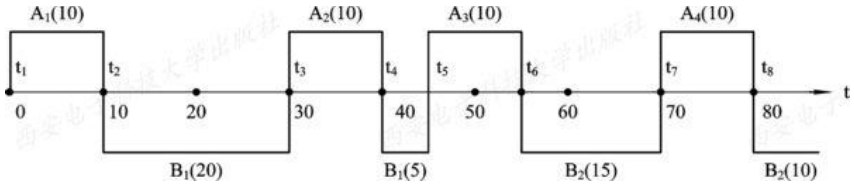
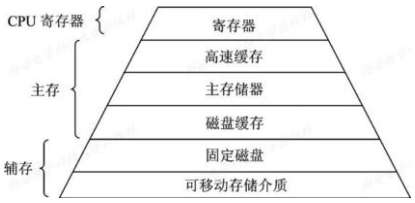


图 2 利用 ELLF 算法进行调度的情况

五、存储器管理

1.存储器的层次结构



■ 多层结构存储器系统:通用计算机而言，存储层次至少应具有三级：**最高层为 CPU 寄存器，中间为主存，最底层是辅存。**

■ 存储管理的任务和功能

1.主要任务：为多道程序的并发提供良好的环境、提高存储器的利用率、逻辑上扩充主存空间、方便用户使用存储器。

2.存储管理必须具备的功能：存储空间的分配和回收、地址映射、存储共享与保护、主存扩充。

2.存储分配形式

存储分配解决多道程序之间共享主存的存储空间。

1.直接存储分配方式：程序员使用存储器的**物理地址**进行编程，以确保各程序之间互不重叠。

缺点：用户编程不方便；存储器的利用率不高。

2.静态存储分配方式：程序员使用存储器的**逻辑地址**进行编程，当连接程序对它们进行装入、连接时，才确定它们在主存中的**物理地址**，从而产生可执行程序。这种分配方式在进行装入、连接时，要求系统必须分配其要求的全部存储空间，否则不能装入该用户程序。一旦装入，直到程序结束时才释放。

缺点：存储器不能有效共享；不能实现对存储器的动态扩展。

3.动态存储分配方式：现代操作系统通常采用的存储管理方法。动态存储分配通常可采用覆盖和交换技术实现。

特点：在进行装入、连接时，不要求一次性将整个程序装入主存中，可根据执行需要一部分一部分的装入；同时，已装入主存的程序不在执行时，系统可以回收该程序占据的存储空间。

3.重定位：将逻辑地址转换称物理地址。

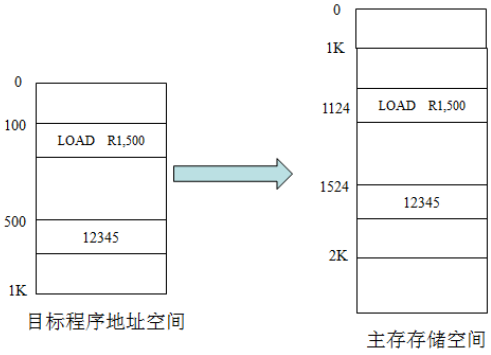
■ 地址空间和存储空间：源程序经过编译或是汇编之后，产生了目标程序，而编译程序总是从 0 号地址单元开始，为目标程序指令顺序分配地址，我们称为相对地址或逻辑地址，这些地址的集合称为地址空间。**存储空间**就是指主存中一系列存储信息的物理单元的集合，这些物理单元的编号称为物理地址或绝对地址。

■ 重定位：

(1)静态地址重定位：指用户程序在装入是由装配程序一次完成，即地址变换只是在装入时一次完成，以后不在改变。

特点：实现简单。

不足：必须分配一个连续的存储空间；难以实现程序和数据共享。

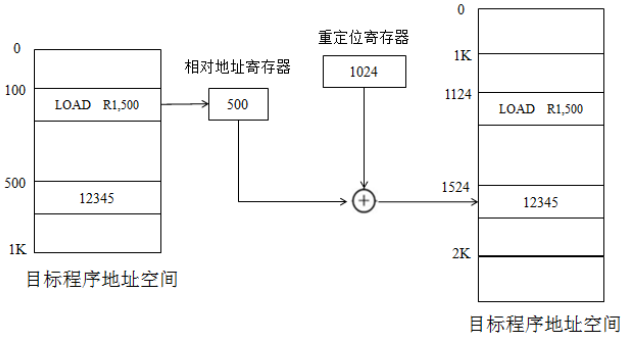


(2)动态地址重定位

必须有硬件地址变换机构的支持，即须在系统中增设一个**重定位寄存器**，用它来存放程序(数据)在内存中的起始地址。程序在执行时，真正访问的内存地址是**相对地址**与**重定位寄存器**中的地址相加而形成的。

特点：1.执行时程序可以在主存中移动，对于移动后的程序，只需将新的起始地址写入重定位寄存器即可。2.利于程序段共享。3.为实现虚拟存储管理提供了基础。

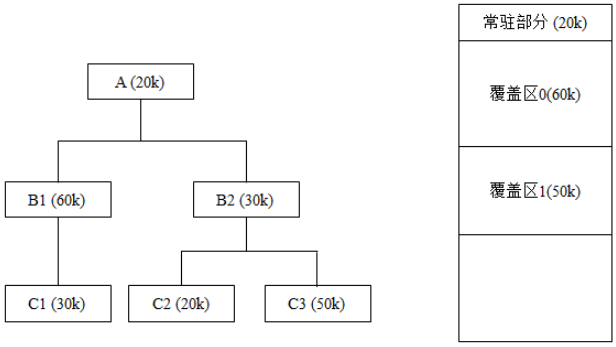
缺点：1.实现存储管理的软件比较复杂。2.需要硬件支持。



4.覆盖与交换

覆盖与交换技术是从逻辑上扩充主存的二种方法。

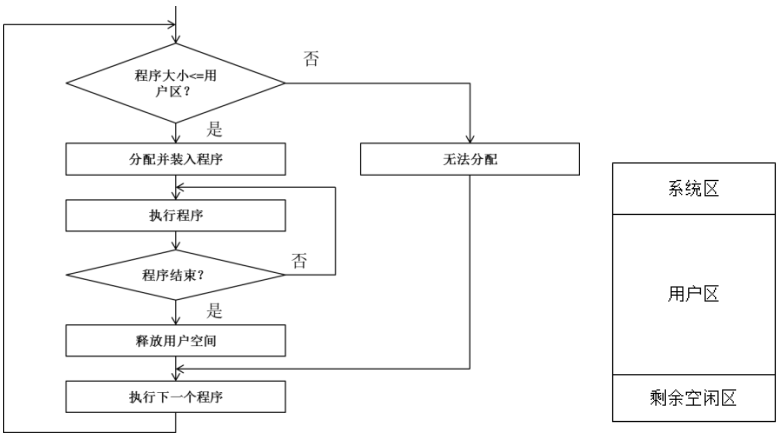
1.覆盖技术：把程序划分成若干功能相互独立的程序段，并且让那些不会同时被 CPU 执行的程序段共享同一主存区。通常这些程序段保存在外存中，当 CPU 要求某一程序段执行时，才将该程序段调入主存并覆盖某程序段。从用户的角度看，主存扩大了。



2.交换技术：也称为对换技术，最早用于麻省理工学院的单用户分时系统 CTSS 中。由于当时计算机的内存都非常小，为了使该系统能分时运行多个用户程序而引入了交换技术。系统把所有的用户作业存放在磁盘上，每次只能调入一个作业进入内存，当该作业的一个时间片用完时，将它调至外存的后备队列上等待，再从后备队列上将另一个作业调入内存。这就是最早出现的分时系统中所用的交换技术。
交换分为如下两类：(1) 整体交换。(2) 页面(分段)交换。
3. 交换空间的分配与回收
由于交换分区的分配采用的是连续分配方式，因而交换空间的分配与回收与动态分区方式时的内存分配与回收方法雷同。其分配算法可以是首次适应算法、循环首次适应算法或最佳适应算法等。

3.单道环境下的存储管理

在单道程序环境下，存储器管理方式是把内存分为系统区、用户区和剩余空闲区 3 部分，系统区仅提供给 OS 使用，它通常是放在内存的低址部分。而在用户区内存中，仅装有一道用户程序，即整个内存的用户空间由该程序独占。这样的存储器分配方式被称为**单一连续分配**方式。
特点：1.管理简单，只需很少的软件硬件支持。2.便于用户了解使用。
缺点： 1. 存储空间浪费大。2. CPU 效率低。3. 程序和数据不能共享。4. I/O 设备利用率低。



4.分区存储管理

■ 固定分区法

1.分区方法：
(1) 分区大小相等(指所有的内存分区大小相等)。(2) 分区大小不等。
2. 内存分配：为了便于内存分配，通常将分区按其大小进行排队，并为之建立一张分区使用表，其中各表项包括每个分区的起始地址、大小及状态(是否已分配)

分区号	大小(KB)	起址(K)	状态
1	12	20	已分配
2	32	32	已分配
3	64	64	未分配
4	128	128	已分配

(a) 分区说明表

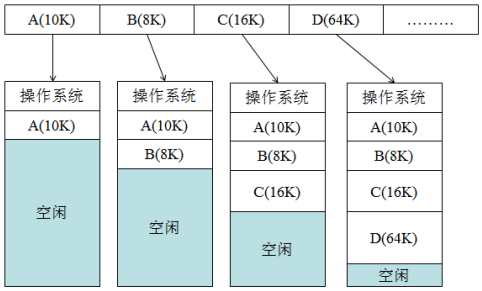
空间	操作系统
24 KB	作业 A
32 KB	作业 B
64 KB	作业 C
128 KB	
…	…
256 KB	

