考研计算机基础班讲义

操作系统

主讲：孙卫真

欢迎使用新东方在线电子教材

目 录

第一章 操作系统概述

考纲要求

（一）操作系统的概念、特征、功能和提供的服务
（二）操作系统的发展与分类
（三）操作系统的运行环境
（四）操作系统体系结构

本章的内容在统考中一般出 1 题选择题，分值 2 分。

1.1 操作系统的概念、特征、功能和提供的服务

1.1.1 操作系统的概念

操作系统是计算机系统中最重要、最基本的系统软件，操作系统位于硬件和用户程序之间。一方面，它能向用户提供使用计算机的接口；另一方面，它能管理计算机软硬件资源，提高其利用率；再者，利用虚拟技术，扩展了计算机的功能和使用范围。

因此，操作系统的定义为：操作系统是控制和管理计算机软、硬件资源，以尽可能合理、高效的方法为不同用户及其应用程序提供服务的一种系统程序。

1.1.2 操作系统的特征

操作系统具有并发、共享、虚拟和不确定性四大特征。其中，最重要的是并发特征，其他三个特征都是以并发为前提的。

1.1.3 操作系统的功能

操作系统主要有进程管理、存储管理、文件管理、输入/输出管理和作业管理五大功能。

1.1.4 操作系统所能提供的服务

操作系统为用户程序和系统程序提供了一系列的服务，这些服务可使使用计算机的人更快捷、高效和简单地完成自己的工作。

1．命令输入

提供人机对话平台。

2．系统调用服务

提供编程时的系统服务。

1.2 操作系统的发展与分类

1.2.1 操作系统的发展

操作系统的发展目前呈现出多样化的局面，大型计算机、巨型计算机需要满足其集群计算，高性能计算的需求；军用计算机、工业控制计算机希望操作系统能实时响应；嵌入式计算机要求精简、功能专一；便携式设备要求省电，电池持续耐力强等等。因此，操作系统将会随着用户对系统不断的新要求，在硬件的支持下，得到更加快速、强大地发展。

1.2.2 操作系统的分类

1．单用户操作系统

2．批处理操作系统

批处理系统又分为以下两类。

（1）单道批处理系统

（2）多道批处理系统

3．分时操作系统

4．实时系统

5．网络操作系统

6．分布式操作系统

7．并行操作系统

1.3 操作系统的运行环境

1.3.1 内核态与用户态

多数系统将处理器工作状态划分为内核态和用户态。前者一般指操作系统管理程序运行的状态，具有较高的特权级别，又称为特权态、系统态或管态；后者一般指用户程序运行时的状态，具有较低的特权级别，又称为普通态、目态。

1.3.2 中断、异常

所谓中断（interrupt）是指处理机对系统中或系统外发生的异步事件的响应。异常（有时也称为陷阱 trap）是指由系统发起的一次确定的服务过程。

中断与异常的区别与联系：就比较通用的观点来看，中断是强迫性的，异常是自愿性的；中断一般外来的，异常是程序发出的，中断服务于所有程序，异常一般为发出异常的程序服务。

1.3.3 系统调用

系统调用的处理过程是这样的，当系统调用发生时，处理器通过一种特殊的机制，通常是中断或者异常处理，把控制流程转移到监控程序内的一些特定的位置。同时，处理器模式转变成特权模式。其次，由监控程序执行被请求的功能代码。这个功能代码代表着对一段标准程序段的执行，用以完成所请求的功能。第三，处理结束之后，监控程序恢复系统调用之前的现场；把运行模式从特权模式恢复成为用户方式；最后将控制权转移回原来的用户程序。

系统调用与一般程序调用的不同：

（1）运行在不同的系统状态。调用的程序是运行在用户态，被调用的程序运行在系统态。

（2）进入的方式不同。过程调用语句直接跳转到被调用过程；而系统调用则必须通过运行系统调用命令。

（3）返回方式不同。过程调用直接返回；系统调用则不直接返回，有重新调度过程。

（4）代码层次不同。过程调用是用户级程序，而系统调用是系统级程序。

（5）系统调用一般不能嵌套或递归。

1.4 操作系统体系结构

常见的操作系统体系结构有整体式结构、层次式结构和微内核（客户/服务器）结构等。本节我们对这些常见的操作系统体系结构做简要的介绍。

1．整体式结构

首先确定操作系统的总体功能，然后将总功能分解为若干个子功能，实现每个子功能的程序称为模块。它的主要优点是：结构紧密，接口简单直接，系统效率较高。

2．层次式结构

所谓层次式结构就是把操作系统的所有功能模块，按功能流图的调用次序，分别将这些模块排列成若干层，各层之间的模块只能是单向依赖或单向调用关系。这样不但操作系

统的结构清晰，而且不构成循环。

3．微内核（客户/服务器）结构

这种模式内核提供所有操作系统基本都具有的那些操作，如线程调度、虚拟存储、消息传递、设备驱动以及内核的原语操作集和中断处理等。这些部分通常采用层次结构并构成了基本操作系统。

第二章 进程管理

考纲要求

（一）进程与线程
（二）处理机调度
（三）同步与互斥
（四）死锁
本章是统考的重点，出题分数在 10-16 分，一般会出 1 题综合题，2-5 题选择题。

2.1 进程与线程（理解概念、掌握原理、综合应用）

2.1.1 进程的概念

1．进程概念的定义

进程是并发程序的动态运行，是多道程序系统中程序的动态运行过程。

进程是一个活动的实体，除了指令代码，进程通常还包括进程堆段、栈段（包含临时数据，如方法参数、返回地址和局部变量）和数据段（包含常量或全局变量等）。

进程是程序在数据集合上运行的过程，它是系统进行资源分配和调度的一个独立单位。

2．进程的特征

动态性：动态性是进程最基本的特征。即进程由创建而产生，由调度而运行，因得不到资源而暂停运行，以及由撤销而消亡。

并发性：多进程同时存在于内存中，在一段时间内，同时运行。

独立性；进程是自我封闭的，有共享代码段的进程互相之间也是独立的。

异步性：进程按各自独立的、不可预知的速度向前推进，导致程序具有不可再现性。因此，在操作系统中，必须采取某种措施来保证各程序之间能协调运行。

结构性：由程序代码（段）、数据（段）和进程控制块组成。

2.1.2 进程的状态与转换

1．进程的基本状态包括有以下几种（三状态模型）：

运行状态（Running）：进程占用处理机正在运行其程序。单处理机系统中只能有一个进程处于运行状态，多处理机系统中可能有多个进程处于运行状态。

阻塞状态（Blocked）：也叫等待或睡眠状态，是进程由于等待某种事件的发生而处于暂停运行的状态。如进程因等待输入/输出的完成、等待数据到达、等待缓冲空间等。

就绪状态（Ready）：进程已分配到除处理机以外的所有必要资源，具备了运行的条件，可能会有多个进程处于就绪状态，排成就绪队列。图 2-1 说明了三状态进程模型及其转换。

