# 操作系统

## 第1章 操作系统概述

### 1.1操作系统的发展和分类

操作系统相关的技术经历了如下发展过程：手工操作阶段（无OS），脱机输入输出技术，批处理技术，多道程序设计技术。

多道程序运行的特点是：多道，宏观上并行，微观上串行。

并行性是指两个或两个以上事件在同一时刻发生，这要求在硬件上至少要有两个处理机。

并发性是指两个或两个以上事件在同一时间间隔内发生。

操作系统的分类

1.批处理操作系统

特征：脱机，多到和成批处理

2.分时操作系统

特征：多路性，交互性独占性，及时性

3.实时操作系统

分为两类：实时控制系统和实时信息处理系统

特征：及时响应和高可靠性

### 1.2操作系统的特征和功能

#### 1.2.1特征

抽象，共享，并发

#### 1.2.2功能

1.进程管理

在多道程序环境下，CPU的分配是以进程为基本单位的，因此对CPU的管理可以归结为对进程的管理。进程管理应实现下述主要功能：

（1）进程控制；（2）进程同步；（3）进程通信；（4）进程调度。

2.存储管理

存储管理应实现下述主要功能：

（1）内储存分配；（2）内存保护；（3）内存扩充。

3.文件管理

文件管理应实现下述主要功能：

（1）磁盘空间管理；（2）目录管理；（3）文件操作；（4）文件保护。

4.设备管理

计算机外部设备的管理是操作系统中最庞杂、琐碎的部分。设备管理应实现下述主要功能：

（1）设备分配；（2）设备传输控制；（3）设备独立性。

5.用户接口

为方便用户使用操作系统，操作系统还提供了用户接口。通常操作系统提供两种接口：（1）命令接口，如DOS命令行，Windows图形界面；（2）编程接口，如API。

## 第2章 进程管理

### 2.1进程的概念

进程是程序在处理机上的一次执行过程。

进程由以下几部分组成：（1）至少一个可执行程序；（2）一个独立的进程空间；（3）系统资源；（4）一个执行栈区；

程序、数据和栈的集合称为进程映像（Process Image）。

为了管理进程，操作系统为每一个进程定义了一个数据结构——进程控制块PCB（Process Control Block），用于记录进程的各种信息。

### 2.2进程状态的转换

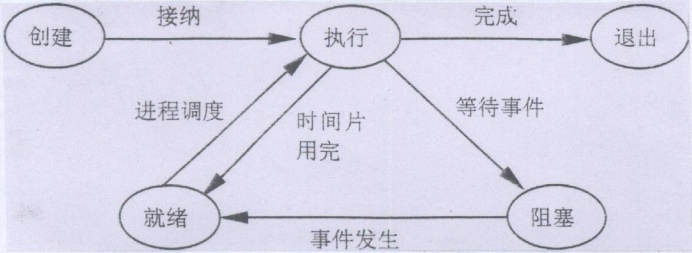


图2.1 三态转换

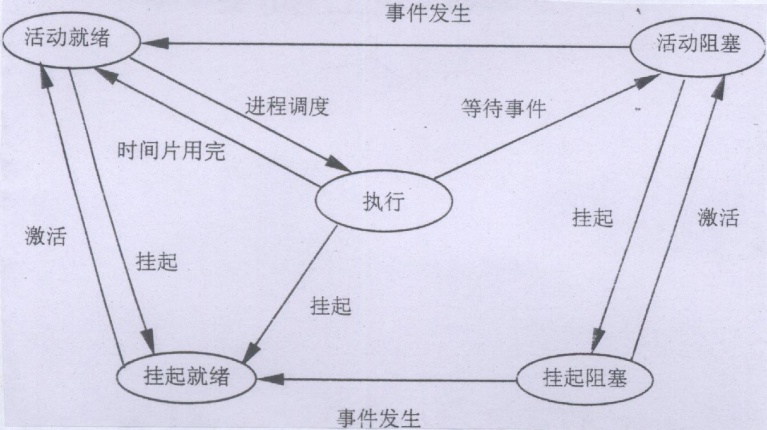


图2.2 五态转换

### 2.3线程

#### 2.3.1线程的引入

回顾一下进程的两个基本职能：

（1）进程是系统资源的分配单位；（2）进程是处理机的分配单位。

进程切换的开销比较大，为了提高操作系统的并发性能，不少操作系统研究者考虑将进程的两个基本属性分离开来，为此引入了线程，让线程去完成第二个基本职能，而进程只完成第一个职能。

#### 2.3.2线程的定义

线程是进程内一个相对独立的、可调度的执行单元。线程自己基本上不拥有资源，只用有一点在运行时必不可少的资源（如程序计数器、一组寄存器和栈），但它可以与同属一个进程的其他线程共享进程拥有的全部资源。

#### 2.3.2线程和进程的比较

（1）调度。线程是调度的单位，进程不是。

（2）拥有资源。进程是资源的分配单位，线程不拥有资源，但线程可以访问其所在进程的所有资源。

（3）并发性。引入线程后，不仅进程之间可以并发执行，而且同一进程内的多个线程之间也可以并发执行。从而使操作系统具有更好的并发性，大大提高了系统的吞吐量。

（4）系统开销。进程切换开销很大，而线程切换只需保存少量寄存器内容，开销很小。

### 2.4进程同步

#### 2.4.1同步与互斥的概念

同步关系：若某一进程收不到另一进程提供的信息就不能继续运行下去，这种制约关系的基本形式是“进程——进程”，是一种直接制约关系，称为同步关系。

互斥关系：若某一进程要求使用某一资源，而该资源正在被另一进程使用，并且该资源不允许两个进程同时使用，那么该进程只好等待。这种制约关系的基本形式是“进程——资源——进程”，是一种间接制约关系，称为互斥关系。

临界资源（Critical Resource, CR）：一次仅允许一个进程使用的资源。

临界段（Critical Section, CS）：各进程必须互斥执行的那种程序段，该程序段实施对临界资源的操作。

临界段应遵循的原则：空闲让进，忙则等待，有限等待，让权等待。

#### 2.4.2实现临界区的硬件方法

硬件方法的主要思想是用一条指令完成标志的检查和修改两个操作，因而保证了检查操作和修改操作不被打断，或通过中断屏蔽的方式来保证检查和修改作为一个整体执行。

**1.中断屏蔽方法**

CPU只有在发生中断时才会引起进程切换，因此简单而直接的方法是禁止一切中断发生，这样就能保证当前运行的进程能将临界段顺利执行完，从而保证了互斥的正确实现，然后在开中断。

下面是用中断屏蔽方法实现临界区的典型模式：

...