(b) 存储空间分配情况

3. 缺点：固定分区存储分配技术，分区的大小是在**系统初始化**时进行的。但用户作业占据的存储空间，不可能刚好等于某个分区的大小，所以在分配的分区中，通常都有一部分未被进程占用浪费的存储空间，我们称这部分空间“**碎片**”

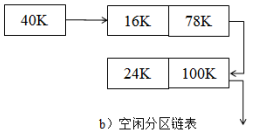
■ 动态分区法

1. 基本概念：采用动态分区分配方式，在系统启动时，除了操作系统常驻主存部分外，只存在一个空闲分区，分配程序将该依次划分给调度程序选中的进程，并且分配的大小可随用户进程对主存的要求而改变，这种分配方式不会产生“碎片”现象，从而大大提高的主存的利用率。



区号	分区长度	起始地址
1	16K	40K
3	24K	78K
5	9K	100K

a）空闲分区表



b）空闲分区链表

作业进程号	请求长度
P1	15K
P2	25K

c）请求表

2. 动态分区分配与回收

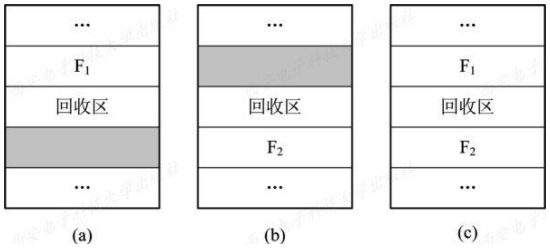
动态分区的分配方式：

（1）首次适应(first fit，FF)算法：分配第一个大小满足要求分区

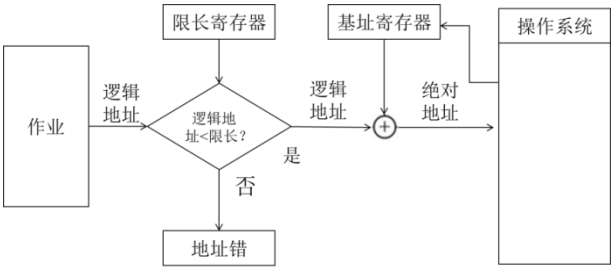
（2）最佳适应(best fit，BF)算法：分配满足要求的最小分区

（3）最坏适应(worst fit，WF)算法：与最佳适应算法相反，挑选一个最大空闲区，从中分一部分存储空间给作业使用

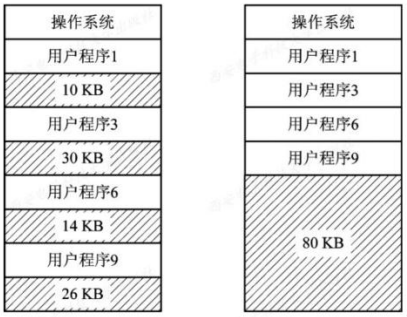
■ 动态分区的回收



■ 地址转换与存储保护



■ 移动技术



■ 分区存储管理优缺点

优点：

1. 实现了多道程序设计，从而提高了系统资源的利用率。
2. 系统要求的硬件支持少，管理简单，实现容易。

缺点：

1. 由于作业装入时的连续性，导致主存的利用率不高，采用移动技术可以提高主存的利用率，但增加了系统的开销。
2. 主存的扩充只能采用覆盖与交换技术，无法真正实现虚拟存储。

5.页式存储管理

■ 基本原理：

1. 将主存划分成多个大小相等的**页框**。
2. 受页架尺寸限制，程序的逻辑地址也自然分成**页**。
3. 不同的**页**可以放在不同的**页框**中，不需要连续。

■ 优越性：

1. 实现**连续存储**到非连续存储的飞跃，为实现虚拟存储打下了基础。
2. 解决了主存中的“**碎片**”问题。

■ 缺点：

（1）要求有硬件支持，如动态地址变换，缺页中断处理机构；

（2）必须提供相应的数据结构来管理存储器，而这些数据结构不仅占用了部分储存空间，同时它们的建立和管理也要花费 CPU 的时间。

（3）虽能解决了分区管理中区间的零头问题，但在页式存储管理系统中的页内的零头问题仍然存在。

（4）对于静态页式存储管理系统，用户作业要求一次性装入主存，将给用户作业的运行带来一定的限制。

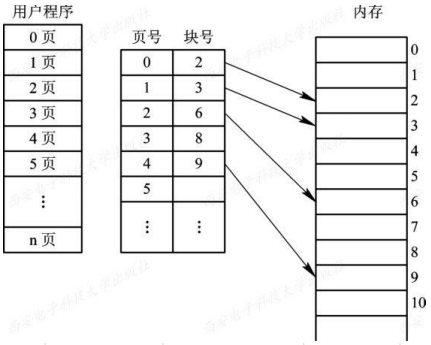
（5）在请求页式存储管理中，需要进行缺页中断处理，特别是请求调页的算法，若选择不当，还有可能出现“抖动”现象，增加了系统开销，降低了系统的效率。

■ 分类：静态页式存储管理、虚拟页式存储管理

■ 静态页式存储管理

用户作业**执行前**，将该作业的程序和数据**全部装入**到主存中，然后，操作系统通过页表和硬件地址变换机构实现逻辑地址到物理地址的转换。

5.1 主存页架的分配与回收



1、页表：在页式系统中，允许将进程的各个页离散地存储在内存的任一物理块中，为保证进程仍然能够正确地运行，即能在内存中找到每个页面所对应的物理块，系统又为每个进程建立了一张页面映像表，简称**页表**。

进程号	请求页面数	页表起始地址	页表长度	状态
1	20	1024	20	已分配
2	30	1044	30	已分配
3	21			未分配
…				

2、请求表：当系统有多个作业或进程时，系统必须知道每个作业或进程的页表起始地址和长度，才能进行主存分配和地址变换。

0	1	2	3	……	27	28	29	30	31
0	1	1	0	……	1	1	0	1	1
0	0	1	1	……	0	1	1	1	0
1	0	0	1	……	1	1	0	1	0

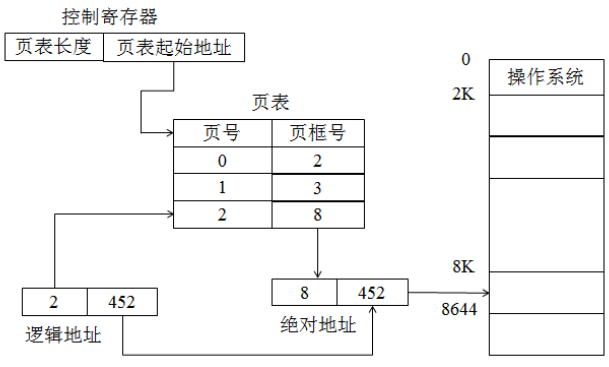
3、存储页框表：描述主存空间的分配情况，页框表指出了各页框是否已分配，以及未被分配的页框总数。

4、页框分配与回收算法：

1）分配：首先，从请求表中查出作业和进程数要求的页框数，然后由存储页框表检查是否有足够的空闲页框，若没有，则本次无法分配；如果有，则分配并设置页表，并填写请求表中的相应表项（页表起址、页表长度和状态），之后，再按一定的查询算法搜索出所要求的空闲页框，并将对应的页框号填入页表中。

2）回收：页框的回收算法也较为简单，当进程执行完毕时，根据进程页表中登记的页框号，将这些页框插入到存储页框表中，使之成为空闲页框，最后拆除该进程所对应的页表即可。

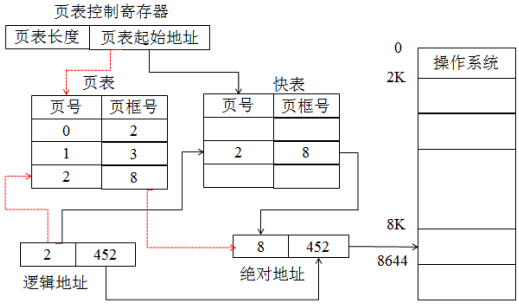
5.2 页式地址变换



进程在运行期间，需要对程序和数据的地址进行变换，即将用户地址空间中的逻辑地址变换为内存空间中的物理地址，由于它执行的频率非常高，每条指令的地址都需要进行变换，因此需要采用硬件来实现。页表功能是由一组专门的寄存器来实现的。一个页表项用一个寄存器。

5.3 快表

为提高地址转换速度，设置一个专用的高速存储器（Cache），用来存放页表的一部分，我们称这部分页表为**快表**。



有效访问时间(Effective Access Time，EAT)：假设 λ 表示查找快表所需时间，a 表示命中率，t 表示访问内存所需时间。

那么只用页表：EAT = t + t = 2t

引入快表后：EAT=a×λ+(t+λ)(1—a)+t=2t+λ—t×a

5.4 虚拟页式存储器

1. 常规存储管理方式的特征

(1) **一次性**：在作业运行前，要求将全部的内容一次性装入主存。

(2) **驻留性**：作业装入主存后，一直占据主存的部分空间，一直要等作业运行结束。

2. 局部性原理

(1) **时间局限性**。表现在如果程序中某一条指令一旦执行，则在不久以后还可能被继续执行，同样，若某一个数据被访问后不久，还可能被继续访问，其典型的情况是程序中存在着大量的**循环**。

(2) **空间局限性**。表现在如果程序访问了某一个程序单元，其附近的存储单元则不久也会被访问，即程序在一段时间内访问的地址，可能集中在一定的范围内，其典型的情况是程序**顺序**执行。

3. 虚拟存储器的基本思想

基于局部性原理可知，应用程序在运行之前没有必要将之全部装入内存，而仅须将那些当前要运行的少数页面或段先装入内存便可运行，其余部分暂留在盘上。

4. 虚拟存储器的特征

(1) **多次性**。用户程序在运行前并不是一次将全部内容装入到内存中，而是在程序的运行过程中，系统不断的对程序和数据部分的调入调出，完成程序的多处装入工作。

(2) **对换性**。程序在运行期间，允许将暂时不用的程序和数据调出主存（**换出**），放入外存的对换区中，但以后需要时再将他调了主存（**换入**），这便是虚拟存储器的换入、换出操作。