进程还有五状态和七状态模型，需要作一般了解。而三状态的模型是最基本的模型。

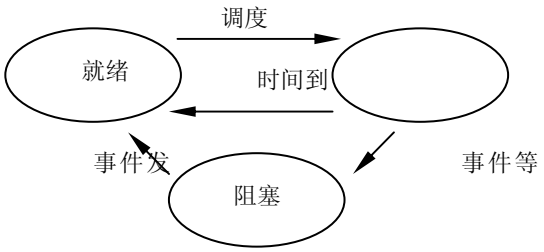

图 2-1 三状态进程模型

4．进程状态的转换

(1)就绪状态到运行状态：调度程序为就绪状态的进程分配处理机后，进入运行状态。

(2)运行状态到阻塞状态：正在运行的进程因需要等待某事件而无法运行，让出处理机。

(3)阻塞状态到就绪状态：进程所等待的事件发生了，进程就从阻塞状态进入就绪状态。

(4)运行状态到就绪状态：正在运行的进程因时间片用完而被暂停运行；或者在可抢先式调度方式中，一个优先级高的进程到来后，正在运行的优先级低的进程被强制撤下处理机，转换为就绪状态。

2.1.3 进程控制

1．进程原语

所谓原语即原子操作，要么全做，要么不做，一般是通过屏蔽中断来完成的。

（1）进程创建。

（2）进程撤销。

（3）进程阻塞。

（4）进程唤醒。

（5）进程挂起。

（6）进程激活。

2.1.4 进程组织

1．进程实体

程序：描述进程所要完成的功能，特指二进制的指令代码。

数据集合：程序运行所需要的数据结构。包括常数，变量，堆，数据栈等。

进程控制块：进程控制块包含了进程的描述信息、控制信息和资源信息。

2．进程控制块（PCB）

PCB 是保存进程状态和进程控制的标识，也是进程存在的唯一标识（也称进程表 PT）。创建进程则产生 PCB，撤销进程则系统就要回收 PCB。PCB 表项如图 2-2。

进程描述信息	进程控制和管理信息	资源分配清单	处理机相关信息
进程标识符（PID） 用户标识符	进程当前状态 进程优先级 代码运行入口地址 程序的外存地址 进入内存时间 处理机占用时间 信号量使用	代码段指针 数据段指针 堆栈段指针 文件描述符 键盘 鼠标	通用寄存器值 地址寄存器值 控制寄存器值 标志寄存器值 状态字

图 2-2 PCB 表项内容

系统中 PCB 采用适当的方式组织起来，对其进行管理。图 2-3 所示。

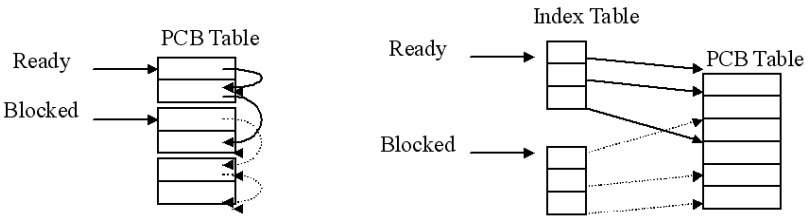

图 2-3 PCB 表组织方式

2.1.5 进程通信

进程间的信息交换工作称为进程间的通信。P、V 操作称为低级通信。高级通信是指进程之间以较高的效率传送大量数据的通信方式。高级通信方式可分为三大类：共享内存、消息传递以及管道机制。

2.1.6 线程概念与多线程模型

1．线程的发明

线程是进程中的一个实体，它是操作系统进行独立调度和分派的基本单位，但不是资源分配的基本单位。线程自己不拥有系统资源，只拥有在运行中必不可少的资源，它同样有就绪、阻塞和运行三种基本状态。并且它与同属一个进程的其他线程共享本进程所拥有的全部资源（内存、文件以及设备）。

2．线程实现方式

内核线程：由操作系统根据内部需求进行创建和撤销，内核线程依赖于操作系统内核的运行，因此操作系统知道内核线程的存在。也可以被用户所调用。

用户线程：与操作系统内核无关，用户程序利用操作系统提供的线程库来编写形成，它的创建、同步、调度和管理等诸多工作均由用户来实现。调度由用户编写的应用程序内部进行。但是由于操作系统不知道用户线程的存在，所以用户线程一旦因各种原因而阻塞，则其所在的整个进程也会阻塞。操作系统仅把处理机时间配额分配给进程，所以用户多线程时每个线程共享一个进程的时间配额，运行会变慢。

2.2 处理机调度

2.2.1 调度的基本概念

1．作业调度

作业调度又称宏观调度或高级调度。对处于后备状态的作业进行选择，并建立相应的进程。一般在批处理系统中，大多配有作业调度，而在其它系统中，通常不需配置作业调度。作业调度的运行频率较低，通常为几分钟一次。

2．进程调度

进程调度是指决定就绪队列中哪个进程将获得处理机，并实际将处理机分配给该进程的操作。

3．交换调度

交换调度又称中级调度。其主要任务是按照给定的原则和策略，将处于外存对换区中，且具备运行条件的就绪进程调入内存，或将处于内存就绪状态或内存阻塞状态的进程交换到外存对换区。