关中断

临界区

开中断

...

**2.硬件指令方法**

（1）“Test-and-Set”指令

bool Test\_and\_Set(bool \*lock)

{

bool old = \*lock;

\*lock = true;

return old;

}

临界段代码：

...

while(Test\_and\_Set()&lock);

critical section;

lock = false;

...

（2）“Swap”指令

void Swap(bool \*a, bool \*b)

{

bool tmp = \*a;

\*a = \*b;

\*b = tmp;

}

临界段代码：

...

key = true;

while(key){

Swap(&key, &lock);

}

critical section;

lock = false;

...

硬件方法的缺点是，进程在等待进入临阶段时需要耗费处理机时间，不能实现让权等待。

#### 2.4.3信号量

1965年，Dijkstra提出了一种称为信号量(semaphore)的同步互斥工具，称为信号量机构。信号量机构由信号量和P操作（又称为wait操作）、V操作（又称为signal操作）两部分组成。信号量（S）为一整型变量，表示资源的数目，只能被两个标准的原语操作访问。

P操作和V操作可以用如下函数描述：

struct semaphore{

int count;

queue\_t q;

};

P(semaphore \*s)

{

s->count--;

if(s->count < 0){

阻塞该进程;

将该进程插入阻塞队列s->q;

}

}

V(semaphore \*s)

{

s->count++;

if(s->count <= 0){

从阻塞队列s->q中取出第一个进程;

将该进程插入就绪队列;

}

}

#### 2.4.4四个经典的同步互斥问题

**1.生产-者消费者问题**

（1）将缓冲区当成一个整体

semaphore full = 0;

semaphore empty = N;

semaphore mutex = 1;

main()

{

producer();

consumer();

}

producer()

{

while(true){

生产一个产品;

P(empty);

p(mutex);

将产品放入缓冲区

V(mutex);

V(full);

}

consumer()

{

while(true){

P(full);

P(mutex);

从缓冲区取出一个产品;

V(mutex);

V(empty);

消费这个产品;

}

（2）将n个缓冲区当成一个循环队列

semaphore full = 0;

semaphore empty = N;

semaphore pmutex = 1; //生产者之间的互斥信号量

semaphore cmutex = 1; //消费者之间的互斥信号量

int in = 0, out = 0;

main()

{

producer();

consumer();

}

producer()

{

while(true){

生产一个产品;

P(empty);

p(pmutex);

将产品放入buffer[in];

in = (in + 1) % N;

V(pmutex);

V(full);

}

consumer()

{

while(true){

P(full);

P(cmutex);

从buffer[out]取出一个产品;

out = (out + 1) % N;

V(cmutex);

V(empty);

消费这个产品;

}

**2.读者-写者问题**

（1）读者优先

semaphore rmutex = 1;

semaphore wmutex = 1;

int reader\_count = 0;

main()

{

reader();

writer();

}

reader()

{

while(true){

P(rmutex);

if(reader\_count == 0) P(wmutex);

reader\_count ++;

V(rmutex);

读数据;

P(rmutex);

reader\_count--;

if(reader\_count == 0) V(wmutex);

V(rmutex);

}

}

writer()

{

while(true){

P(wmutex);

写数据;

V(wmutex);

}

}

（2）写者优先

semaphore rmutex = 1;

semaphore wmutex = 1;

semaphore s = 1; // 用于控制写者优先

int reader\_count = 0;

main()

{

reader();

writer();

}

reader()

{

while(true){

P(s);

P(rmutex);

if(reader\_count == 0) P(wmutex);

reader\_count ++;

V(rmutex);

V(s);

读数据;

P(rmutex);

reader\_count--;

if(reader\_count == 0) V(wmutex);

V(rmutex);

}

}

writer()

{

while(true){

P(s);

P(wmutex);

写数据;

V(wmutex);

V(s);

}

}

**3.理发师睡眠问题**

一个理发店由一个有N张椅子的等候室和一间理发室组成。若没有要理发的顾客，则理发师就去睡觉；若一个顾客走进理发店且所有的椅子都被占用了，则该顾客就离开理发店；若理发师正在为人理发，则该顾客就找一张空椅子坐下等待；若理发师在睡觉，则顾客就唤醒他。使用信号量设计一个协调理发师和顾客的程序。

【解】customers 用来记录等候的顾客数（不包括正在理发的顾客），barbers记录正在等候顾客的理发师数，其值为0或1，mutex用于互斥地访问count，count记录等候的顾客数，它实际上是customers的一份拷贝，之所以使用count是因为无法读取信号量的当前值。算法伪代码如下：

semaphore customers = 0; // 等候的顾客数

semaphore barbers = 0; // 等候顾客的理发师数

semaphore mutex = 1; // 用于理发师和顾客互斥访问 count

int count = 0; // 等候的顾客数，是customers的一份拷贝

main()

{

barbers();

customers();

}

barbers()

{

while(true){

P(customers); // 是否有等候的顾客

P(mutex);

count --; // 顾客数减 1

V(barbers); // 理发师开始理发

V(mutex);

理发;

}

}

customers()

{

P(mutex);

if(count < N){

count ++; // 若有空椅子则等候的顾客数加 1

V(customers);

V(mutex);

P(barbers);

理发;

}

else{

V(mutex); // 无空椅子则离开

}

}

**4.哲学家就餐问题**

semaphhore chopsticks[5] = {1, 1, 1, 1, 1};

main()

{

philosopher(0);

philosopher(1);

philosopher(2);

philosopher(3);

philosopher(4);

}

philosopher(int i)

{

while(true){

思考;

P(chopstick[i]);

P(chopstick[(i+1)%5]);

进餐;

V(chopstick[i]);

V(chopstick[(i+1)%5]);

}

}

上述算法可以保证不会有相邻的哲学家同时进餐，但有可能死锁。如果每个哲学家都同时拿起了自己左边的筷子，当他们试图去那右边的筷子时，都将因无筷子而无限期地等待下去。

### 2.5进程通信

共享存储方法：共享内存，共享文件

消息传递方法：消息传递，管道通信

### 2.6进程调度

#### 2.6.1调度的层次

高级调度，又称长程调度，作业调度，其主要任务是按一定的原则从外存上处于后备状态的作业中选择一个或多个，给它们分配内存、输入输出设备等必要的资源，并创建相应的进程。