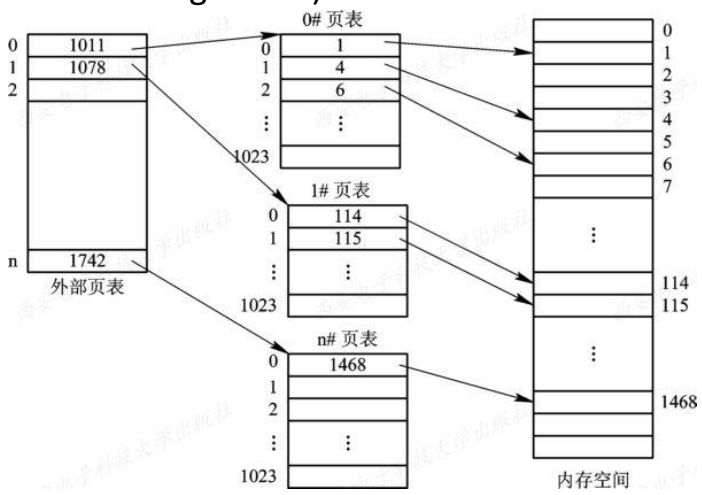
5.4 页式存储管理技术实现虚拟存储器

1.技术支持

(1)**硬件**支持：请求分页的页表机制、缺页中断机构、地址变换机构

(2)实现请求分页的**软件**

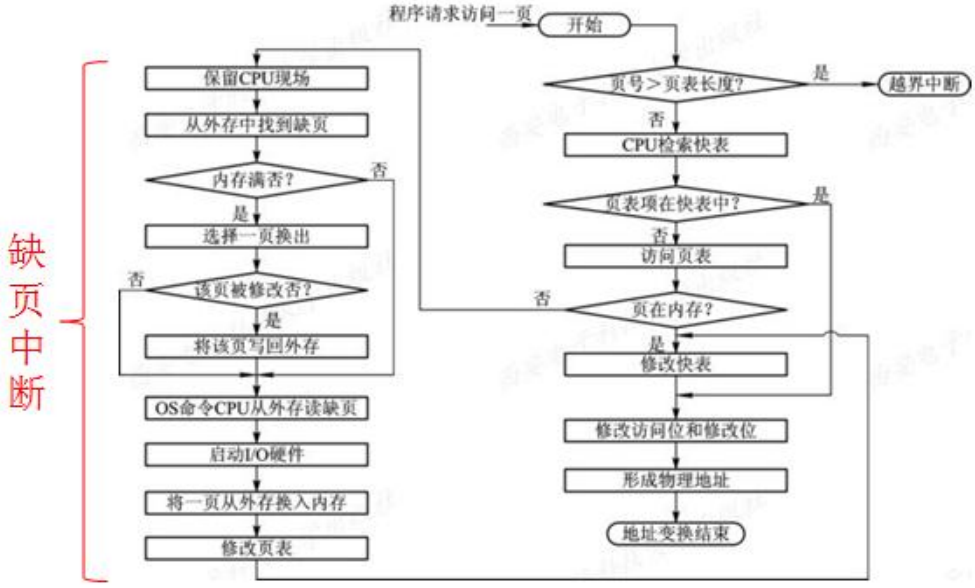
2.两级页表(Two-Level Page Table)



3.多级页表：对于 32 位的机器，采用两级页表结构是**合适**的，但对于 64 位及以上的机器，可能需要使用多级页表结构。

4.缺页中断机构：在页式虚拟存储系统中，每当要访问的页面不在内存时，便产生一个**缺页中断**，请求操作系统将所缺的页调入内存，此时应将缺页的进程阻塞，如果内存中有空闲块，则分配一个块，将要调入的页装入该块，并修改页表中相应的页表项，若此时内存中没有空闲块，只要淘汰某页。**缺页中断**与一般中断区别在于：(1) 在指令执行期间产生和处理中断信号。(2) 一条指令在执行期间可能产生多次缺页中断

5.地址变换机构：地址变换机构是在页式系统地址变换机构的基础上，为实现虚拟存储器，再增加了某些功能所形成的，如产生和处理缺页中断，以及从内存中换出一页的功能等等。



5.5 页面置换

1.调页策略

(1) 预调页策略

预调页策略的优点在于，当在外存上查找一页时需经历较长时间，如果进程的许多页是存放在外存的一个连续区域中，一次调入若干相邻的页，要比每次调入一页更高效。

(2) 请求调页策略

当进程运行中需要访问某部分程序和数据，而其所在页面又不在主存时，立即提出请求，由系统将其所需页面调入主存，请求调页策略比较容易实现，故在目前的页式虚拟存储系统当中，大多采用此策略，也称之为请求分页系统。

2.分配策略

在请求分页系统中，可采取两种内存分配策略，即固定和**可变**分配策略。在进行置换时，也可采取两种策略，即全局置换和局部置换。于是可组合出以下三种适用的策略。

1) 固定分配局部置换(Fixed Allocation，Local Replacement)

2) 可变分配全局置换(Variable Allocation，Global Replacement)

3) 可变分配局部置换(Variable Allocation，Local Replacement)

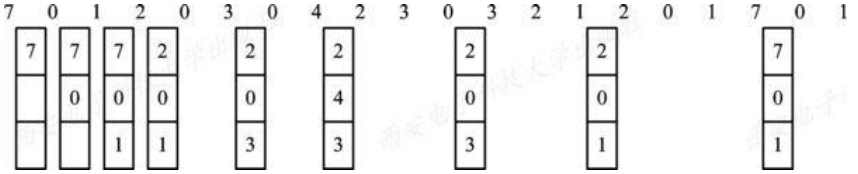
3.缺页率

假设一个进程的逻辑空间为 n 页，系统为其分配的内存物理块数为 m(m≤n)。如果在进程的运行过程中，访问页面成功(即所访问页面在内存中)的次数为 S，访问页面失败(即所访问页面不在内存中，需要从外存调入)的次数为 F，则该进程总的页面访问次数为 A = S + F，那么该进程在其运行过程中的缺页率即为：

$$f = \frac{F}{A}$$

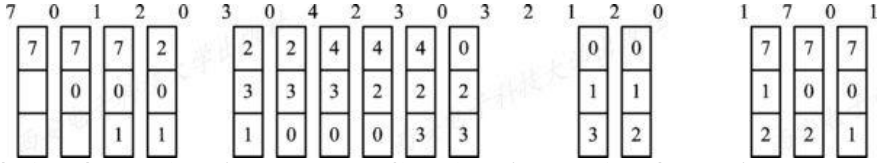
4. 页面置换算法

（1）最佳(Optimal)置换算法：最佳置换算法是由 Belady 于 1966 年提出的一种理论上的算法。其所选择的被**淘汰页面将是以后永不使用的**，或许是在**最长(未来)时间内不再被访问**的页面。采用最佳置换算法通常可保证获得最低的缺页率。但由于人们目前还无法预知，一个进程在内存的若干个页面中，哪一个页面是未来最长时间内不再被访问的，因而该算法是无法实现的，但可以利用该算法去评价其它算法。



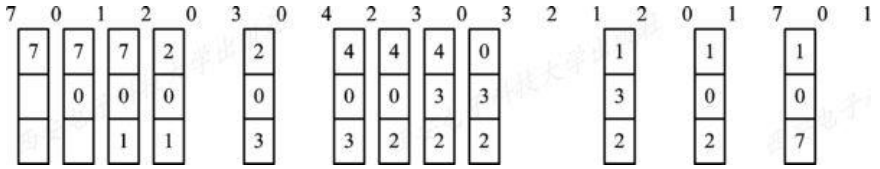
发生缺页中断的次数为 9，页面置换的次数为 6，缺页中断率为：9/20=45%

（2）先进先出(FIFO)页面置换算法：该算法总是淘汰最先进入内存的页面，即选择在内存中**驻留时间最久的页面**予以淘汰。



发生缺页中断的次数为 15，页面置换的次数为 12，缺页中断率为：15/20=75%

（3） 最近最久未使用置换算法 LRU

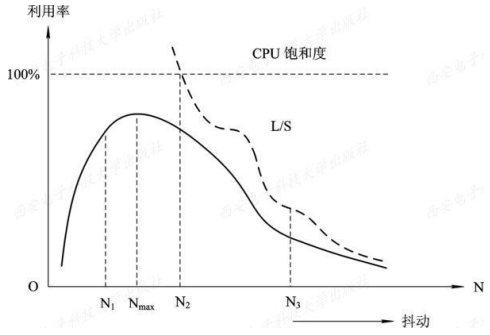


发生缺页中断的次数为 12，页面置换的次数为 9，缺页中断率为：12/20=60%

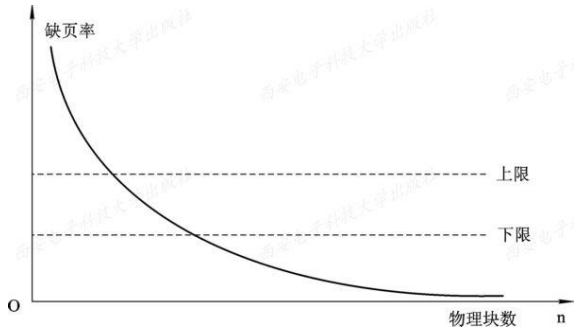
5.6 虚拟页式存储管理系统的性能分析

（1）多道程序度与处理机利用率

由于虚拟存储器系统能从逻辑上扩大内存，这时，只需装入一个进程的部分程序和数据便可开始运行，故人们希望在系统中能运行更多的进程，即增加多道程序度，以提高处理机的利用率。