2.2.2 调度的时机、切换和过程

1．引起进程调度的事件

正在运行的进程运行完毕或发生某事件而不能再继续运行；

运行中的进程因提出输入/输出请求而暂停运行；

在进程通信或同步过程中运行了某种原语操作，如 P 操作等；

在可抢先式调度中，有一个比当前进程优先级更高的进程进入就绪队列；

在时间片轮转法中，时间片用完。

2．调度队列

在单处理机系统中，只有一个进程处于运行状态。

3．分派程序（dispatcher）

进程调度算法只是决定哪一个进程将获得处理机，是策略的制定者，而将处理机分配给该进程的具体操作是由分派程序完成的。分配程序是机制，是实际操作者，因此其运行效率较高。这里充分体现了策略与机制分离的设计思想。

2.2.3 调度的基本准则

调度的基本准则包括：
处理机利用率：尽可能让昂贵的处理机处于繁忙中。
吞吐量：单位时间内所完成进程的数量尽量多。
周转时间：从作业提交到作业完成所花费的时间。要让周转时间尽可能地小。
后备时间：是指作业抵达系统后在外存等待进入内存的时间，越小越好。
等待时间：是指在就绪队列中等待调度进入处理机的时间。
响应时间：是指从提交请求到产生第一响应输出的时间。

2.2.4 调度方式

1．不可抢先方式
2．可抢先方式
3．进程调度算法比较
（1）先来先服务（FCFS）。
（2）短作业或短进程优先（SJF&SPF）。
（3）高响应比优先调度（HRRN）算法。
响应比 Rp =（等待时间+预计运行时间）/ 预计运行时间 = 周转时间 / 预计运行时间
（4）高优先级优先调度算法。
分静态优先级和动态优先级。
（5）时间片轮转调度算法（RR）。
（6）多级反馈队列调度算法。

2.3 同步与互斥

2.3.1 进程同步与互斥的基本概念

1．基本概念
在多道程序系统中，由于进程，各进程之间有两种形式的制约关系：
（1）间接相互制约：源于资源共享。
（2）直接相互制约：源于进程合作。
进程同步：主要源于进程合作，为进程之间的直接制约关系。
进程互斥：主要源于资源共享，是进程之间的间接制约关系。
临界资源：一次只允许一个进程使用的资源称为临界资源，如打印机、公共变量等。
临界区：在每个进程中，访问临界资源的那段程序称为临界区。
2．同步机制应遵循的准则
同步机制应遵循下述四条准则：

（1）空闲则进：当临界区空闲时，进程可以立即进入，以便有效地利用临界资源。
（2）遇忙等待：当已有进程进入其临界区时，其它进程必须等待，以保证互斥。
（3）有限等待：对要求进入的进程，应在有限的时间内使之进入，以免陷入“死等”。
（4）让权等待：对于等待的进程，它必须立即释放处理机，以避免进程忙等。

2.3.2 实现临界区互斥的基本方法

1．用软件实现的同步互斥机制
（1）算法一：单标志法
（2）算法二：双标志法先检查
（3）算法三：双标志法后检查
（4）算法四：Peterson’s Algorithm
2．进程互斥的硬件方法
（1）检测和设置（TS）指令
（2）swap 指令（或 exchange 指令）该指令的作用是交换两个字（字节）的内容

2.3.3 信号量

对信号量 S 进行 P 操作，记为 P（S），处理过程如下

```
--S.Q;            //表示申请一个资源
if (S.Q < 0)      //表示没有空闲资源
{
   调用进程进入等待队列 S.Q;
   阻塞调用进程;
}
```

对信号量 S 进行 V 操作，记为 V（S），处理过程如下

```
++S.Q;            //表示释放一个资源
if (S.Q <= 0)     //表示有进程处于阻塞状态
{
   从等待队列 S.Q 中取出一个进程 P;
   进程 P 进入就绪队列;
}
```

2.3.4 管程

一个管程定义了一个数据结构和能为并发进程所运行的一组操作，这组操作能同步进程和改变管程中的数据。

管程由三部分组成：
局部于管程的共享数据说明；
对该数据结构进行操作的一组过程；
对局部于管程的数据设置初始值的语句。

2.3.5 经典的同步与互斥问题

1. 生产者-消费者问题

用 C 语言和信号量机制描述生产者-消费者问题的程序如下：有界缓冲区的大小为 100。

```
#define N 100                          //有界缓冲区大小
typedef int semaphore;                 //定义信号量
semaphore mutex = 1;                   //临界区互斥信号量
semaphore empty = N;                   //空闲缓冲区
semaphore full = 0;                    //缓冲区初始化为空
void producer(void)
{   int item;                          //局部变量
    while(1){
        item = produce_item();         //生产数据
        P（empty);                     //获取空数据槽
        P（mutex);                     //获得进入临界区的信号量
        insert_item(item);             //将数据放入缓冲池
        V（mutex);                     //释放互斥量
        V（full);                      //数据量加一
    }
}
void consumer(void)
{   int item;                          //局部变量
    while(1){
        P（full);                      //获取数据槽
        P（mutex);                     //获得进入临界区的信号量
        item = remove_item();          //将数据从缓冲池读出
        V（mutex);                     //释放互斥量
        V（empty);                     //数据量减一，即空槽加一
        consume_item(item);      //消费数据
    }
}
```

2. 读者-写者问题

设置互斥信号量 wmutex 表示写者间、读者和写者间互斥，用 readcount 变量来记录读者数，由于 readcount 是读者间共享变量，属于临界资源，它也需互斥，为此，又增设互斥信号量 rmutex。程序如下：

```
typedef int semaphore;                 //定义信号量
semaphore rmutex = 1;                  //读者计数器的互斥量
semaphore wmutex = 1;                  //写-写，读-写互斥量
```

```
int readcount = 0;                    //读者计数器
void reader(void)                     //读者进程
{
   while (1) {                        //进程被调度
      P(rmutex);                      //取得读者计数器的互斥量
      readcount = readcount + 1;      //进来一个读者，读者数量加一
      if (readcount == 1) P(wmutex);     //如果是第一个读者，取得读-写互斥量
      V(rmutex);                      //释放读者计数器的互斥量
      read_data_base();               //读数据
      P(rmutex);             //读者读完数据要离开，先取得读者计数器的互斥量
      readcount = readcount  -  1;    //读者数量减一
      if(readcount == 0) V(wmutex);
                             //如果是最后一个离开的读者，释放读-写互斥量
      V(rmutex);                      //释放读者计数器的互斥量
      use_dataread();                 //读者使用数据
   }
}
void writer(void)                     //写者进程
{
   while (1) {                        //进程得到调度
     think_up_data();                 //写者产生数据
     P(wmutex);                       //获得写-写，读-写操作互斥量
     write_data_base();               //写入数据库
     V(wmutex);                       //释放写-写，读-写操作互斥量
   }
}
```