中级调度，又称中程调度，交换调度，其主要任务是按一定的原则将处于外存对换区的重新又具备运行条件的进程调入内存，或将处于内存的暂时不能运行的进程交换到外存对换区。

低级调度，又称短程调度，进程调度，其主要任务是按某种策略从就绪队列中选择一个进程，将处理机分配给它。

#### 2.6.2进程调度算法

1.先来先服务（FCFS）算法

按照进入就绪队列的先后顺序进行调度，非剥夺方式

2.优先级调度算法

按进程的优先级进行调度，优先级有静态优先级和动态优先级，调度方式可以是剥夺方式和非剥夺方式。

3.时间片轮转调度算法

按照先来先服务原则，选择一个进程运行一段时间，当该进程用完这段时间后（即使未完成），系统将其送入就绪队列队尾。它相当于剥夺方式的先来先服务算法。

4.短进程优先调度算法

从就绪队列中选择那些所需运行时间（估计时间）最短的进程运行，非剥夺方式。

5.最高响应比优先调度算法

响应比=（等待时间+请求服务时间）/请求服务时间，非剥夺方式

6.多级反馈队列调度算法

本算法基于以下考虑：

（1）为提高系统吞吐量和降低进程平均等待时间而照顾短进程。

（2）为得到较好的输入/输出设备利用率和对交互用户的及时响应而照顾输入/输出型进程。

（3）在进程运行过程中，按进程运行情况来动态地考虑进程的性质（是输入/输出型还是计算型进程），同时进行相应的调度。

具体来说，系统中有多个进程就绪队列，每个就绪队列对应一个优先级。第1级队列的优先级最高，以下各级队列的优先级逐次降低。优先级最高的第1级队列中的进程的时间片最小，随着队列的级别增加，其进程的优先级降低了，但时间片却增加了。通常下放一级，其时间片增加1倍。各级队列均按先来先服务原则调度。

### 2.7死锁

#### 2.7.1死锁的必要条件

互斥，占有等待，非剥夺，循环等待

#### 2.7.2死锁处理策略

死锁预防，死锁避免，死锁检测及解除

#### 2.7.3死锁预防

要想防止死锁发生，只要破坏死锁的四个必要条件之一即可。

1.互斥

有些资源根本不能同时访问，例如打印机，因此破坏这个条件不太可能。

2.占有等待

（1）资源静态分配法，进程只有在申请到了所有资源后才能开始运行。

（2）进程提出新的资源请求但不能立刻得到满足时，必须主动释放所有已获得的资源。

3.非剥夺

允许操作系统剥夺进程所占有的资源

4.循环等待

资源按序分配法，其基本思想是将所有资源都按类型赋予一个编号，进程称按照编号递增的顺序申请资源，同类资源一次申请完。也就是说，进程提出请求资源Ri，则在以后的请求中，只能请求排在Ri后面的资源。

#### 2.7.4死锁避免

银行家算法

例2.1 在某时刻系统的资源分配情况如表2.1所示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Max | Allocation | Need | Available |
| R1 R2 R3 R4 | R1 R2 R3 R4 | R1 R2 R3 R4 | R1 R2 R3 R4 |
| P0  P1  P2  P3  P4 | 0 0 4 4  2 6 5 0  3 6 10 10  0 6 8 4  0 6 6 10 | 0 0 3 2  1 0 0 0  1 3 5 4  0 0 3 2  0 0 1 4 | 0 0 1 2  1 6 5 0  2 3 5 6  0 6 5 2  0 6 5 6 | 1 6 2 2 |

请问

（1）系统状态是否安全？

（2）若进程P2提出资源请求Rquest(1, 2, 2, 2)后，系统能否将资源分配给它？

（3）如果系统立即满足P2的上述要求，请问，系统是否立即进入死锁状态？

【解】

（1）使用安全性算法对上表的状态进行分析，如下表，找到一个安全序列{P0, P3, P1, P2}，故系统是安全的。

开始时刻的安全性检查

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Work | Need | Allocation | Work+Allocation | Finish |
| A B C D | A B C D | A B C D | A B C D |
| P0 | 1 6 2 2 | 0 0 1 2 | 0 0 1 2 | 1 6 5 4 | True |
| P3 | 1 6 5 4 | 0 6 5 2 | 0 0 3 2 | 1 6 8 6 | True |
| P4 | 1 6 8 6 | 0 6 5 6 | 0 0 1 4 | 1 6 9 10 | True |
| P1 | 1 6 9 10 | 1 6 5 0 | 1 0 0 0 | 2 6 9 10 | True |
| P2 | 2 6 9 10 | 2 3 5 6 | 1 3 5 4 | 3 9 14 14 | True |

（2）进程P2发出资源请求Rquest2(1, 2, 2, 2)后，按银行家算法进行检查：

① Rquest2(1, 2, 2, 2)≤Need2(2,3 ,5,6)；

② Rquest2(1, 2, 2, 2)≤Available2(1 ,6, 2, 2)；

③ 系统尝试满足P2的请求，并修改Allocation2，Need2和Available向量：

Allocation2 = Allocation2 + Rquest2 = (2, 5, 7, 6)；

Need2 = Need2 - Rquest2 = (1, 1, 3, 4)；

Available = Available - Rquest2 = (0, 4, 0, 0)；

④系统执行安全性算法，检查此次资源分配后，系统是否处于安全状态。

P2申请资源后的资源分配表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Max | Allocation | Need | Available |
| R1 R2 R3 R4 | R1 R2 R3 R4 | R1 R2 R3 R4 | R1 R2 R3 R4 |
| P0  P1  P2  P3  P4 | 0 0 4 4  2 6 5 0  3 6 10 10  0 6 8 4  0 6 6 10 | 0 0 3 2  1 0 0 0  1 3 5 4  0 0 3 2  0 0 1 4 | 0 0 1 2  1 6 5 0  1 1 3 4  0 6 5 2  0 6 5 6 | 0 4 0 0 |

此时对所有的进程，Available<Needi，即Available不能满足任何进程的请求，故系统进入不安全状态。因此系统不能将资源分配给P2。

（3）系统立即满足P2的请求(1, 2, 2, 2)后，并没有立即进入死锁状态，因为此时没有进程提出新的资源请求，并因得不到资源而进入阻塞状态。只有当上述进程提出新的请求，并导致所有没执行完的多个进程因得不到资源而阻塞，系统才进入死锁状态。