（2）产生“抖动”的原因：同时在系统中运行的进程太多，由此分配给每一个进程的物理块太少，不能满足进程正常运行的基本要求，致使每个进程在运行时，频繁地出现缺页，必须请求系统将所缺之页调入内存。这会使得在系统中排队等待页面调进/调出的进程数目增加。显然，对磁盘的有效访问时间也随之急剧增加，造成每个进程的大部分时间都用于页面的换进/换出，而几乎不能再去做任何有效的工作，从而导致发生处理机的利用率急剧下降并趋于 0 的情况。我们称此时的进程是处于“抖动”状态。



（3）“抖动”的预防方法

采取局部置换策略：在页面分配和置换策略中，如果采取的是可变分配方式，则为了预防发生“抖动”，可采取局部置换策略。

5.6 段式及段页式存储管理

分页管理方式是从计算机的角度考虑设计的，以提高内存的利用率，提升计算机的性能，且分页通过硬件机制实现，对用户完全透明。

分段管理方式的提出则是考虑用户和程序员，以满足方便编程，信息保护和共享，动态增长以及动态链接等方多方面的需要。

■ 段式存储管理

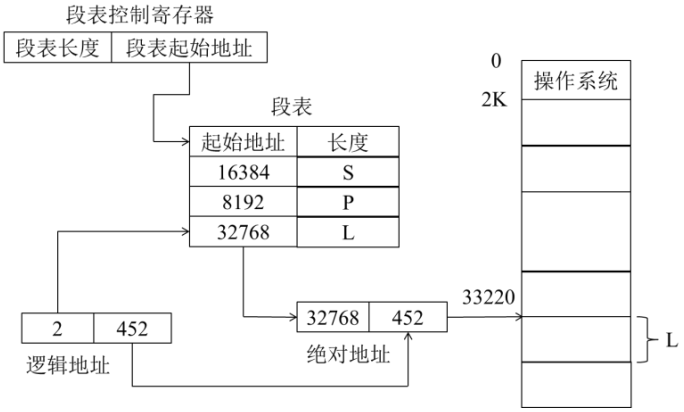
1. 分段

分段地址中的地址具有如下结构：

2. 段表

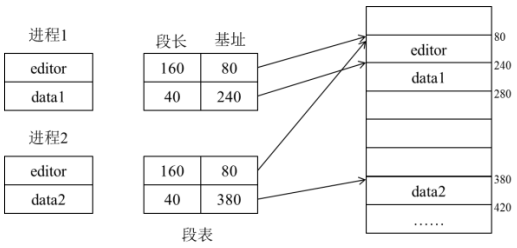
段号	段长	本段在主存中的起始地址

3. 地址变换机构



4. 信息共享和保护

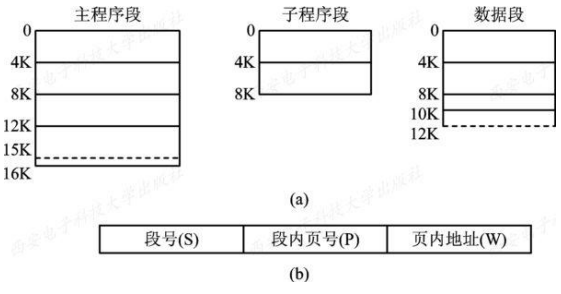
在分段系统中，由于是以段为基本单位的，不管该段有多大，我们都只需为该段设置一个段表项，因此使实现共享变得非常容易。我们以共享 editor 为例，此时只需在(每个)进程 1 和进程 2 的段表中，为文本编辑程序设置一个段表项，让段表项中基址(80)指向 editor 程序在内存的起始地址。



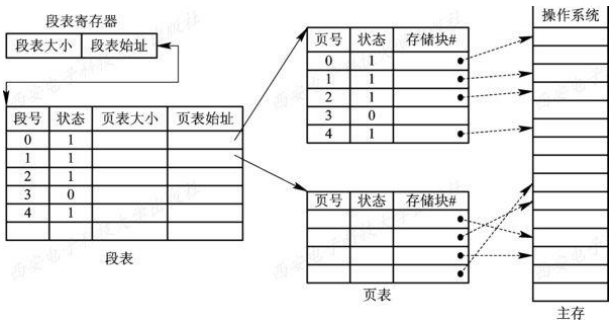
5、段页式虚拟存储管理

页式存储管理能有效的提高内存利用率，而段式存储管理能反映程序的逻辑结构，并有利于段的共享，如果将这两种存储管理方法结合起来，就形成了**段页式存储管理**方式。

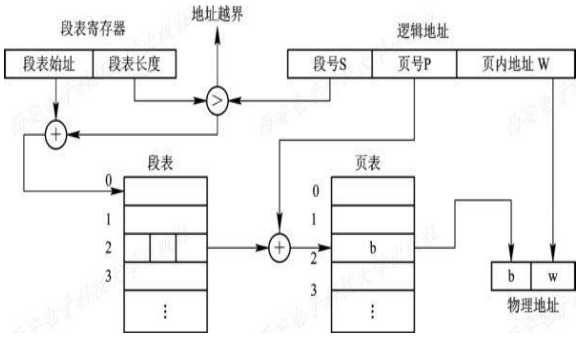
■ 基本原理：



■ 地址映射：



■ 地址变换过程：



■ 分页管理和管理的比较分段

	分页	分段
目的	页是信息的**物理单位**，分页是为实现离散分配方式，以消除内存的外零头，提高内存的利用率，或者说，分页仅仅是由于系统管理的需要，而不是用户的需要。	段是信息的**逻辑单位**，它含有一组意义相对完整的信息，分段的目的是为了能更好的满足用户的需要。
长度	页的大小固定且由系统决定，由系统把逻辑地址划为页号和页内地址两部分，是有机器硬件实现的，因而在系统中只能有一种大小的页面。	段的长度不固定，决定于用户所编写的程序，通常由编译程序对程序进行编译时，根据信息的性质来划分。
地址空间	作业地址空间是一维的，即单一的线性地址空间，程序员只需利用一个记忆符即可表示一个地址。	作业地址空间是二维的，程序员在标志一个地址时，即需给出段名，又需给出段内地址。

六、文件管理

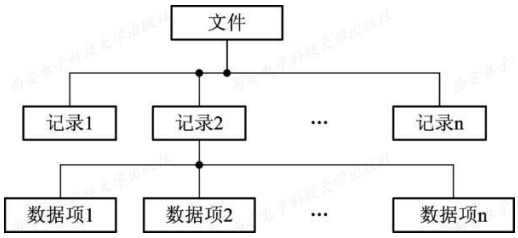
1.文件

文件系统的管理功能是将其管理的程序和数据通过组织为一系列文件的方式实现的。而**文件是指具有文件名的，在逻辑上具有完整意义的一组相关元素的集合**。

■ **数据项、记录和文件**

在文件系统中，数据项是最低级的数据组织形式，可把它分成以下两种类型：基本数据项、组合数据项。记录是一组相关数据项的集合，用于描述一个对象在某方面的属性。一个记录应包含哪些数据项，取决于需要描述对象的哪个方面。

由于对象所处的环境不同可把他作为不同的对象。文件是指由创建者所定义的、具有文件名的一组相关元素的集合，可分为有结构文件和无结构文件两种。



■ 文件命名：由文件名和扩展名构成

文件名 MS-DOS 由 1-8 个字符组成;WINDOWS 由 1-255 个字符组成（支持**长文件名**），**扩展名**一般由 1-3 个字符组成。

■ **文件类型**

1) 按用途分类：系统文件、用户文件、库文件
2) 按文件中数据的形式分类：源文件、目标文件、可执行文件
3) 按存取控制属性分类：只执行文件、只读文件、读写文件
4) 按组织形式和处理方式分类：普通文件、目录文件、特殊文件

■ 文件属性

除了文件内容外还包含文件属性信息：名称、类型、位置、大小、保护、时间日期等

2.文件操作：

■ 存取方法：顺序存取方法、直接存取方法（又称**随机存取方法**）、索引存取方法

■ 基本操作：创建文件、删除文件、读文件、写文件。

■ 打开和关闭操作

■ 其它文件操作

3.文件结构

■ 文件的逻辑结构：无结构的**流式文件**、有结构的**记录式文件**

■ 选取文件逻辑结构遵循的基本要求：

（1）有助于提高对文件的检索速度

（2）方便对文件进行修改

（3）尽量减少文件占用的存储空间，不要求大片的连续存储空间

■ 根据文件的组织方式，可把有结构文件分为三类：顺序文件、索引文件、索引顺序文件。

■ 顺序文件

顺序文件的最佳应用场合是在对文件中的记录进行批量存取时(即每次要读或写一大批记录)。所有逻辑文件中顺序文件的存取效率是最高的。此外，对于顺序存储设备(如磁带)，也只有顺序文件才能被存储并能有效地工作。

■ 索引文件

（1）按关键字建立索引

（2）具有多个索引表的索引文件：为每种可能成为检索条件的域(属性或关键字)都配置一张索引表。

■ 索引顺序文件

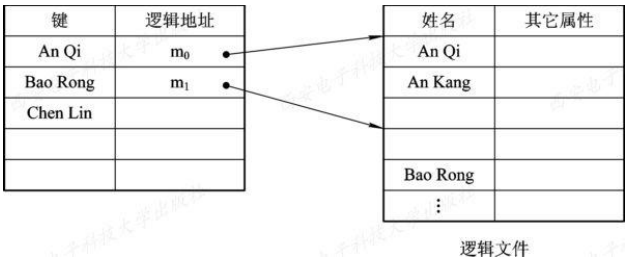
索引顺序文件是对顺序文件的一种改进，它基本上克服了变长记录的顺序文件不能随机访问，以及不便于记录的删除和插入的缺点。但它仍保留了顺序文件的关键特征，即记录是按关键字的顺序组织起来的。它又增加了两个新特征：一个是引入了文件索引表，通过该表可以实现对索引顺序文件的随机访问；另一个是增加了溢出(overflow)文件，用它来记录新增加的、删除的和修改的记录。