3．哲学家进餐问题

解决办法

（1）至多只允许四个哲学家同时进餐，以保证至少有一个哲学家可以获得二只筷子：

```
typedef int semaphore;                    //定义信号量
semaphore chopstick[5]={1, 1, 1, 1, 1}; //初始化信号量
semaphore eating = 4;                     //仅允许四个哲学家可以进餐
void philosopher(int i)                   //第 i 个哲学家的程序
{
        while(1){
       thinking();                        //工作之一
       P(eating);                         //请求进餐，若是第五个则先挨饿
       P(chopstick[i]);                   //请求左手边的筷子
```

```
            P(chopstick[(i+1)%5]);          //请求右手边的筷子
            eating();                       //进餐
            V(chopstick[(i+1)%5]);          //释放右手边的筷子
            V(chopstick[i]);              //释放左手边的筷子
            V(eating);                      //释放信号量给其他挨饿的哲学家
        }
    }
```

（2）另一种解决方法，仅当哲学家的左、右两支筷子均可用时，才允许他拿起筷子进餐。

```
    typedef int semaphore;                  //定义信号量
    semaphore chopstick[5]={1, 1, 1, 1, 1}; //初始化信号量
    semaphore mutex = 1;                    //设置取筷子的信号量
    void philosopher(int i)                 //第 i 个哲学家的程序
    {
        while(1){
            thinking();
            P(mutex);                       //在取筷子前获得互斥量
            P(chopstick[i]);
            P(chopstick[(i+1)]%5);
            V(mutex);                       //释放互斥量
            eating();
            V(chopstick[(i+1)]%5);
            V(chopstick[i]);
        }
    }
```

（3）规定奇数号哲学家先拿起其左边筷子，然后再去拿右边筷子；而偶数号哲学家则相反。程序代码如下：

```
    typedef int semaphore;                  //定义信号量
    semaphore chopstick[5]={1, 1, 1, 1, 1}; //初始化信号量
    void philosopher(int i)                 //第 i 个哲学家的程序
    {
        while(1){
            thinking();
            if(i%2 == 0) {                  //偶数哲学家，先右后左
                P(chopstick[i+1]%5);
                P(chopstick[i]);
                    eating();
                V(chopstick[i+1]%5);
```

```
            V(chopstick[i]);
        }
        else{                              //奇数哲学家，先左后右
            P(chopstick[i]);
            P(chopstick[i+1]%5) ;
            eating();
            V(chopstick[i]);
            V(chopstick[i+1]%5);
        }
    }
}
```

2.4 死锁

2.4.1 死锁的概念

1．死锁的概念

系统中两个或两个以上的进程无限期地相互等待永远不会发生的条件，系统处于一种停滞状态，这种情况称为死锁。

2．死锁产生的原因

死锁的原因有以下二点：

（1）进程推进顺序不当和（2）对互斥资源的分配不当。

必须要指出的是，系统资源不足并不是产生死锁的原因，进程资源如果不足则进程就不会被创建，只有在资源部分分配以后，剩余的资源不能满足某些个进程的请求，造成进程集无法推进的现象才是死锁。

3．产生死锁的四个必要条件

互斥条件：任一时刻只允许一个进程使用资源。

非剥夺条件：进程已经占用的资源，不会被强制剥夺。

占用并请求条件：进程占有部分资源，申请更多的资源，且不会释放已经占有的资源。

循环等待：请求资源的进程形成了循环。

2.4.2 死锁处理策略

对死锁的处理，常用的方法有忽略死锁、死锁的检测与恢复、死锁的避免和死锁的预防。

2.4.3 死锁忽略

死锁忽略最典型的算法是鸵鸟算法。

2.4.4 死锁检测和恢复

1．资源分配图算法

2．资源矩阵法

3．死锁的解除与系统恢复

恢复死锁常用的方法有如下几种：

（1）资源剥夺法：挂起某些死锁进程，并抢占它的资源。

（2）进程撤销法：通过撤销占有资源多的进程或代价量小的进程，以恢复死锁。

（3）进程回退法：设置还原点，让一个或多个进程回退到足以解除死锁的地步。

（4）重新启动系统：代价最大，一切从头开始。我们要尽量避免采用此方法。

2.4.5 死锁避免

1．安全与不安全状态

某一时刻，系统能按某种顺序为每个进程分配其所需资源，使每个进程都能顺利地完成，则称此时系统处于安全状态。反之，称之为不安全状态。

2．银行家算法

银行家算法问题描述是：一个银行家把他的固定资金借给若干顾客，使这些顾客能满足对资金的要求又能完成其交易，也使银行家可以收回全部的现金。只要不出现一个顾客借走所有资金后还不够、还需要借贷，则银行家的资金应是安全的。

表 2-1 资源分配表

进程	已分配资源			尚需资源			可用资源		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	2	0	0	0	0	1	0	2	1
P2	1	2	0	1	3	2			
P3	0	1	1	1	3	1			
P4	0	0	1	2	0	0			

经过分析。

表2-2 用银行家算法进行资源分配

进程	已分配资源			尚需资源			可用资源		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P	2	0	0	0	0	1	0	2	1