#### 2.7.5死锁检测及解除

1.死锁检测算法

死锁检测算法与死锁避免算法很相似，差别仅在于死锁避免算法将进程今后所有可能的资源请求都考虑在内，而死锁检测算法仅考虑当前资源分配状态和当前资源请求。

2.死锁解除

（1）剥夺资源法；（2）撤销进程法。

## 第3章 存储管理

### 3.1存储管理概述

#### 3.1.1存储管理的功能

存储管理应具有以下功能：（1）内存分配与回收；（2）地址转换；（3）内存扩充；（4）内存保护。

#### 3.1.2存储分配的方式

1.直接方式。程序员在编写程序或编译程序对源程序编译时采用内存物理地址。

2.静态分配方式。在将程序装入内存时才确定程序在内存中的位置，程序在装入内存时必须分配它要求的全部空间，如果没有足够的空闲内存，就无法装入该程序。此外，一旦程序进入内存后，在整个运行过程中不能在内存中移动，也不能再申请新的内存。

3.动态分配方式。程序在内存中的位置也是在装入是确定的，但在其执行过程中可以申请新的内存，也可以在内存中移动。

#### 3.1.3重定位

1.静态重定位

在将目标代码加载到主存时，将所有地址修改为“原地址+目标代码所在主存起始地址”。

2.动态重定位

需要硬件地址转换机构，比如利用一个重定位寄存器，当程序的当前模块加载到主存时，将模块内存起始地址送入重定位寄存器，以后在执行访存指令是自动按“原地址+重定位寄存器”进行访问。

动态重定位的特点是允许程序分配到不连续的内存块，程序只需要装入部分代码即可运行。

直接分配方式不需要重定位，静态分配方式一般使用静态重定位，动态分配方式则需要动态重定位。

#### 3.1.4虚拟存储器

基于程序访问的局部性原理，在程序装入时，可以将程序的一部分装入内存，而将其余部分放在外存，就可以启动程序开始执行。在程序执行过程中，当所访问的信息不在内存时，由操作系统将所需要的部分调入内存，然后继续执行程序。另一方面，操作系统将内存中暂时不使用的内容换出到外存上，从而腾出空间。从效果上看，这样的计算机系统好像为用户提供了一个存储容量比实际物理内存大得多的存储器，这个存储器称为虚拟存储器。

常用的虚拟内存技术有请求分页存储管理，请求分段存储管理和请求段页式存储管理。

#### 3.1.5存储管理方法的分类

存储管理方法分为连续存放与非连续存放两类。连续存放的存储管理方法有单道连续分配法，多道固定划分法和多道可变划分法；非连续存放的存储管理方法有分页、分段和段页式存储管理。

### 3.2单道连续分配式存储管理

单道连续分配通常只能用于单任务的操作系统，它将内存分为两个连续的存储区域，一个存储区域固定分配给操作系统，另一个存储区域给用户作业使用。

### 3.3分区存储管理

分区存储管理是满足多道程序设计需要的一种最简单的存储管理方法。分区存储管理是把内存划分成若干分区，除操作系统占用一个分区之外，其余的每一个分区容纳一个用户程序。按分区数目的变化情况，可将分区存储管理进一步划分为固定分区存储管理和动态分区存储管理。

#### 3.3.1固定分区（多道固定划分法）

固定分区将内存空间划分为若干个固定大小的分区，每个分区中可以装入一道程序。分区的大小可以不等，但事先必须确定，在运行时不能改变。

为了实现固定分区分配，系统需要建立一张分区说明表，以记录内存的分配情况，表项一般包含分区号、分区的大小、分区的起始地址及状态。

#### 3.3.2动态分区（多道可变划分法）

动态分区并不事先将主存划分成一块块的分区，而是在作业进入主存时，根据作业的大小动态地建立分区，并使分区的大小正好适应作业的需要。因此系统中分区的大小是可变的，分区的数目也是可变的。

1.动态分区中的数据结构

为了实现动态分区分配，系统中必须设置相应的数据结构来记录内存的使用情况，常用的数据结构有：

（1）空闲分区表。设置一个空闲分区表来登记系统中的空闲分区，每个空闲分区对应一个表项，每个表项包含分区号、分区起始地址、分区大小及状态，如表6.5所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 分区号 | 大小 | 起始地址 | 状态 |
| 1  2  3  4  5 | 32KB  ...  520KB  ...  ... | 352KB  ...  504KB  ...  ... | 空闲  空表目  空闲  空表目  ... |

（2）空闲分区链。用单链表将空闲分区链接起来，构成空闲分区链。

2.分区分配算法

首次适应算法，循环首次适应算法，最佳适应算法，最坏适应算法

3.分区的回收

系统在空闲分区表（或空闲分区链）中检查是否有相邻的空闲分区，如有相邻空闲分区，则合并为一个大的空闲区。

4.动态分区的动态管理

（1）拼接技术

内部碎片是指分配给作业的存储空间中未被利用的部分，又称内零头；外部碎片是指系统中无法利用的小存储块，又称外零头。如固定分区分配中存在内部碎片，而动态分区分配中存在外部碎片。

解决碎片问题的办法之一是将存储器中所有已分配的分区移动到主存的一端，使本来分散的多个小空闲区连成一个大的空闲区，这种方法称为拼接，也可以称为紧凑或紧缩。

（2）可重定位动态分区

与动态分区的差别在于，增加了拼接技术，即在程序运行过程中允许它在主存空间中移动，移动时要进行地址重定位。

### 3.4覆盖与交换

所谓覆盖技术，就是把一个大的程序划分为一系列的覆盖，每个覆盖就是一个相对独立的程序单位，把程序执行时并不要求同时装入内存的覆盖组成一组，称为覆盖段，这个覆盖段分配到同一个存储区域，这个存储区域称为覆盖区，它与覆盖段一一对应。

交换技术就是把暂时不用的某个程序及数据部分（或全部）从内存移到外存中去，以便腾出必要的内存空间，或把指定的程序或数据从外存读到内存中，并将控制权转给它，让其在系统上运行的一种内存扩充技术。

二者的区别：覆盖由程序员负责，需要程序员提供程序的覆盖结构，交换由操作系统负责；覆盖主要在同一个作业或进程内进行，而交换主要是在作业或进程之间进行。

交换技术和虚拟存储器的相同之处是：都将本应处于实际内存的内容调至辅存，提高系统效率，不同之处是：交换技术并未提供大于实际内存空间的逻辑空间，该技术并不是直接面向用户的；而虚拟存储器则是提供更大的逻辑空间供用户使用，是直接面向用户的。

### 3.5分页存储管理

#### 3.5.1分页原理

1.分页原理

在分页存储管理中，用户作业的地址空间被划分成若干大小相等的区域，称为页。相应地，将主存的存储空间也分成与页大小相等的区域，称为块或物理块。

在调度作业运行时，必须将它的所有页面一次调入主存；若主存没有足够的物理块，则作业等待。这样的分页方式称为简单分页或纯分页。

分页存储管理系统中的逻辑地址结构如图3.1所示，它包含两部分，前一部分为页号，后一部分为页内偏移。

页号

页内偏移

图3.1 分页系统中的逻辑地址结构

为了便于在内存中找到进程的每个页面所对应的物理块，系统为每个进程建立一张页面影响表，简称页表，记录页面对应的物理块号。页表一般放在内存中，图3.2说明了页表的作用。

