细心&方法

2023年3月27日　　16:11

方法：

*选择题思维：绝对化、死板化！（完全按着笔记来，不要被绝对性或者模棱两可的词语迷惑正误性，也不要额外考虑没记到的各种可能情况！）

*PV操作代码设计：

*取值范围为0~n的信号量一般分别设emptyi和fulli两个量（此时应该有两个效果相反的操作，分别为P(emptyi)...V(fulli)和P(fulli)...V(emptyi)

*若两进程可能发生同时性（不是资源不够，只是不能同时操作！）导致的冲突，设置mutex变量避免（初始值一般设为1）~

*设完初始值后，操作部分的开头结尾加上begin/end;若为并行则改为Parbegin/end;

*一直循环用while(1)，可以用到if-else~

易错题

2023年6月2日　　21:30

13. 下列选项中，不能改善磁盘设备 I/O 性能的是（ ）。

A. 重排 I/O 请求次序　　B. 在一个磁盘上设置多个分区

C. 预读和滞后写　　D. 优化文件物理块分布

①概述

2022年12月18日　　21:18

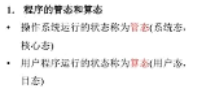
*定义：是控制和管理计算机硬件和软件资源、合理组织和管理计算机的工作流程以方便用户使用的程序的集合。
*特征：
*并发（操作系统形成的标志，不同于并行，并发在任意时刻只有一个部分在工作）、共享、虚拟、不确定性（因为并发）
并发：指两个或者多个事件在同一时间间隔内发生，是宏观上的并行和微观上的串行。
共享：系统中软硬件资源不再为某个用户（程序）独占，而是可供多个程序共同使用。
虚拟：指将一个物理实体变为若干逻辑单元。
不确定性：也称为并发程序的异步执行性，主要指程序的执行过程存在高度动态的特征，从而可能引发执行结果的不确定，操作系统必须有效解决这一问题。
*处于计算机硬件和软件之间
*操作系统发展历史：人工操作方式→脱机输入/输出技术→批处理技术→多道程序设计技术
*操作系统形成的标志：
操作系统形成的标志是：多道程序设计技术（即并发）的出现。
原因：多道程序设计技术的出现给计算机系统的管理带来了巨大挑战，如：内存共享与保护问题，处理器调度问题，I/O设备的共享及竞争问题等，为解决这些问题，形成了现代操作系统的雏形。
*分时系统：

*实时系统VS分时系统：
*各自特点：
分时系统特点：独立性，同时性，及时性，交互性。
实时系统特点：响应速度快，可靠性和安全性高。
*三个主要区别：
设计目标不同：实时系统多是专用系统，分时系统多是通用系统。
交互性强弱不同：实时系统外界操作是严格控制的，因此交互性弱。分时系统允许系统和用户之间有较强的会话能力，交互性强。
响应时间长短不同：实时系统以控制过程中信息处理能接受的延迟为标准。分时系统以人能接受的等待时间为标准。
*内核结构分类&特点：
有如下几种结构：微内核、单内核和混合结构（只写前两种也可）
微内核：内核保持尽量小，只实现操作系统的基本功能，而将更多功能放在内核之外运行，各模块之间**仅通过消息传递**进行通讯。其特点是：内核简单，安全可靠，可移植性好。
单内核：内核容纳更多的功能，分为若干个模块，模块间的通信通过**调用其它模块中的函数**实现（直接调用）。其特点为：执行效率高，但可移植性相对减弱。
混合结构：单内核和微内核的结合，取长补短，为大多数商业操作系统采用。

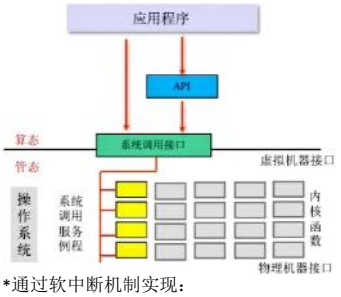
①用户接口类型

2023年2月13日　17:04

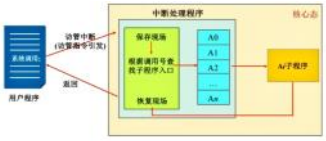
用户接口类型
（I）作业/用户控制级接口
面向人
（II）程序级接口
*操作系统留给程序员的接口是系统调用