（1）一级索引顺序文件

最简单的索引顺序文件只使用了一级索引。

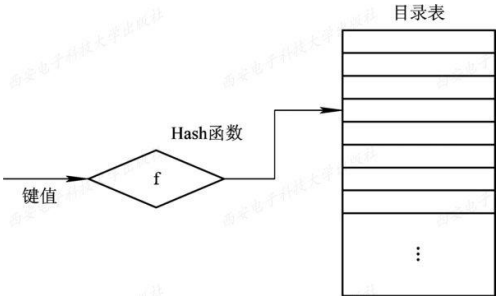
检索时间：

对于一个含有 N 个记录的顺序文件，查找某个关键字值的记录时平均需要查找 N/2 次。对于具有一级索引顺序文件，为能查找某个关键字值的记录时平均需要查找$\sqrt{N}$次。



（2）两级索引顺序文件：对于一个非常大的文件，为找到一个记录而须查找的记录数目仍然很多，为了进一步提高检索效率，可以为顺序文件建立多级索引，即为索引文件再建立一张索引表，从而形成两级索引表。

■ 直接文件：关键字决定了记录的物理地址。哈希(Hash)文件是目前应用最广的一种直接文件。



4.文件目录

■ 功能：实现按名存取、提高检索速度、文件共享、允许文件重名。

■ 操作：创建目录、删除目录、改变目录、移动目录、链接(Link)操作、查找

■ 文件目录内容：

（1）文件控制块 FCB(File Control Block)

文件目录在文件名与文件自身之间一种映射。为了能对文件进行正确的存取，必须提供用于描述和控制文件信息的数据结构。

文件名	扩展名	属性	备用	时间	日期	第一块号	盘块数

MS-DOS 的文件控制块

（2）索引结点：查找目录的过程中，须将存放目录文件的第一个盘块中的目录调入内存，然后将用户所给定的文件名，与目录项中的文件名逐一比较。因此，有的系统（UNIX，Linux）采用将文件名和文件描述信息分开，将文件描述信息单独形成一个称为**索引节点**的数据结构。

文件名	索引结点编号
文件名 1	
文件名 2	
…	…

0 13 14 15

UNIX 的文件目录

■ 文件目录结构

（1）单级文件目录

这是最简单的文件目录。在整个文件系统中只建立一张目录表，每个文件占一个目录项，目录项中含文件名、文件扩展名、文件长度、文件类型、文件物理地址以及其它文件属性。此外，为表明每个目录项是否空闲，又设置了一个状态位。

文件名	扩展名	文件长度	物理地址	文件类型	文件说明	状态位	
文件名 1							
文件名 2							
文件名 3							

单级文件目录

单级目录结构的特点：

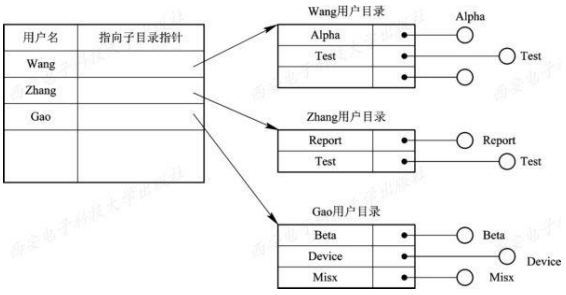
优点：结构简单，使用方便，易于实现。

缺点：查找速度慢、不允许重名、不便共享、不适合多用户操作系统

（2）两级文件目录

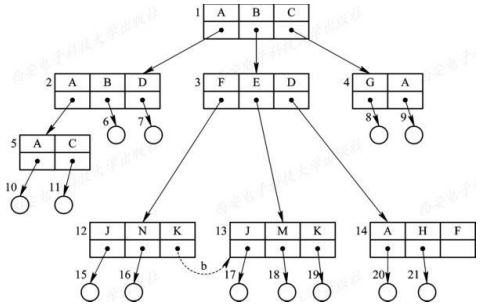
克服单级文件目录所存在的缺点，可以为每一个用户再建立一个单独的用户文件目录 UFD(User File Directory)。这些文件目录具有相似的结构，它由用户所有文件的文件控制块组成。

此外，在系统中再建立一个主文件目录 MFD(Master File Directory)；在主文件目录中，每个用户目录文件都占有一个目录项，其目录项中包括用户名和指向该用户目录文件的指针。



（3）树形多级结构目录(Tree-Structured Directory)

在现代 OS 中，最通用且实用的文件目录无疑是树形结构目录。它可以明显地提高对目录的检索速度和文件系统的性能。主目录在这里被称为根目录，在每个文件目录中，只能有一个根目录，每个文件和每个目录都只能有一个父目录。把数据文件称为树叶，其它的目录均作为树的结点，或称为子目录。



（4）无环图目录结构

严格的树形结构目录中，每个文件只允许有一个父目录，父目录可以有效地拥有该文件，其它用户要想访问它，必须经过其属主目录来访问该文件。这就是说，对文件的共享是不对称的，或者说，**树形结构目录是不适合文件共享的**。

■ 文件路径

（1）绝对路径

在树形结构目录中，从根目录到任何数据文件都只有一条唯一的通路。在该路径上，从树的根(即主目录)开始，把全部目录文件名与数据文件名依次地用“/”连接起来，即构成该数据文件唯一的路径名。

（2）相对路径

当进程访问局限于某个子目录时，可以设置一个“当前目录”，又称为“工作目录”。从当前目录开始，到达要访问的最终文件所进过的路径称为**相对路径**。

5.文件共享与保护

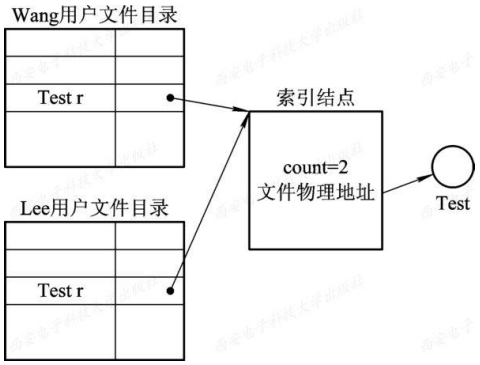
■ 利用符号链接(Symbolic Linking)实现共享

优点：在利用符号链方式实现文件共享时，**只是文件主才拥有指向其索引结点的指针；而共享该文件的其他用户则只有该文件的路径名，**并不拥有指向其索引结点的指针。这样就不会发生在文件主删除一共享文件后留下一悬空指针的情况。当文件的拥有者把一个共享文件删除后，如果其他用户又试图通过符号链去访问一个被删除的共享文件，

则会因系统找不到该文件而使访问失败，于是再将符号链删除，此时不会产生任何影响。

缺点：每次访问共享文件时，都可能要多次地读盘。这使每次访问文件的开销甚大，且增加了启动磁盘的频率。此外，要为每个共享用户建立一条符号链，而由于链本身实际上是一个文件，尽管该文件非常简单，却仍要为它配置一个索引结点，这也要耗费一定的磁盘空间。

■ 利用索引结点的共享方式



■ **文件保护：口令保护、加密保护、访问控制**

（1）访问类型：读、写、执行、添加、删除、列表、重命名、复制

（2）**访问控制**：用户身份认证，如 WINDOWS，数据库等身份认证。

（3）**口令**：用户在建立一个文件时提供一个口令，系统为其建立 FCB 时附上相应的口令，同时告诉共享该文件的其他用户。

（4）**密码**：用户对文件内容进行加密，文件被访问时需要使用密钥

6.文件存储空间管理

■ **空闲表法**

空闲表法属于**连续分配方式**，它与内存的动态分配方式雷同，它为每个文件分配一块连续的存储空间。即系统也为外存上的所有空闲区建立一张空闲表，每个空闲区对应于一个空闲表项，其中包括表项序号、该空闲区的第一个盘块号、该区的空闲盘块数等信息。再将所有空闲区按其起始盘块号递增的次序排列，形成空闲盘块表。

序号	第一空闲盘块号	空闲盘块数
1	2	4
2	9	3
3	15	5
4	—	—

存储空间的分配与回收：

空闲盘区的分配与内存的分区(动态)分配类似，同样是采用首次适应算法、最佳适应算法和最坏适应算法等，它们对存储空间的利用率大体相当。

■ 空闲链表法

1) 空闲盘块链：磁盘上的所有空闲空间以盘块为单位拉成一条链，其中的每一个盘块都有指向后继盘块的指针。**缺点：空闲盘块链可能很长，影响效率。**

2) 空闲盘区链：将磁盘上的所有空闲盘区(每个盘区可包含若干个盘块)拉成一条链。在每个盘区上除含有用于指示下一个空闲盘区的指针外，还应有能指明本盘区大小(盘块数)的信息。**缺点：分配与回收的过程较复杂。**

■ **位示图法**

利用二进制的一位来表示磁盘中一个盘块的使用情况。当其值为“0”时，表示对应的盘块空闲；为“1”时，表示已分配。有的系统把“0”作为盘块已分配的标志，把“1”作为空闲标志。(它们在本质上是相同的，都是用一位的两种状态来标志空闲和已分配两种情况。)磁盘上的所有盘块都有一个二进制位与之对应，这样，由所有盘块所对应的位构成一个集合，称为位示图。

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	1	0	0	0	1	1	1	0	0	1	0	0	1	1	0
2	0	0	0	1	1	1	1	1	1	0	0	0	0	1	1	1
3	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	0
4																
...																
16																