1									
P4	0	0	1	2	0	0	2	2	1
P2	1	2	0	1	3	2	2	2 X	2
P3	0	1	1	1	3	1			

此时的安全序列不存在，故处于不安全状态。

2.4.6 死锁预防

所谓死锁预防，就是采用某种策略，限制并发进程对资源的请求，使系统在任何时刻都不满足死锁的四个必要条件。死锁预防主要是针对破坏四个必要条件进行的。

破坏互斥条件：某些设备可以通过 SPOOLING 系统将独享设备改造成为共享设备，以此可以解决互斥问题，例如打印机。

破坏非剥夺条件：资源暂时释放策略，申请新的资源得不到满足则暂时释放已有的资源。

破坏占用并请求条件；一次性申请全部资源。

破坏循环等待条件：资源有序申请，给资源编号，使用时按升序进行。

第三章 内存管理

考纲要求

（一） 内存管理基础

（二）虚拟内存管理

本章也是联考的重点，分值在 8-16 分间，1 题综合题，或 2-5 题选择题。

3.1 内存管理基础

3.1.1 内存管理概念

1．存储管理的功能

（1）内存空间的分配与回收，包括内存的分配和共享。

（2）地址转换：内存管理配合硬件进行地址转换，把逻辑地址转换成物理地址。

（3）内存空间的扩充：借助于虚拟存储器或交换覆盖技术来达到扩充内存容量的目的。

（4）存储保护：为了避免相互干扰和破坏，必须提供保护功能。

2．地址重定位

（1）逻辑地址空间

（2）物理地址空间

（3）地址重定位

（4）重定位类型

地址重定位分为静态重定位和动态重定位两类。把作业在装入过程中随即进行的地址变换方式，称为静态重定位。在作业执行过程中，当访问内存单元时才进行的地址变换方式，称为动态重定位。动态重定位是在程序执行过程中由硬件地址变换机构实现的。

动态重定位的主要优点如下：用户作业在执行过程中，可以动态申请存储空间和在内存中移动；有利于程序段的共享。

3．链接

（1）静态链接。

（2）装入时动态链接。

（3）运行时动态链接。

3.1.2 交换与覆盖

覆盖是指一个作业的某些程序段，或几个作业的某些部分轮流使用某一段存储空间。

交换实质上是使用外存做缓冲，让用户程序在较小的存储空间中通过不断地换出换入而可以运行较大的作业。交换可以是进程的整体交换，称为“进程交换”。如果交换以进程的部分页面或段为单位进行，则分别称之为“页面交换”或“分段交换”，又统称为“部分交换”。这种交换方法是实现请求分页及请求分段式存储管理的基础，虚拟存储系统就是采用了这种部分交换而得以实现。

3.1.3 存储分配方式

1．静态分配

在装配程序把目标模块进行链接装入时确定它们在内存中的位置。

2．动态分配

其执行过程中可根据需要申请附加的存储空间。

3.1.4 连续分配管理方式

1．固定式和可变式分区存储管理

（1）固定式分区存储管理（考纲不作要求）

（2）可变分区存储管理：根据作业的实际需要动态地划分存储空间。

（3）分配算法

首次适应算法（First Fit）

下次适应算法（Next Fit）

最佳适应算法（Best Fit）

最坏适应算法（Worst Fit）

采用“内存紧缩”技术，可以把碎片集中起来形成一个大的空闲区。

3．分区的存储保护

（1）界地址保护：界地址保护又称为界限寄存器保护。

界限寄存器方式：下界寄存器存放起始地址，上界寄存器存放结束地址。

基址寄存器和限长寄存器：基址寄存器存放起始地址，限长寄存器存放最大长度。

（2）存储键保护：同一作业的各页面所对应的内存块都要指定一个相同的，但又不与其他作业相重的键码。这个键码存于快速寄存器和该作业的程序状态字 PSW 中，当程序要访问某一块时，将程序状态字中的键码与被访问块的键码进行比较，若相符，则表明允许本次访问，否则发出越界中断。

3.1.5 非连续分配管理方式

1．简单分页存储管理

分页存储管理技术中的基本作法是：

（1）等分内存：把内存划分成大小相等的单位，称为存储块，或称为页框（Page Frame）。

（2）用户逻辑地址空间的分页：把用户的逻辑地址空间划分成若干个与存储块大小相等的单位，称之为页面或页（Page）。并给各页从起始开始依次编以连续的页号 0，1，2，……。

（3）逻辑地址的表示：用户的逻辑地址从基地址“0”开始连续编址，即相对地址。

（4）内存分配原则：系统以存储块为单位把内存分给作业或进程。

（5）页表和页表地址寄存器：作业的一页可以分配到内存空间中任何一个可用的存储块。

简单分页管理方法在作业或进程开始执行之前，把该作业或进程的程序段和数据全部装入内存的各个空闲块中，并通过页表和硬件地址变换机构实现虚拟地址到内存物理地址的地址映射。

（6）快表。

（7）页的共享与保护：保护的项目一般有读、写、运行等。

2．分段存储管理

段式存储管理按用户作业中的自然段来划分逻辑空间，例如代码段，数据段，堆栈段等等。每段占用连续的地址空间，因此作业的逻辑地址是二维的，由段号和段内地址组成。

（1）把程序按内容或过程（函数）关系分成段，每段有自己的名字。

（2）地址转换。

（3）段的共享与保护。

（4）分段存储管理特点：优点是便于程序模块化处理和便于处理变换的数据结构；便于动态链接；便于共享分段；可以实现虚拟存储器，使作业的地址空间不受内存容量的限制。

3.2 虚拟内存管理

3.2.1 虚拟内存基本概念

1．局部性原理

（1）时间局部性：程序中的某条指令一旦运行，不久以后该指令可能再次运行。产生时间局部性的典型原因是由于程序中存在着大量的循环操作。

（2）空间局部性：一旦程序访问了某个存储单元，不久以后其附近的存储单元也将枝访问，其典型情况是程序顺序运行。

2．虚拟内存

基于局部性原理，应用程序在运行之前并不必全部装入内存，仅需将当前运行到的那部分程序和数据装入内存便可启动程序的运行，其余部分仍驻留在外存上。当要运行的指令或访问的数据不在内存时，再由操作系统通过请求调入功能将它们调入内存，以使程序能继续运行。如果此时内存已满，则还需通过置换功能，将内存中暂时不用的程序或数据调至盘上，腾出足够的内存空间后，再将要访问的程序或数据调入内存，使程序继续运行。

3．实现虚拟内存的基础

硬件基础：一定容量的内存；大容量的外存；地址变换机构（含快表）；缺页中断机构。

软件基础：虚实转换的数据结构（页表、段表等）；中断服务处理程序；操作系统支持。

4．虚拟内存的主要特征

（1）多次性。

（2）对换性。

（3）虚拟性。

3.2.2 请求分页管理方式

1．请求分页的基本原理

请求分页存储管理是在简单分页管理基础上发展起来的。请求页式管理在作业或进程开始执行之前，不要求把作业或进程的程序段和数据段一次性地全部装入内存，而只把当前需要的一部分页面装入内存，其它部分在作业执行过程中需要时，再从外存上调入内存。