1. 程序的管态和算态
- 操作系统运行的状态称为管态(系统态，核心态)
- 用户程序运行的状态称为算态(用户态，目态)

*系统功能调用
*定义：用户在程序中调用操作系统提供的一些子功能，是用户在程序级请求操作系统服务的一种手段。
*总体过程：



*通过软中断机制实现：



*基本过程（上图的文字描述）：执行访管指令->引发中断->进入中断处理过程->设置系统调用号和参数->保护CPU现场->查询系统调用子程序的入口地址->执行调用程序->返回结果->恢复处理器现场->返回用户态
*三种指令：
*特权指令

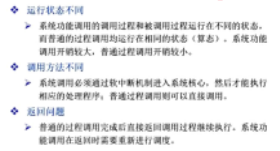
2. 特权指令
- 只能在管态下运行而不能在算态下执行的一类特殊指令。这类指令通常和硬件操作相关，如I/O指令、存取中断寄存器、时钟寄存器、地址寄存器指令，CPU清内存指令等等。

*访管指令

访管指令：也叫陷阱指令、自陷指令，是在算态下执行的一条特殊的指令，可引发中断，使得程序由算态切换到管态（自愿进管）。

*广义指令：广义指令是操作系统提供的每一个子功能（系统调用程序）被抽象成的一个系统调用命令。
*区别&联系：访管指令本身不是特权指令，它会引发访管中断，进而进入系统态执行系统调用（广义指令），在系统调用程序中可能会嵌入特权指令。

*系统功能调用与普通过程调用区别：


- 运行状态不同
  - 系统功能调用的调用过程和被调用过程运行在不同的状态。而普通的过程调用均运行在相同的状态（算态）。系统功能调用开销较大，普通过程调用开销较小。
- 调用方法不同
  - 系统调用必须通过软中断机制进入系统核心，然后才能执行相应的处理程序，普通过程调用则可以直接调用。
- 返回问题
  - 普通的过程调用完成后直接返回调用过程继续执行，系统功能调用在返回时需要重新进行调度。

*系统功能调用特点：

*必须使用的是操作系统提供的功能

*一定要有状态的变化，即由算态进入管态然后又回到算态

PS：

*API函数与系统功能调用接口不是一一对应的

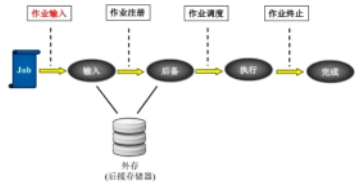
*系统功能调用是操作系统提供给应用程序的唯一接口

②作业管理

2023年2月15日　16:29

0基本概念

（I）作业的处理过程（批处理系统）（四个过程需要记住！）



转换原因：

（1）用户将自己的程序和数据提交给系统的**后援存储器(外存)**后，作业即进入输入状态；

（2）然后由作业注册程序为进入系统的作业建立**作业控制块(JCB)**，并把它加入到**后备作业队列**中，等待作业调度程序调度，这时作业处于后备状态；

（3）一个处于后备状态的作业被作业调度程序选中并分配了必要的资源，建立了一组相应的进程后，该作业进入执行状态；

（4）当程序正常运行结束**或因发生错误而终止**时，作业进入完成状态，退出系统。

（II）分类

- 脱机作业：
  - 特点：用户不直接和计算机系统交互，其执行过程由操作员辅助完成。
  - 常在批处理系统使用，也称为批量型作业/后台作业。
- 联机作业：
  - 特点：用户在作业执行过程中可直接和计算机系统交互（人机对话），控制执行的过程。
  - 多用于分时系统，也称为交互型作业/终端作业/前台作业。