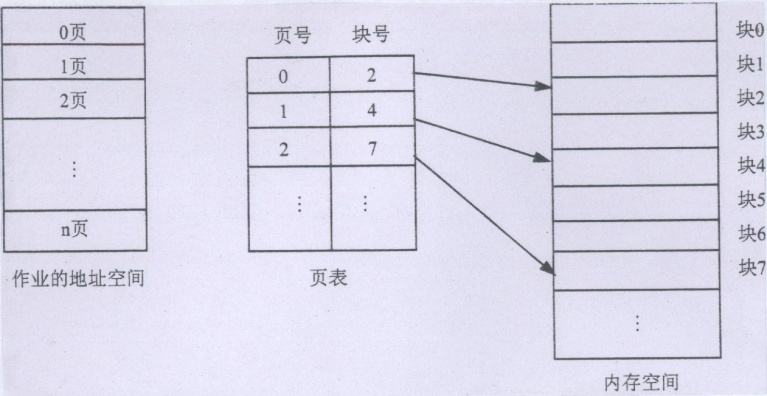


图3.2 页表的作用

2.地址变换过程

逻辑地址到物理地址的变换要借助页表来实现，页表通常存放在内存中。为了实现上的方便，系统中设置了一个页表寄存器，其中存放页表在内存的起始地址和页表的长度。进程未执行时，页表的起始地址和长度存放在进程控制块中。当进程执行时，才将页表起始地址和长度存入页表寄存器中。

当进程要访问某个逻辑地址时，分页系统的地址变换机构自动地将逻辑地址分为页号和页内偏移两部分，再以页号为索引去检索页表。在执行检索之前，先将页号与页表长度进行比较，如果页号超过了页表长度，则表示本次所访问的地址已超越进程的地址空间，系统产生地址越界中断。若未出现越界，则由页表始址和页号计算出相应页表项的位置，从中得到该页的物理块号。最后，将物理块号与逻辑地址中的页内偏移拼接在一起，就形成了该逻辑地址的物理地址。图3.3给出了分页存储管理系统中的地址变换机构。

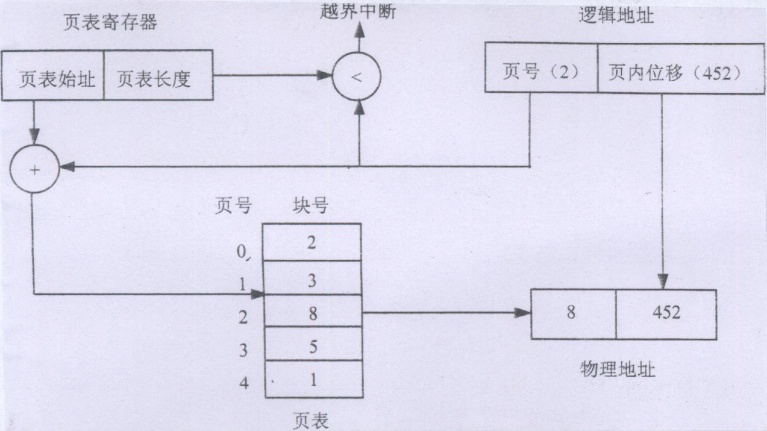


图3.3 分页系统的地址变换过程

若页表放在内存中，则存取一个数据或指令至少需要访问两次主存，第一次是访问页表，得到所访问逻辑地址的物理地址，第二次才根据得到的物理地址存取数据或指令。显然，这种情况下访存速度慢了一倍。为了提高地址变换的速度，可以在地址变换机构中增设一个具有并行查找能力的高速缓冲存储区（一般用联想存储器实现），称为快表，将最近使用过的那些页表项存放在快表中，页表的其余部分仍存放在内存中。

### 3.6请求分页存储管理

#### 3.6.1请求分页原理

在简单分页的基础上，增加了请求调页、页面置换功能，形成了一种虚拟存储器。

扩充后的页表表项如下图所示。

外存地址

修改位

状态位

访问频度

物理块号

页号

图3.4 扩充后的页表项

#### 3.6.2页面调度策略

1.页面调入策略

（1）请求调页：当进程所访问页面不在主存时，才调入该页。这种调入策略实现简单，但容易产生较多的缺页中断，使得外存I/O次数较多，时间开销较大，容易产生抖动现象。

（2）预调页：一次调入缺页及与缺页相邻的几个页面。

2.页面分配和置换策略

（1）固定分配局部置换。为每一个进程分配固定数量的物理块，在进程的整个运行期间进程拥有的物理块数不再改变。若在进程运行中出现缺页，则只能从本进程的几个页面中选择一页换出，然后再调入缺页，以保证分配给该进程的内存空间量不变。

（2）可变分配局部置换。采用这种策略时，为每一个进程分配一定数量的物理块，当某个进程发生缺页时，只允许从该进程的页面中选出一页换出，如果该进程频繁发生缺页中断，则系统再为该进程分配若干物理块，直到进程的缺页率降低到适度程度为止；反之，若一个进程在运行过程中缺页率特别低，则系统可适当减少分配给该进程的物理块数。

（3）可变分配全局置换。当某个进程发生缺页时，由系统从空闲物理块队列中取出一个分配给该进程。当系统的物理块用完时，操作系统才选出某个进程中的一页调出。

#### 3.6.3页面置换算法

1.最佳算法（OPT）

最佳算法是从主存中年选择永远不再需要的页面或在最长的时间以后才需要访问的页面予以淘汰。

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 7 | 7 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  |  | 1 | 1 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 1 | 1 | 1 | 1 |

× × × × × × × ×

图3.5 OPT算法下驻留集的变化（从上而下，指针动）

2.先进先出算法（FIFO）

在主存中选择驻留时间最长的页面予以淘汰。

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 7 | 7 | 2 | 2 | 2 | 2 | 4 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 |
|  |  | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 |

× × × × × × × × × × × ×

图3.6 FIFO算法下驻留集的变化（循环队列，指针动）

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 0 | 1 | 2 | 2 | 3 | 0 | 4 | 2 | 3 | 0 | 0 | 0 | 1 | 2 | 2 | 2 |
|  | 7 | 0 | 1 | 1 | 2 | 3 | 0 | 4 | 2 | 3 | 3 | 3 | 0 | 1 | 1 | 1 |
|  |  | 7 | 0 | 0 | 1 | 2 | 3 | 0 | 4 | 2 | 2 | 2 | 3 | 0 | 0 | 0 |