2．页表的扩充

如图 3-1。

读写控制	页框号

（a） 简单分页系统的页表

外存地址	其它扩展	读写控制	修改位	引用位	存在位	页框号

（b）虚拟请求分页系统的页表

图 3-1 页表的表项

（1）存在位（present/absent）：表示该页是否在内存。

（2）修改位（modified）：该位为“0”时，在示访页面中的数据未被修改过。

（3）引用位（referenced）：表示该页面在最近期间是否被访问引用过。

（4）外存地址(swap area address)：指出该页面在外存上的存放地址。

（5）其它：如页面保护位（protection），禁止缓存位（cache disabled）等。

3．地址变换

请求分页的地址变换初始过程十分类似于简单分页系统的地址变换。

4．缺页中断处理

当存在位为 0 时，表示该页不在内存，则必须确定它在外存中的存放地址。并把它从外存中调入内存。若内存中没有空闲块时，首先按照某种策略选择某页进行淘汰。以腾出空闲块供本次调入的页占用。这个过程也被称之为页面置换。若被选中淘汰的页面中的信息修改过（修改位 = 1）还必须将其写回外存。如内存中有空闲块，则根据该页在外存的地址，调入所需页面，并更新页表表项，最后恢复被中断的指令重新执行。

5．调页策略

这是一个何时把页面装入内存的问题。如果出现缺页中断，表明企图对一个不存在于内存的页面要求访问。显然，应该立即装入该页面。这种仅当需要时才调取页面的策略，称为请求式调页，采用请求式调页策略的分页系统称为请求式分页；而把事先调取页面的策略称为预调页。

3.2.3 页面置换算法

1．随机淘汰算法

在无法确定那些页被访问的概率较低时，随机地选择某个用户的页面并将其换出。

2．先进先出算法（FIFO）

FIFO(first in first out)算法：总是选择驻留内存时间最长的页面进行淘汰。其理由是：最早调入内存的页面，其不再被使用的可能性最大。

FIFO 算法忽略了一种现象的存在，就是在内存中停留时间最长的页往往也是经常被访问的页。将这些页淘汰，很可能刚置换出去，又请求调用该页，致使缺页中断较频繁，严重降低内存的利用率。

FIFO 的另一缺点是它有一种异常现象。称为 Belady 异常。

3．最佳置换算法（OPT）

最佳置换算法的基本思想是：从内存中移出永远不再需要的页面。

4．最近最久未使用页面置换算法（LRU）

这种算法的基本思想是，利用局部性原理，根据一个作业在执行过程中过去的页面访问历史来推测未来的行为。它认为过去一段时间里不曾被访问过的页面，在最近的将来可能也不会再被访问。

5．最近没有使用页面置换算法（NRU）

该算法只要求对应于每个存储块（页面）设置一个“引用位”和“修改位”。利用这二位组织成四种状态，“引用位”:“修改位”=0：0；0：1；1：0；1：1。每次置换时，总取最小值的页面置换，若相同则随机置换或先进先出置换。

6．时钟算法（CLOCK）

时钟算法是将作业已调入内存的页面链成循环队列，使用页表中的“引用位”，用一个指针指向循环队列中的下一个将被替换的页面。

3.2.4 页面分配策略

1．固定分配局部置换策略

2．可变分配全局置换策略

3．可变分配局部置换策略

3.2.5 工作集

工作集也称为驻留集，是某一个进程调入物理内存的页面的集合，这些页面是频繁地被使用到的，因此长期驻留内存是有利于提高处理机的效率。

工作集模型是基于局部性原理假设的。

3.2.6 抖动

如果分配给进程的存储块数量小于进程所需要的最小值，进程的运行将很频繁地产生缺页中断。这种频率非常高的页面置换现象称之为抖动（也称为颠簸）。往往是刚被淘汰的

页面马上被选中调页而进入内存。抖动将引起严重的系统性能下降。

防止抖动现象有多种办法，例如，采取局部替换策略，引入工作集算法，挂起或撤销若干进程等。

第四章 文件管理

考纲要求

（一） 文件系统基础

（二）文件系统实现

（三）磁盘组织与管理

本章在操作系统中也居于重点位置，历年考题在 8-16 分间，通常综合题 1 题，其它为选择题。

4.1 文件系统基础

4.1.1 文件的概念

1．文件定义

文件是具有符号名的一组信息的集合。

2．文件属性

文件的属性是指与文件记录的数据相关的所有相关信息，包括：

基本信息：文件名，文件别名，文件类型等；存储地址信息：文件物理位置，文件长度；文件访问控制信息：文件的创建者，所有者，许可那些用户能够读写或运行；文件使用信息：文件创建的时间日期，最近的使用日期时间等。

3．文件操作

对文件操作有创建文件、读文件、写文件、截断文件、设置文件的访问位置等。

对记录的操作有插入记录、修改记录、删除记录、检索记录等。

4.1.2 文件结构

1．文件逻辑结构

文件逻辑结构指用户概念中的文件，独立于物理结构，又称逻辑文件。

一般常用的文件其结构主要分为如下三类：

（1）无结构文件：把文件看作是命名了相关联的字符流集合，或称流式文件。

（2）累积文件：文件体为无结构记录序列，通过特定分隔符来划分记录，各记录大小和组成可变。新记录总是添加到文件末尾。

（3）索引文件：记录大小不必相同，不必排序，存放在主文件中。索引文件主文件不排序。另外建立索引，每个索引项指向一个记录，索引项按照记录中的某个关键字域排序。

2．文件物理结构

文件物理结构是指文件在存储介质上的组织方式，它依赖于物理的存储设备，又称物理文件。

常用的文件物理结构有：

（1）顺序结构：是把一个逻辑上连续的记录构成的文件分配到连续的物理块中。

（2）链接结构：把文件信息存放在非连续的物理块中，每个物理块均设有一个指针指向其后续连续的另一个物理块，从而使得存放同一文件的物理块链接成一个串联队列。链接方式又分为显式链接和隐式链接。显式链接的链接指针在专门的链接表中，隐式链接的指针在存放文件信息的物理块中。