盘块分配时可分三步进行：

(1) 顺序扫描位示图，从中找出一个或一组其值为“0”的二进制位(“0”表示空闲时)。

(2) 将所找到的一个或一组二进制位转换成与之相应的盘块号。假定找到的其值为“0”的二进制位位于位示图的第 i 行、第 j 列，则其相应的盘块号应按下式计算：

$$b = n(i-1) + j$$

式中，n 代表每行的位数。

(3) 修改位示图，令 map[i, j] = 1。

盘块的回收分两步：

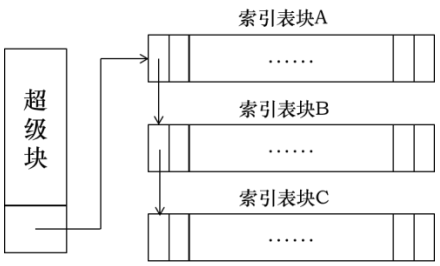
(1) 将回收盘块的盘块号转换成位示图中的行号和列号。转换公式为：

$$i = (b-1)\text{DIV}\ \ n+1$$
$$j = (b-1)\text{MOD}\ \ n+1$$

(2) 修改位示图。令 map[i, j] = 0。

■ **链接索引表法**

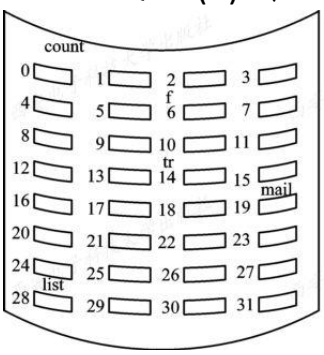
又称**成组链接法**。使用若干个空闲盘块作为索引表块，来指出存储空间中所有的空闲块。



7.文件分配

文件分配是指如何分配空白块来存放文件中的内容，也称为文件的**物理结构**。文件的物理结构直接与外存的组织方式有关。对于不同的外存组织方式，将形成不同的文件物理结构。目前常用的外存组织方式有： (1) 连续组织方式。 (2) 链接组织方式。 (3) 索引组织方式。

■ 连续分配：又称连续组织方式，要求为每一个文件分配一组相邻接的盘块。采用连续分配方式时，可把逻辑文件中的记录顺序地存储到邻接的各物理盘块中，这样所形成的文件结构称为顺序文件结构，物理文件称为顺序文件。



目录

file	start	length
count	0	2
tr	14	3
mail	19	3
list	28	3
f	6	2

优点：顺序访问容易、顺序访问速度快

缺点：要为文件分配连续存储空间、必须事先知道文件长度

不能灵活删除和插入记录、容易形成磁盘碎片

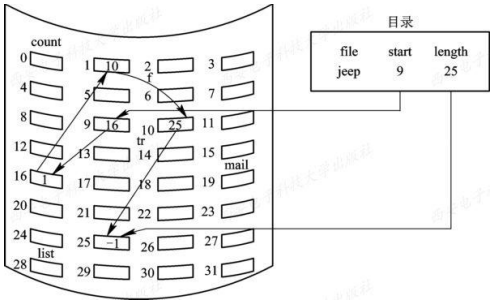
■ 链接分配：采用链接分配方式时，可为文件分配多个不连续的盘块，再通过每个盘块上的链接指针，将同属于一个文件的多个离散的盘块链接成一个链表，由此所形成的物理文件称为链接文件。

(1) 消除了磁盘的外部碎片，提高了外存的利用率。

(2) 对插入、删除和修改记录都非常容易。

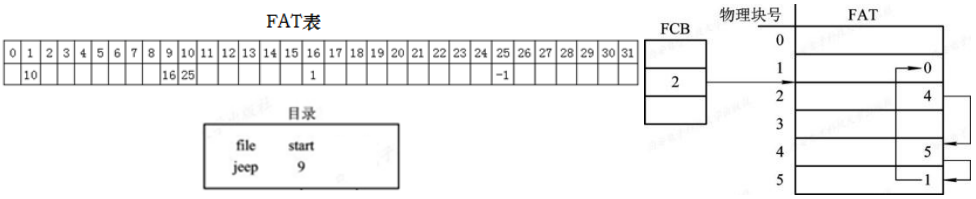
(3) 能适应文件的动态增长，无需事先知道文件的大小。

1. 隐式链接：在采用隐式链接组织方式时，在文件目录的每个目录项中，都须含有指向链接文件第一个盘块的指针和文件的长度。



2. 显式链接

把链接文件各物理块的指针显式地存放在内存的一张链接表中。该表在整个磁盘中仅设置一张（FAT 表）。



隐式链接和显式链接的比较：

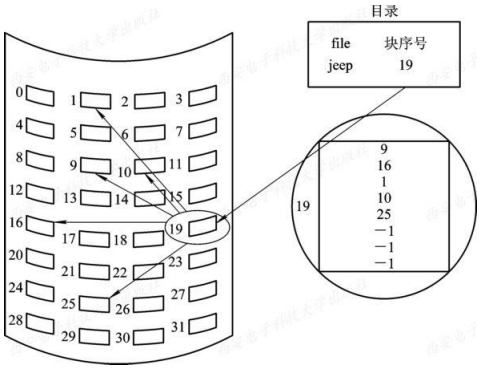
隐式链接指针在磁盘上，每次都要读取磁盘。

显式链接将磁盘上文件的链式结构用内存的一张 **FAT 表**去描述，查找时可以在内存中进行，定位后再读磁盘。速度比隐式快很多。

■ **索引链接分配**

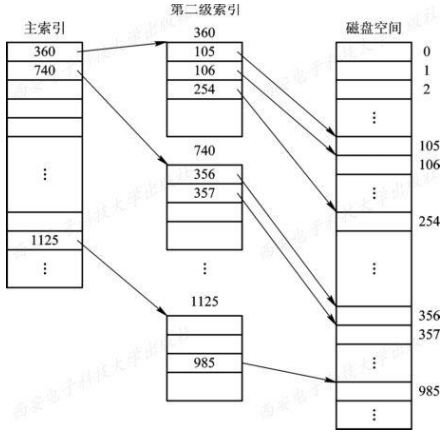
（1）一级索引链接分配

链接组织方式虽然解决了连续组织方式所存在的问题(即不便于随机访问)，但又出现了另外两个问题，即：① 不能支持高效的直接存取，要对一个较大的文件进行存取，须在 FAT 中顺序地查找许多盘块号;② FAT 需占用较大的内存空间，由于一个文件所占用盘块的盘块号是随机地分布在 FAT 中的，因而只有将整个 FAT 调入内存，才能保证在 FAT 中找到一个文件的所有盘块号。



（2）多级索引链接分配

在为一个大文件分配磁盘空间时，如果所分配出去的盘块的盘块号已经装满一个索引块时，OS 须再为该文件分配另一个索引块，用于将以后继续为之分配的盘块号记录于其中。依此类推，再通过链指针将各索引块按序链接起来。

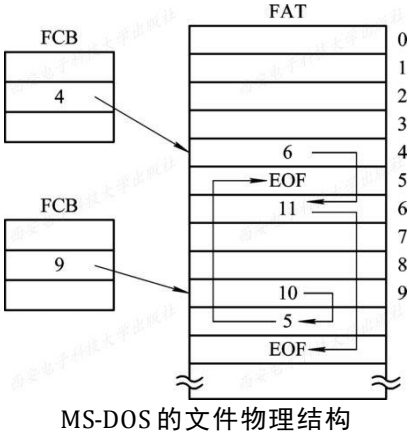


8.Windows 文件管理

■ FAT12

1) 早期的 FAT12 文件系统

FAT12 是以盘块为基本分配单位的。由于 FAT 是文件系统中最重要的数据结构，为了安全起见，在每个分区中都配有两张相同的文件分配表 FAT1 和 FAT2。在 FAT 的每个表项中存放下一个盘块号，它实际上是用于盘块之间的链接的指针，通过它可以将一个文件的所有的盘块链接起来，而将文件的第一个盘块号放在自己的 FCB 中。



MS-DOS 的文件物理结构

2) 以簇为单位的 FAT12 文件系统

如果把每个盘块(扇区)的容量增大 n 倍，则磁盘的最大容量便可增加 n 倍。但要增加盘块的容量是不方便和不灵活的。为此，引入了簇(cluster)的概念。

■ FAT16

FAT12 对磁盘容量限制的原因在于， FAT12 表中的表项有限制，亦即最多只允许 4096 个。这样，随着磁盘容量的增加，必定会引起簇的大小和簇内碎片也随之增加。

■ FAT32

由于 FAT16 表长只有 65 535 项，随着磁盘容量的增加，簇的大小也必然会随之增加，为了减少簇内零，也就应当增加 FAT 表的长度，为此需要再增加 FAT 表的宽度，这样也就由 FAT16 演变为 FAT32。

块大小	FAT12	FAT16	FAT32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

FAT 中簇的大小与最大分区的对应关系

■ NTFS

NTFS(New Technology File System)是一个专门为 Windows NT 开发的、全新的文件系统，并适用于 Windows 2000/XP 及后续的 Windows OS。

（1）磁盘组织：

NTFS 是以簇作为磁盘空间分配和回收的基本单位的。一个文件占用若干个簇，一个簇只属于一个文件。这样，在为文件分配磁盘空间时，就无须知道盘块的大小，只要根据不同的磁盘容量，选择相应大小的簇，即使 NTFS 具有了与磁盘物理块大小无关的独立性。

（2）文件的组织：

在 NTFS 中，以卷为单位，将一个卷中的所有文件信息、目录信息以及可用的未分配空间信息，都以文件记录的方式记录在一张主控文件表 MFT(Master File Table)中，该表是 NTFS 卷结构的中心，从逻辑上讲，卷中的每个文件作为一条记录，在 MFT 表中占有一行，其中还包括 MFT 自己的这一行。每行大小固定为 1 KB，每行称为该行所对应文件的元数据(metadata)，也称为文件控制字。