× × × × × × × × × × × ×

图3.7 FIFO算法下驻留集的变化（从上而下，挤牙膏）

FIFO算法存在Belady现象，即工作集窗口加大，缺页次数反而增多。

3.最近最少使用算法（LRU）

选择最近一段时间内最长时间没有被访问过的页面予以淘汰。

7 0 1 2 0 3 0 4 2 3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0/7 | 1/7 | 2/7 | 0/2 | 1/2 | 2/2 | 3/2 | 0/4 | 1/4 | 2/4 |
|  | 0/0 | 1/0 | 2/0 | 0/0 | 1/0 | 0/0 | 1/0 | 2/0 | 0/3 |
|  |  | 0/1 | 1/1 | 2/1 | 0/3 | 1/3 | 2/3 | 0/2 | 1/2 |

× × × × × × × ×

0 3 2 1 2 0 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0/0 | 1/0 | 2/0 | 0/1 | 1/1 | 2/1 | 0/1 |
| 1/3 | 0/3 | 1/3 | 2/3 | 2/3 | 0/0 | 1/0 |
| 2/2 | 3/2 | 0/2 | 1/2 | 0/2 | 1/2 | 2/2 |

× × ×

图3.8 LRU算法下驻留集的变化（从上而下，计数器）

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 |
|  | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 |
|  |  | 7 | 0 | 1 | 2 | 2 | 3 | 0 | 4 | 2 | 2 | 0 | 3 | 3 | 1 | 2 |

× × × × × × × × × × ×

图3.9 LRU算法下驻留集的变化（从上而下，挤牙膏，被访问则置顶）

4.最近未使用算法（Clock算法）

2 3 2 1 5 2 4 5 3 2 5 2

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2\* | 2\* | 2\* | 2\* | 5\* | 5\* | 5\* | 5\* | 3\* | 3\* | 3\* | 3\* |
|  | 3\* | 3\* | 3\* | 3 | 2\* | 2\* | 2\* | 2 | 2\* | 2\* | 2\* |
|  |  |  | 1\* | 1 | 1 | 4\* | 4\* | 4 | 4 | 5\* | 5\* |

× × × × × × × ×

图3.10 NRU算法下驻留集的变化（从上而下，标志位）

例3.1 在页式虚存管理系统中，假定驻留集为m个页，初始所有页均为空，在长为p的引用串中具有n个不同页号（n>m），对于FIFO、LRU两种页面置换算法，试给出页故障数的上限和下限，说明理由，并举例说明。

【解】对于FIFO、LRU置换算法，页故障（即缺页中断）上限均为p，下限均为n。

由于驻留集初始为空，所以n个不同的页面调入内存都要发生缺页中断，因此下限是n。

最坏的情况是，每当访问一个页面时，该页都不在内存，所以上限是p。举例如下。

m=3, p=12, n=4，有引用串1、1、1、2、2、3、3、3、4、4、4、4，则两种置换算法产生的缺页中断数均为4。

### 3.7分段与段页式存储管理

#### 3.7.1分段存储管理

1.分段原理

在分段存储管理系统中，作业的地址空间由若干个逻辑分段组成，每个分段是一组逻辑意义相对完整的信息集合，每个分段都有自己的名字，每个分段都从0开始编址，并采用一段连续的地址空间。因此，整个作业的地址空间是二维的。

分段存储管理系统的逻辑地址结构由段号和段内偏移组成，其结构如图3.11所示。

段内偏移

段号

图3.11 分段系统中的逻辑地址结构

2.地址变换过程

系统为每个进程建立一张段表，其中每个表项描述一个分段的信息，表项中包含段号、段长和该段的内存起始地址。系统中设置了一个段表寄存器，用于存放段表起始地址和段表长度。在进行地址变换时，系统将逻辑地址中的段号与段表长度进行比较，若段号超过了段表长度，则表示段号越界，于是产生越界中断信号；若未越界，则根据段表起始地址和段号计算出该段对应段表项的位置，从中读出该段在内存的起始地址，然后，在检查段内偏移是否超过该段的段长度，若超过则同样发出越界中断信号；若未越界，则将该段的起始地址与段内偏移相加，从而得到了要访问的物理地址。为了提高内存的访问速度，也可以使用块表。

3.请求分段存储管理

段表项：

状态位

访问频度

内存始址

段长

段号

外存地址

修改位

图3.12 扩充后的段表项

4.分段与分页的区别

（1）页是信息的物理单位，分页时为了实现离散分诶方式，仅仅是出于系统管理的需要，而不是用户的需要。段是信息的逻辑单位，分段是为了更好地满足用户的需要。

（2）页的大小固定且由系统决定，把逻辑地址划分为页号和页内偏移两部分，是由机器硬件实现的。段的长度不固定，且由用户所编写的程序决定，通常由编译系统在对源程序进行编译时根据信息的性质来划分。

（3）分页系统中作业的地址空间是一维的，程序员只需利用一个值来表示地址；分段系统中作业的地址空间是二维的，程序员在标识一个地址时，既要给出段名，又要给出段内偏移。

#### 3.7.2段页式存储管理

页内偏移

段内页号

段号

逻辑地址：

每个进程一个段表，每个段一个页表。

若段表和页表全放在主存中，则访问主存中的一条指令或数据，至少需要访问主存三次，应考虑使用联想存储器。

## 第4章 文件管理

### 4.1文件系统的概念

文件是具有文件名的一组相关信息的集合，设备也可以看作一种特殊的文件。文件系统是操作系统中与文件管理有关的软件和数据的集合。

文件系统的层次结构：TODO

### 4.2文件结构和存储设备

#### 4.2.1文件的逻辑结构

可分为无结构的流式文件和有结构的记录式文件。

#### 4.2.1文件的物理结构

文件的物理结构是指文件在外存上的存储组织形式。常见的文件物理结构有：顺序结构、链接结构和索引结构。

#### 4.2.3文件的访问方式

顺序访问，随机访问，按键访问

#### 4.2.4存储设备

磁带，顺序访问设备。磁带只能采用顺序结构。

磁盘，随机访问设备。三种物理结构都可以采用。

访问一次磁盘所需要的时间可以分解为寻道时间、旋转延迟时间和传输时间。

磁盘调度算法：

（1）先来先服务算法（FCFS）

（2）最短寻道时间优先（SSTF）算法

（3）扫描算法（SCAN），也称为电梯算法

（4）循环扫描算法（CSCAN）

### 4.3存储空间的分配和管理

#### 4.3.1存储空间的分配

连续分配，链接分配，索引分配

#### 4.3.2空闲存储空间的管理

1.空闲文件目录

存储设备上的一个空闲块也可以看作是一个文件，称为空闲文件或空白文件。空闲文件目录法为这些空闲文件单独建立一个目录，每个空闲文件在这个目录中占一个表目。

2.空闲块链

将所有空闲块链接起来。

3.位示图

这种方法是为文件存储设备建立一张位示图（也称位图），以反映整个存储空间的分配情况。在位示图中，每一个二进制位都对应一个物理块，当某位为“1”时，表示对应的物理块分配，若某位为“0”，则表示对应的物理块空闲。