（3）索引结构：指为每个文件建立一个索引表，其中每一个表项指出文件记录所在的物理块号，表项按逻辑记录编写，顺序或按记录内某一关键字顺序排列，对于大文件，为检索方便，可以建立多级索引，还可以把文件索引表也作为一个文件，称为索引表文件。

多重索引结构（混合索引结构）采用了间接索引方式，第一级索引表的表项指出下一级索引表的位置（物理块号），下一级索引表的表项指出再下一级索引表的位置，这样间接几级，最末一级索引表的表项则指向相应记录所在的物理块号。

4.1.3 目录结构

为实现“按名存取”，必须建立文件名与外存空间中的物理地址的对应关系，体现这种对应关系的数据结构称为目录。

1．文件目录管理基本要求

实现“按名存取”：用户只需提供文件名，即可对文件进行存取，这是目录管理基本功能。实现文件共享：允许不同的用户使用同一个文件。

允许文件重名：采用多级目录。

2．文件组成

文件包含两部分内容：文件说明（或称文件头）与文件体。文件体是文件本身的信息，可能是记录式文件或是字符流式文件。文件说明就是文件控制块。目录是由一组文件的文件说明（即文件控制块 FCB）组成的文件，它本身也是一种文件。

3．文件控制块（FCB）组成

（1）基本信息类：文件名、文件外存地址、文件逻辑结构、文件物理结构。

（2）存储控制信息类：文件拥有者的权限、核准用户的权限、一般用户的权限。

（3）使用信息类：文件建立的日期与时间，上一次修改的日期与时间、当前的使用信息。

4．文件目录组织形式

（1）单级目录结构

（2）二级目录结构

（3）多级目录结构

多级目录由称为树形目录，将文件的多级目录结构以图形化表示，即是图形化目录。

4.1.4 文件共享

文件共享是指在不同用户之间共同使用某些文件。实现文件共享主要有三种方式：

（1）绕道法（软链接法）：路径名是由当前目录到信息文件通路上所有各级目录的目录名加上该信息文件的符号名组成。

（2）链接法（硬链接法）：即一个目录中的一个表项直接指向被共享文件所在的目录表项，而不是直接指向文件。

（3）基本目录表方法

基本文件目录和符号文件目录结构是把所有文件目录的内容分成两部分：一部分包括文件的结构信息、物理块号、存取控制和管理信息等文件说明，并用文件系统赋予的唯一的内部标识符来标识；另一部分包括符号文件名和系统赋予的该文件的内部标识符组成。这两部分分别称为基本文件目录表(BFD)和符号文件目录表(SFD)。

4.1.5 文件保护

1．访问类型

通过限制可进行的文件访问类型，保护机制可提供控制访问（特别地为防止文件被破坏，一般对写和修改操作需要特别控制）。访问类型有：

读；写；修改；运行；添加；删除；列表清单。

2．访问控制

解决文件保护问题最为常用的是根据用户身份进行控制。实现基于身份访问的最普通方法是为每个文件或目录增加一个访问控制列表。所有用户组对文件权限的集合形成了一个二维表即文件访问控制表，不同用户对同一文件或目录需要不同类型的访问。

3．文件系统安全

为了尽量减少在系统发生故障时文件信息破坏，最简便的措施是为重要的文件保存多个副本，即“定期转储”，当系统出现故障，就可以装入转储的文件来恢复文件系统。

（1）全量转储：把文件存储器中的全部文件定期（例，每周、每天）复制到备份磁带上。

（2）增量转储：全量转储只能恢复上次转储时的状态。

4.2 文件系统实现

4.2.1 文件系统层次结构

文件系统的层次结构指明其调用结构。

4.2.2 目录实现

1．线性列表

2．散列表

4.2.3 文件实现

文件的实现是依据文件的物理结构来实施的。文件的实现要使用多个磁盘和内存结构。不同的操作系统采用不同的方法。虽然这些结构因操作系统和文件系统而异，但还是有一些通用规律。

4.3 磁盘组织与管理

4.3.1 磁盘的结构

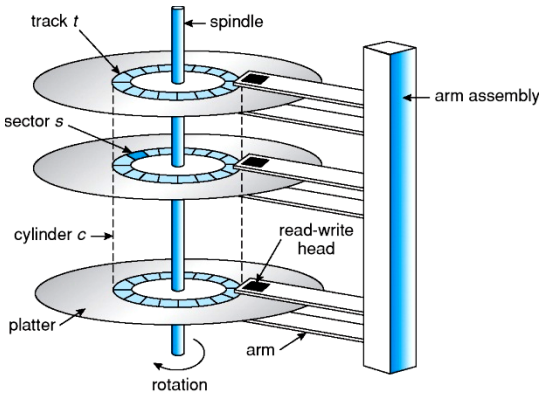

图 4-1 磁盘结构

4.3.2 磁盘调度算法

读写一次磁盘所需的时间可分为以下几种：

（1）设备等待：设备或总线忙，需要等候。

（2）寻道时间：将读/写磁头移动到相应的柱面所花费的时间。

（3）旋转延迟时间：扇区转到磁头位置所需的时间。

（4）传输时间：数据写入磁盘或从磁盘读出的时间。

常用的磁臂调度算法有：

1．先来先服务（FCFS）调度

根据进程请求访问磁盘的时间顺序，先来先服务。

2．最短寻道时间优先（SSTF）调度

根据磁头的当前位置首先将请求队列中距磁头最短的请求为之服务。

3．扫描算法（SCAN）调度

也叫“电梯”算法，磁头固定从外向内然后从内向外逐柱面运动。如此往复。

4．循环扫描（C-SCAN）调度

循环扫描算法，即磁头从盘面上的一端向另一端移动，遇到请求立即服务，返回是直接快速移至起始端，而不服务于任何请求。

5．察看（LOOK）调度

通常磁头只移动到一个方向上最远的请求为之。接着马上回头，而不是继续到磁盘的尽头。这种形式的 SCAN 和 C-SCAN 称为察看 LOOK 和循环察看 C-LOOK 调度，这是因为它们在朝一个给定方向移动前会察看是否有请求。