七、IO 管理

1.基本概念

■ I/O 设备，又称输入输出设备、外围设备、外部设备、外设。用于计算机系统与外部世界(如用户、其他计算机设备)信息交换或存储。

■ I/O 操作：内存与外设间的信息传递操作。它影响计算机系统的通用性和可扩展性，影响计算机系统综合处理能力及性价比的重要因素。

2.I/O 管理目标与功能

■ **I/O 管理目标：**

（1）为用户提供方便、统一的界面。

方便：为用户屏蔽具体设备的复杂物理特性；

统一：对不同设备尽量使用统一的操作方式。

（2）提高资源的利用率。提高 CPU 和 I/O 设备之间、设备与设备之间的并行操作程度，以提高它们的利用率

■ **I/O 管理功能：**

（1）设备控制。由设备处理程序完成。

目前对 I/O 设备有四种控制方式：中断技术、直接存储器访问(DMA)、通道技术和缓冲技术。

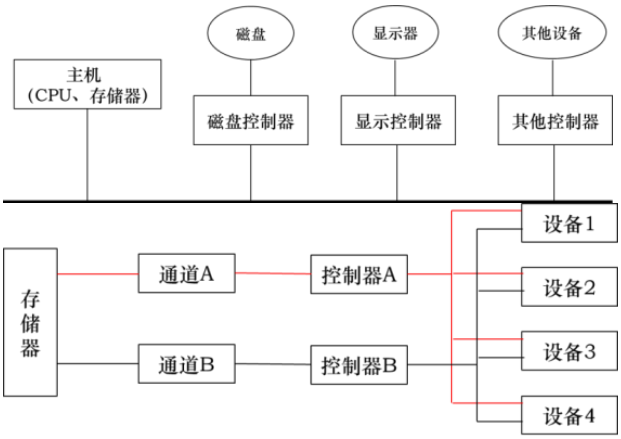
（2）设备的分配与回收。由设备的分配与回收程序完成。

（3）其他功能。

3.IO 系统

由 I/O 设备及其接口线路、控制部件、通道和管理软件组成。

总线型 I/O 系统：



通道型 I/O 系统：

■ **I/O 设备：**

按设备的从属关系分类：系统设备、用户设备

按设备共享属性分类：独占设备、共享设备、虚拟设备

按设备传输速率分类：低速设备、中速设备、高速设备

按设备信息交换单位分类：字符(字节流)设备、块设备

按设备使用特性分类：人机交互类设备、存储设备、网络通信设备

■ **设备控制器：**

设备控制器是 CPU 和 I/O 设备之间的接口，它接收从 CPU 发来的指令，然后去控制 I/O 设备工作，使 CPU 不需要直接进行设备控制，从而可以更高效地工作。

功能：接收和识别命令、数据交换、获取设备的状态、地址识别

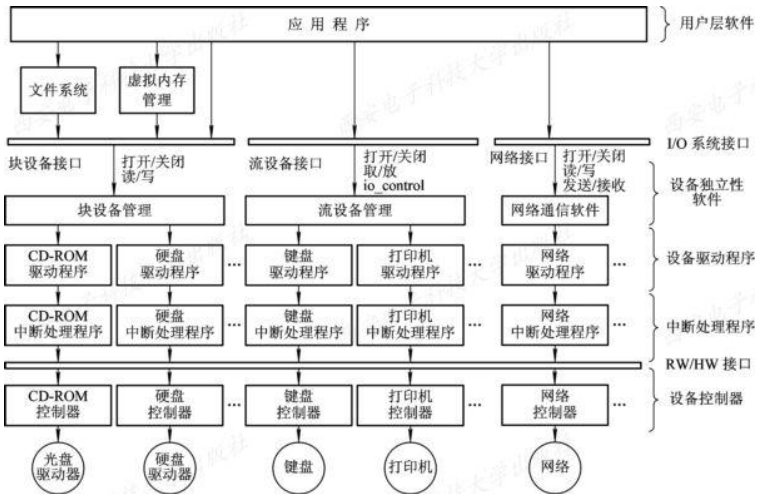
组成：与 CPU 的接口、I/O 控制逻辑、与设备的接口

■ 通道：

将原 CPU 处理的 I/O 任务由通道来承担，从而解放 CPU。实际上，通道就是一种特殊的 CPU，具有执行 I/O 指令的能力。

通道种类：字节多路通道(Byte Multiplexor Channel) **、数据选择通道(Block Selector Channel)、数据多路通道(Block Multiplexor Channel)**

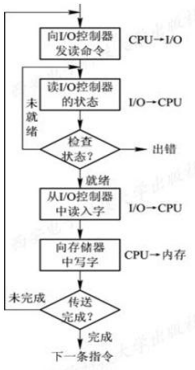
■ **I/O 接口**



4.I/O 控制方式

■ 程序直接控制方式

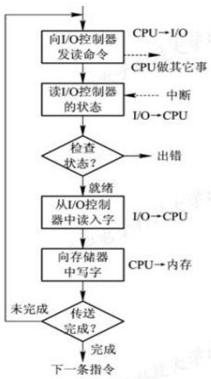
也称询问方式。，计算机从外部设备读取数据到存储器，每次读一个字节数据。对读入的每个字，CPU 需要对外设状态进行循环检查，直到确定该字已经在 I/O 控制器的寄存器中。



缺点：**CPU的高速与外设的低速之间的矛盾，导致 CPU 资源的浪费，效率低下。**

■ 中断控制方式

当某进程要启动某个 I/O 设备工作时，便由 CPU 向相应的**设备控制器**发出一条 I/O 命令，然后立即返回继续执行原来的任务。设备控制器于是按照该命令的要求去控制指定 I/O 设备。此时，CPU 与 I/O 设备并行操作。



■ DMA 控制方式,也称直接存储器访问方式。

虽然中断驱动 I/O 比程序 I/O 方式更有效，但它仍是以字(节)为单位进行 I/O 的。每当完成一个字(节)的 I/O 时，控制器便要向 CPU 请求一次中断。**在 I/O 设备与内存之间开辟直接的数据交换通路，彻底“解放”CPU。**

（1）DMA 方式的特点是:

✓ 数据传输的基本单位是数据块

✓ 所传送的数据是从设备直接送入内存的，或者相反

✓ 仅在传送一个或多个数据块的开始和结束时，才需 CPU 干预，整块数据的传送是在控制器的控制下完成的。DMA 方式较之中断驱动方式又进一步提高了 CPU 与 I/O 设备的并行操作程度。

（2）DMA 控制器的组成:

主机与 DMA 控制器的接口、DMA 控制器与块设备的接口、I/O 控制逻辑。

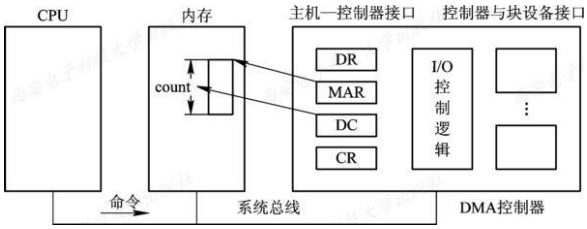
为实现主机与控制器之间的成块数据交换，必须在 DMA 控制器中设置以下四类寄存器:

CR：命令/状态寄存器

DC：数据寄存器

MAR：内存地址寄存器

DR：数据寄存器



DMA 控制器的组成

（3）与中断控制方式的区别:

中断控制方式：在每个数据需要传输时中断 CPU，数据传输是在中断处理时由 CPU 控制完成的。

DMA 控制方式:所要求传送的一批数据全部传送结束才中断 CPU，数据传输是在 DMA 控制下完成的。

■ 通道控制方式：**CPU 与通道高度并行工作**

(1)CPU 遇到 I/O 任务，组织通道程序，置**通道程序地址字 CAW**，启动指定通道。

(2)通道从 CAW 获得通道程序控制 I/O 设备操作,CPU 执行其他任务。

(3) I/O 操作完成后，I/O 通道发出中断，CPU 处理中断，并从**通道程序状态字 CSW** 获得通道执行情况，处理 I/O 操作。

5.缓冲区管理

■ 缓冲区的作用:

(1) 缓和 CPU 与 I/O 设备间速度不匹配的矛盾。

(2) 减少对 CPU 的中断频率，放宽对 CPU 中断响应时间的限制。

(3) 解决数据粒度不匹配的问题。

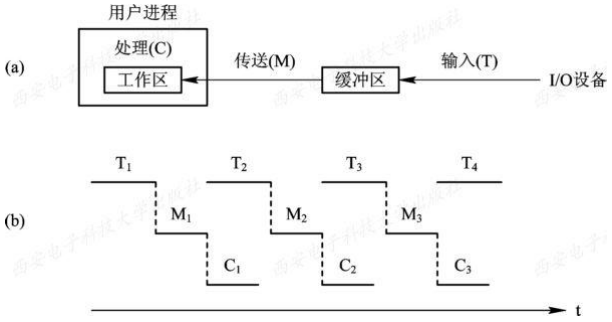
(4) 提高 CPU 和 I/O 设备之间的并行性。

■ 缓冲技术：

常用的缓冲技术有四种：单缓冲、双缓冲、循环缓冲和缓冲池。

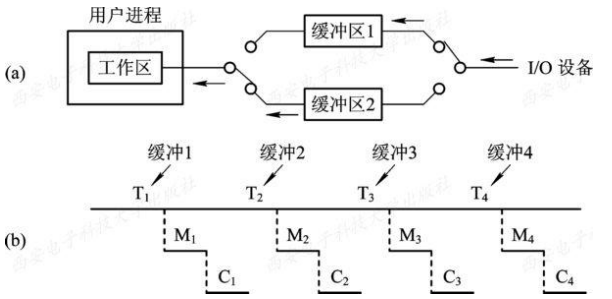
（1）**单缓冲**：每当用户进程发出一次 I/O 请求时，操作系统便在主存中为之分配一块缓冲区。单缓冲的处理时间：