### 4.4文件目录管理

#### 4.4.1文件目录

文件由文件控制块和文件体两部分组成。文件体即文件本身，文件控制块（FCB）用于存放文件的各种属性信息。文件控制块的集合称为文件目录，文件目录是一张表格，每个表项可以是文件控制块，也可以是指向文件控制块的指针。

#### 4.4.2文件目录结构

1.单级目录结构

文件系统只有一张目录表

2.二级目录结构

文件系统中有一个主文件目录MFD(Master File Directory)，每个用户占用一个表项，每一项指向一个用户文件目录UFD(User File Directory)，用户文件目录下才是该用户的文件。

3.树形目录结构

将二级目录加以推广，就形成了多级目录结构，即树形目录结构。

当前目录：考虑到进程在一段时间内所访问的文件通常具有局部性，可在这一段时间内指定某个目录作为当前目录（或称为工作目录）。进程对个文件的访问都是相对于当前目录进行的。这样可以节省很多重复的查找。

### 4.5文件共享与安全

#### 4.5.1文件共享

文件共享有三种典型的方法：

1.硬链接（hard linking）

允许多个目录项指向同一个文件，形成了无环图结构。

2.间接链接（indirect linking）

此时目录项不再是一个FCB，而是一个FCB指针。将原来的FCB中的文件名和文件属性信息分开，文件属性单独构成一个数据结构，称为索引节点（又称i节点），而文件目录中的每个目录项仅由文件名和该文件的索引节点指针构成。UNIX采用了这种方式。

3.符号链接

例如，目录B共享A中的文件C时，在目录B中创建一个文件D，D的内容是C的路径（相对路径或绝对路径）。Windows采用了这种方式，即快捷方式。

#### 4.5.2文件保护

1.访问控制

访问控制矩阵，访问控制表（相当于矩阵的一行），用户权限表（相当于矩阵的一列）

2.口令

3.密码

#### 4.5.3文件转储和恢复

文件转储：全量转储，增量转储

文件恢复：首先从最近一次全量转储文件恢复全部文件，然后由旧到新从增量转储文件恢复文件（可能一个文件被转储过若干次，但只恢复最近一次转储的副本，其他则被覆盖掉）。

### 4.6文件操作

新建，删除，打开，关闭，读，写

## 第5章 设备管理

### 5.1设备管理概述

设备管理也称为输入/输出管理。

#### 5.1.1设备分类

1.人机交互设备：又称为慢速I/O设备。这类设备主要与用户打交道，数据交换速度相对较慢，通常是以字节为单位进行数据交换。

2.存储设备：这类设备主要用于存储数据，数据交换速度较快，通常以多字节组成的块为单位进行数据交换。主要有磁带，磁盘，光盘等。

3.网络通信设备：这类设备数据交换速度通常高于人机交互设备，低于存储设备，主要有各种网络接口、调制解调器等。

#### 5.1.2设备管理的主要功能

设备分配，设备控制，缓冲管理，设备独立性

#### 5.1.3设备控制器与I/O通道

1.设备一般有机械和电子两部分组成，设备的电子部分通常称为设备控制器。设备控制器中通常设有3类寄存器，控制寄存器，状态寄存器，和数据寄存器，分别用于存放控制命令，状态信息和数据。CPU通过读写这3类寄存器来控制设备。

2.I/O通道

I/O通道是专门负责输入输出工作的处理机。根据信息交换方式的不同，可以将通道分为三类：字节多路通道，数据选择通道，数据多路通道。

#### 5.1.4 I/O系统结构

1.微机型I/O系统结构

多采用总线结构，CPU，存储器和外部设备都连接到总线上。

2.主机型I/O系统结构

外部设备不直接连接到总线上你，而是增加了一级I/O通道，通过I/O通道连接到总线上。

### 5.2 I/O控制方式

程序查询方式，中断方式，DMA方式，通道方式

详情见计算机组成原理笔记8.4节。

### 5.3中断技术

#### 5.3.1中断的基本概念

中断是指计算机系统内发生了某一急需处理的事件，使得CPU暂时中止当前正在执行的程序而去执行相应的事件处理程序，待处理完毕后又返回到原来被中断处继续执行。引起中断发生的事件称为中断源。中断源向CPU发出的请求中断处理信号称为中断请求，CPU收到中断请求后转向相应事件处理程序的过程称为中断响应。

发生中断时，刚执行完的那条指令所在的单元号称为断点，断点的逻辑后续指令的单元号称为恢复点。而现场是指中断的那一时刻能确保程序继续运行的有关信息。

在有些情况下，尽管产生了中断源和发出了中断请求，但CPU内部的处理机状态字PSW的中断允许位已被清除，从而不允许CPU响应中断，这种情况称为禁止中断（也称为关中断）。CPU禁止中断后只有等到PSW的中断允许位重新设置后才能接收中断。设置PSW的中断允许位称为开中断。开中断和关中断是为了保证某些程序执行的原子性。

中断屏蔽是指系统用软件的方式有选择地封锁部分中断而允许其余部分中断仍能得到响应。不过，有些中断请求是不能屏蔽甚至不能禁止的，也就是说，这些中断具有最高优先级，不管CPU是否关中断，只要这些中断请求一提出，CPU必须立即响应。例如电源掉电事件引起的中断就是不可禁止和屏蔽的中断。

#### 5.3.2中断的分类与优先级

内中断的优先级往往高于外中断的优先级，时钟中断的优先级往往高于其它外设中断。

根据中断信号的来源可以将中断分为两类：

（1）外中断。指来自处理机和内存外部的中断。

（2）内中断。指在处理机和内存内部产生的中断，内中断一般称为陷入或异常，它包括程序运算引起的各种错误，如地址非法、校验错，存取访问控制错、算术操作溢出、数据格式非法、除数为零、非法指令、用户程序执行特权指令等。