注意，部分教材将 SCAN 和 LOOK 算法都称为扫描算法，考生应该根据题意，合理选择相应的算法，做出符合题意的结果。

4.3.3 磁盘的管理

第五章 输入输出（I/O）管理

考纲要求

（一） I/O 管理概述

（二） I/O 核心子系统

本章在历年联考中出题量在 2-4 分，1 到 2 题选择题。

5.1 I/O 管理概述

5.1.1 I/O 控制方式

1． I/O 设备概念

I/O 设备：是指计算机系统中除控制器、运算器（中央处理机）和内存以外的所有设备，通常也称为外部设备。

I/O 操作：是内存与外设的介质之间的数据传输操作。

2． I/O 设备分类

（1）按交互对象分类：人机交互设备等。

（2）按交互方向分类：输入输出设备等。

（3）按外设特性分类：块传输设备或字符传输设备等。

3． I/O 管理目标

设备管理的功能是按照 I/O 子系统的结构和设备类型指定分配和使用设备的策略。

设备管理的目标是：提高效率；方便使用；方便控制。

4． I/O 应用接口

提供设备使用的用户接口：命令接口和编程接口。

设备分配和释放：使用设备前，需要分配设备和相应的通道、控制器。

设备的访问和控制：包括并发访问和差错处理。

I/O 缓冲和调度：目标是提高 I/O 访问效率。

5． I/O 接口控制方式

设备管理的主要任务之一是控制设备和内存或处理机之间的数据传送，外围设备和内存之间的 I/O 控制方式有四种，如下所述。

（1）程序访问控制方式

（2）中断控制方法

（3）DMA 方式

（4）通道方式

5.1.2 I/O 软件层次结构

1．I/O 软件的目标

（1）设备独立性。

（2）统一命名。

2. I/O 软件层次结构

I/O 软件中，较低的层处理与硬件有关的细节，并将硬件的特征与较高的层隔离；而较高的层则向用户提供一个友好的、清晰而规整的 I/O 接口。

一般的 I/O 软件结构分为四层：中断处理程序，设备驱动程序，与设备无关的设备独立层和用户层。

中断处理程序：I/O 设备中断方式是控制输入输出设备和内存与 CPU 之间的数据传送的主要方式。

设备驱动程序：I/O 设备驱动程序是直接同硬件打交道的软件模块。

设备独立层：在 I/O 软件中，除了设备驱动程序以外，大部分软件是与设备无关的。

用户层软件：用户使用设备的系统调用或 API。通常的 I/O 系统调用由库过程实现。

5.2 I/O 核心子系统

5.2.1 I/O 调度概念

操作系统开发人员通过为每个设备维护一个请求队列来实现调度。当一个应用程序运行阻塞 I/O 系统调用时，该请求就加到相应设备的队列上。

5.2.2 高速缓存与缓冲区

1．高速缓存

高速缓存是可以保留数据拷贝的高速内存。

2．缓冲区

缓冲区是用来保存在两设备之间或在设备和应用程序之间所传输数据的内存区域。缓冲技术可分为：

单缓冲：在设备和处理机之间只设置一个缓冲区，由输入设备和输出设备公用。

双缓冲：为输入和输出设备分配两个缓冲区，两个缓冲区交替使用。

循环缓冲：为 I/O 设备分别设置多个缓冲区，一部分专门用于输入，另一部分专门用于输出。

缓冲池：将多个缓冲区合并在一起构成公用缓冲池进行统一管理，池中的缓冲区可供多个进程共享。

5.2.3 设备分配与回收

1．设备分配概述

设备分配的任务是按照一定的策略请求设备的进程分配合适的设备及相关的硬件。

分为独占型设备、共享型设备和虚拟型设备三种。

2．设备分配用数据结构

设备的分配和管理是通过数据结构来进行的，通过它来描述系统的配置， I/O 设备的状态特性及当前的使用情况。如图 5-1 所示。虚线表示如果该设备存在的话。

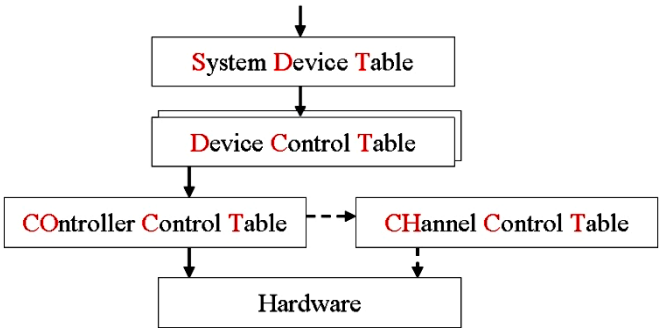

图 5-1 设备分配的数据结构

比较重要的数据结构有：

（1）系统设备表 SDT：整个系统一张系统设备表。

（2）设备控制表 DCT：系统为每一台物理设备配置一张 DCT。

（3）控制器控制表 COCT：系统为每一个控制器设置一张 COCT。

（4）通道控制表 CHCT：只在通道控制方式的系统中存在，每个通道一张。

3．设备分配策略

（1）设备分配原则：设备分配的总原则既要充分发挥设备的使用效率。又要避免造成进程死锁。

（2）设备的分配方式：静态分配主要用于对独占设备的分配，它是在用户作业开始运行之前由系统一次分配该作业所要求的全部设备、控制器和通道，直到该作业被撤销才释放。动态分配是在进程运行过程中按需进行的，进程需要设备时，向系统提出请求，系统按照事先规定的策略给进程分配所需要的设备、控制器和通道，用完之后立即释放。

（3）设备的分配策略：常用的动态设备分配策略有先请求先分配（FIFO）、优先级高者先分配等。

5.2.4 假脱机技术（SPOOLing）

SPOOLing 技术，即同时联机外围操作技术，又称假脱机技术，是指在多道程序环境下，利用多道程序中的一道或两道程序来模拟脱机输入输出中的外围控制机的功能，以达到“脱机”输入输出的目的。

1．SPOOLing 系统的组成

系统主要由以下三部分组成；

（1）输入井和输出井：这是在磁盘上开辟的两个大存储空间。

（2）输入缓冲区和输出缓冲区：这是在内存中开辟的两个缓冲区。

（3）输入进程 SP1 和输出进程 SP0：这是内存中的两个进程。

5.2.5 出错处理