假设 T>C，从初始状态开始，当工作区数据处理完后，时间为 C，缓冲区还没有冲满，当缓冲区充满时，经历了 T 时间，停止冲入数据；然后缓冲区向工作区传送数据，当缓冲区为空时，用时为 M，到达了下一个开始状态，因此，整个过程用时 T+M；假设 T<C，同理，整个过程用时 C+M。所以，单缓冲区的每次处理时间为 MAX(T,C)+M。

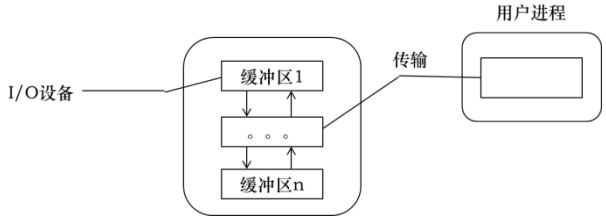


（2）**双缓冲区：**

双缓冲的处理时间：假设 T<C+M，整个过程用时 C+M；假设 T>C+M，整个过程用时 T。所以，双缓冲区的每次处理时间为 MAX(T, C+M)。



（3）**循环缓冲：**

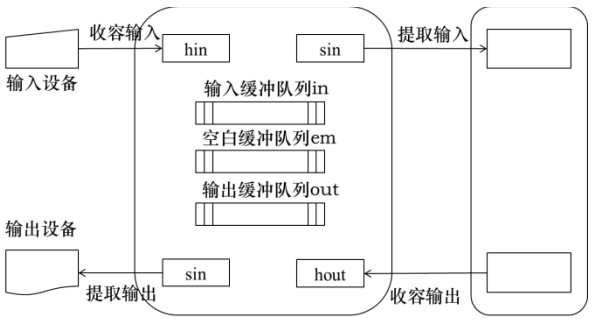


■ 缓冲池

（1）缓冲池结构

缓冲池管理着多个缓冲区，每个缓冲区由用于标识和管理的缓冲首部以及用于存放数据的缓冲体两部分组成。缓冲首部一般包括缓冲区号、设备号、设备上的数据块号、同步信号量以及队列链接指针等。为了管理上的方便，一般将缓冲池中具有相同类型的缓冲区链接成一个队列，于是可形成以下三个队列：空白缓冲队列 emq、输入队列 inq、输出队列 outq。

在缓冲池中还需要四种工作缓冲区：收容输入缓冲区(hin)、提取输入缓冲区(sin)、收容输出缓冲区(hout)、提取输出缓冲区(sout)。



（2）管理：进程申请缓冲区使用过程，进程将缓冲区放入队列过程
（3）缓冲区工作方式：收容输入、收容输出、提取输入、提取输出

6.设备驱动程序

设备驱动程序是 I/O 系统的高层与设备控制器之间的通信程序，其主要任务是**接收上层软件发来的抽象 I/O 要求，如 read 或 write 命令，再把它转换为具体要求后，发送给设备控制器，启动设备去执行；反之，它也将由设备控制器发来的信号传送给上层软件。**由于驱动程序与硬件密切相关，故通常应为每一类设备配置一种驱动程序。例如，打印机和显示器需要不同的驱动程序。

■ **设备驱动程序模式：**

(1) 设备驱动程序作为内核过程实现。
(2) 把设备驱动程序作为独立的进程来实现。

■ **设备驱动程序的功能：**

(1) 接收由与设备无关的软件发来的命令和参数，并将命令中的抽象要求转换为与设备相关的低层操作序列。
(2) 检查用户 I/O 请求的合法性，了解 I/O 设备的工作状态，传递与 I/O 设备操作有关的参数，设置设备的工作方式。
(3) 发出 I/O 命令，如果设备空闲，立即启动 I/O 设备，完成指定 I/O 操作；如果设备忙碌，则将请求者的请求块挂在设备队列上等待。
(4) 及时响应由设备控制器发来的中断请求，并根据其中断类型，调用相应的中断处理程序进行处理。

■ 设备驱动程序的特点：

(1) 驱动程序是实现在与设备无关的软件和设备控制器之间通信和转换的程序，具体说，它将抽象的 I/O 请求转换成具体的 I/O 操作后传送给控制器。又把控制器中所记录的设备状态和 I/O 操作完成情况，及时地反映给请求 I/O 的进程。
(2) 驱动程序与设备控制器以及 I/O 设备的硬件特性紧密相关，对于不同类型的设备，应配置不同的驱动程序。但可以为相同的多个终端设置一个终端驱动程序。
(3) 驱动程序与 I/O 设备所采用的 I/O 控制方式紧密相关，常用的 I/O 控制方式是中断驱动和 DMA 方式。
(4) 由于驱动程序与硬件紧密相关，因而其中的一部分必须用汇编语言书写。目前有很多驱动程序的基本部分已经固化在 ROM 中。
(5) 驱动程序应允许可重入。一个正在运行的驱动程序常会在一次调用完成前被再次调用。

■ **设备驱动程序的处理过程：**

设备驱动程序的主要任务是启动指定设备，完成上层指定的 I/O 工作。但在启动之前，应先完成必要的准备工作，如检测设备状态是

否为“忙”等。在完成所有的准备工作后，才向设备控制器发送一条启动命令。

7.设备分配：

设备分配与回收是设备管理功能之一，当进程向系统提出了 I/O 请求时，由**设备分配程序**负责，按一定策略分配**设备、控制器和通道**，形成一条数据传输通路，以供主机和 I/O 设备之间进行信息交换，在 I/O 完成时，再由系统回收分配相应的设备资源。

■ 分配原则：

(1)提高设备利用率，并避免死锁。

(2) 方便用户使用设备。

■ 分配方式：静态分配（不符合原则）、动态分配（可能造成死锁）

■ 考虑因素：

（1）设备的固有属性

设备的固有属性可分成三种，对它们应采取不同的分配策略：

独占设备的分配策略、共享设备的分配策略、虚拟设备的分配策略。

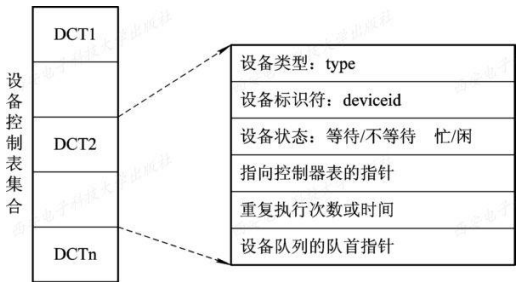
虚拟设备属于可共享的设备，可以将它同时分配给多个进程使用，需要引进 **Spooling 技术**。

（2）设备分配算法：先来先服务、优先级高者优先。

（3）设备分配中的安全性

■ **设备分配中的数据结构：**

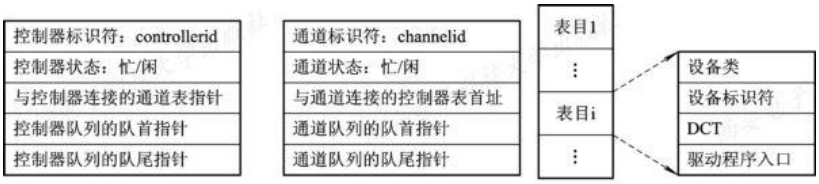
（1）设备控制表 DCT：系统为每一个设备都配置了一张设备控制表，用于记录设备的情况



（2）控制器控制表(COCT)：系统为每一个控制器都设置了用于记录控制器情况的控制器控制表

（3）通道控制表(CHCT)：每个通道都有一张通道控制表

（4）系统设备表(SDT)：这是系统范围的数据结构，记录了系统中全部设备的情况，每个设备占一个表目，其中包括有设备类型、设备标识符、设备控制表及设备驱动程序的入口等



(a) 控制器控制表(COCT) (b) 通道控制表(CHCT) (c) 系统设备表(SDT)

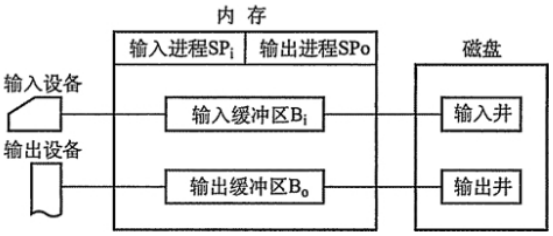
■ **I/O 设备分配的基本流程**

(1) 分配设备。根据 I/O 请求中的设备物理名查找系统设备表 SDT，从中找出该设备的 DCT，若 DCT 为空闲，则分配。

(2) 分配控制器。在 DCT 中查找与该设备链接的 COCT，若控制器状态为空闲，则分配给该进程。
(3) 分配通道。在 COCT 中查找与该控制器链接的 CHCT，若通道状态为空闲，则分配给该进程。
(4)若进程获得设备、控制器和通道，启动 I/O 设备进行数据传输。

■ **Spooling 技术**

利用专门的外围控制机，先将低速 I/O 设备上的数据传送到高速磁盘上，或者相反。这样当处理机需要输入数据时，便可以直接从磁盘中读取数据，极大地提高了输入速度。反之，在处理机需要输出数据时，也可以很快的速度把数据先输出到磁盘上，处理机便可去做自己的事情。



（1）**组成：**输入井和输出井、输入缓冲区和输出缓冲区、输入进程和输出进程、井管理程序。

（2）特点：提高 I/O 速度、将独占设备改造为共享设备、实现虚拟设备功能。