#### 5.3.3中断处理过程

中断处理过程如下：1.保护现场；2.执行中断服务程序；3.恢复现场，继续运行。

更多细节请参见计组笔记8.4.2节。

### 5.4缓冲技术

#### 5.4.1缓冲的引入

设置内存I/O缓冲区的主要原因如下：

（1）缓和CPU与I/O设备速度不匹配的矛盾。一般情况下，程序的运行过程是时而进行计算，时而进行输入或输出的。以输出为例，如果没有输出缓冲区，则程序在输出时，必然由于打印的速度跟不上而是CPU停下来等待；然而在计算阶段，打印机又无事可做。如果设置一个缓冲区，程序可以将带输出的数据先输出到缓冲区中，然后继续执行；而打印机则可以从缓冲区取出数据慢慢打印。

（2）减少中断CPU的次数。例如，假如某设备每接收到一个字节就中断CPU一次，若设置一个具有4字节的缓冲寄存器，则可使CPU被中断的次数降低为前者的1/4。

（3）提高CPU和I/O设备之间的并行性。由于在CPU和设备之间引入了缓冲区，CPU可以从缓冲区读取或向缓冲区写入信息，相应地设备也可以向缓冲区写入或从缓冲区读取信息。在CPU工作的同时，设备也能进行输入输出操作，这样CPU和I/O设备及可以并行工作。

根据系统设置的缓冲区个数，可以将缓冲技术分为单缓冲、双缓冲、循环缓冲和缓冲池。

#### 5.4.2单缓冲

操作系统只设置了一个缓冲区，因此设备与处理机对缓冲区的操作时穿行的。

#### 5.4.3双缓冲

一个输入缓冲，一个输出缓冲。

#### 5.4.4循环缓冲

循环缓冲包含多个大小相等的缓冲区，每个缓冲区有一个链接指针指向下一个缓冲区，最后一个缓冲区指向第一个缓冲区，这样构成一个环形。循环缓冲用于输入输出时，还需要设置两个指针in和out。

#### 5.4.5缓冲池

缓冲池有多个缓冲区组成，缓冲池中的缓冲区按其使用状况可分成三个队列：空缓冲队列、装满输入数据的缓冲队列（输入队列）和装满输出数据的缓冲队列（输出队列）。除上述三个队列之外，还应具有四种工作缓冲区：用于收容输入数据的工作缓冲区、用于提取输入数据的工作缓冲区、用于收容输出数据的工作缓冲区和用于提取输出数据的工作缓冲区。

当输入进程需要输入数据时，便从空缓冲区队列的队首摘下一个空缓冲区，把它作为收容输入工作缓冲区，然后把数据输入其中，装满后再将它挂到输入队列队尾。当计算进程需要输入数据时，便从输入队列取得一个缓冲区作为提取输入工作缓冲区，进程从中提取数据，数据用完后再将它挂到空缓冲队列对为。当计算进程需要暑促数据时，便从空缓冲队列的队首摘下一个空缓冲区，把它作为收容输出工作缓冲区，当其中装满输出数据后，再将它挂到输出队列队尾。当要输出时，由输出进程从输出队列中取得一个装满输出数据的缓冲区，作为提取输出工作缓冲区，当数据提取完后，再将它挂到空缓冲区队列的队尾。

### 5.5设备分配

#### 5.5.1设备分配策略

1.设备的使用性质：独占式，分时共享式，Spooling方式。

2.设备分配算法：先来先服务，优先级高者优先。

#### 5.5.2 Spooling系统

Spooling又称为假脱机输入输出操作，是操作系统的采用的一项将独占设备改造成共享设备的技术。Spooling系统是对脱机输入输出工作的模拟，在该系统中，用一道程序来模拟脱机输入时外围设备控制器的功能，把低俗输入设备上的数据传送到高速磁盘上；再用另一道程序来模拟脱机输出时外围设备控制器的功能，把数据从磁盘上传送到低俗输出设备上。这样，便可以在主机的直接控制下，实现脱机输入输出功能。

Spoling系统的组成如图5.1所示，主要包括以下三部分：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。

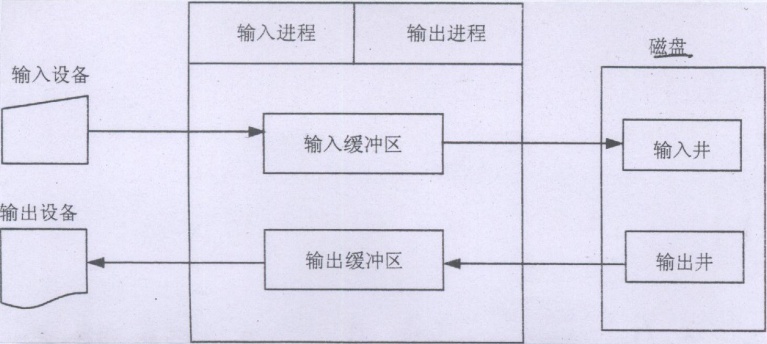


图5.1 Spooling系统的组成

### 5.6 I/O系统层次结构

#### 5.6.1设备驱动程序

设备驱动程序和具体的设备密切相关

#### 5.6.2中断处理程序

无论是那种I/O设备，其中断处理程序的处理过程大致相同，步骤如下：

（1）唤醒被阻塞的驱动程序进程。

（2）保护被中断进程的CPU现场。

（3）分析中断原因，转入相应的设备中断处理程序。由CPU确定引起本次中断的设备，然后转到相应的设备中断处理程序执行。

（4）进行中断处理。设备中断处理程序从设备控制器读出设备状态，以判断本次设备中断是正常结束还是异常介绍。若为正常结束，则设备驱动程序便可做结束处理；若为异常结束，则根据发生异常的原因作相应处理。

（5）恢复被中断进程的现场。

#### 5.6.3与设备无关的I/O

1.设备命名；2.设备保护；3.提供与设备无关的逻辑块；4.缓冲；5.存储类设备的块分配；6.独占式设备的分配和释放；7.错误处理。

#### 5.6.4用户层I/O

I/O库函数，Spooling系统等。

参考书籍：

[1] 操作系统考研指导，曾平，清华大学出版社，2003年1月第1版

如果笔记中有错误或遗漏了重要的考点，欢迎反馈。

电子邮件：[soulmachine@gmail.com](mailto:soulmachine@gmail.com)

作者博客：[www.yanjiuyanjiu.com](http://www.yanjiuyanjiu.com) [研究研究]

本笔记遵循创作共享协议2.0，禁止一切商业用途。