①作业的I/O方式

*分为脱机、联机I/O

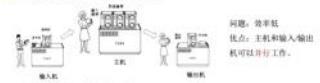
*二者区别核心：

脱机：每个机器各负责一个部分的功能，中间通过操作员连接

联机：一个机器完成I/O&运算功能：联机

*过程图解&优缺点分析：

- 脱机输入/输出(人工干预)



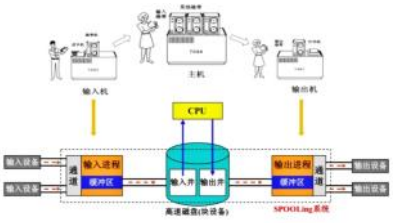
- 联机输入/输出方式

*二者结合：SPOOLing系统

*使独占设备变成了共享设备，但是必须先有独占设备才行，所以并不是不需要独占设备！

- 外围设备同时联机操作(Simultaneous Peripheral Operation On-Line)。
- 特点：兼具脱机和联机方式的优点，可以实现联机方式下的主机和外围设备的同时工作，又称为假脱机，也即以联机的方式得到脱机的效果。
- 主要技术基础：
  - 多道程序设计技术（并发）
  - 通道技术

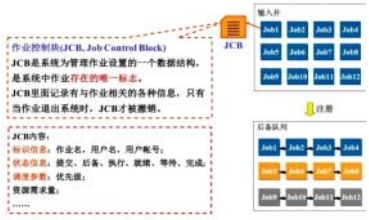
系统原理：



*作业执行时，从磁盘上的输入井读取信息，并把作业的执行结果暂时存放到磁盘上的输出井中

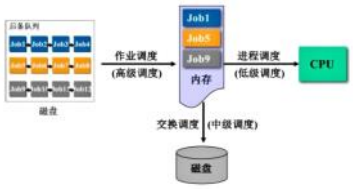
*输出设备忙时，用户程序可以继续执行自己的任务

②作业注册

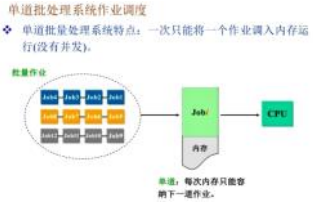


③作业调度

（I）过程图解



*典例：（多道批处理系统同理~）



（II）调度算法分类

1.单道批处理系统

*先来先服务调度算法（FCFS）

优点：简单、公平

缺点：（平均）周转时间（即被执行前等待的时间）长、吞吐量小

*最短作业优先调度算法（SJF）

优点：（FCFS的缺点反过来）

缺点：对长作业不利，出现“饥饿”现象

*最高响应比优先调度算法（HRP）

设置一个响应比（即优先级）：

RP = 作业响应时间 / 作业运行时间
= (作业等待时间+ 作业运行时间) / 作业运行时间
= 1 + (作业等待时间 / 作业运行时间)

折中之策，克服了“饥饿”现象

2.多道批处理系统

*优先级调度算法

类似单道中的最高响应比调度算法

*均衡调度算法

分类和轮流服务

（III）调度算法性能分析

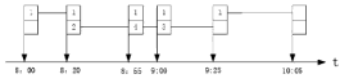
1.评价准则（执行效率用周转系数表示，系数越高效率越低）

*n道作业的操作系统：

*仍然只能单进程/作业执行，但是内存中可容纳n道作业，作业被调度进入内存后不再退出内存，每当作业进入内存时，可以调整运行的优先次序（即当前正在被执行的作业可以重新回到等待状态）

***作业调度的优先级决定进入内存的优先次序，进程调度的优先级决定内存中作业运行的优先次序**

*执行时间序列表示规范示例：



作业	提交时间	执行时间（分钟）	开始时刻	完成时刻	周转时间（分钟）
1	8:00	60	8:00	10:05	125
2	8:20	35	8:20	8:55	35
3	8:30	25	9:00	9:25	55
4	8:35	5	8:55	9:00	25
平均周转时间 T=60 分钟					240

（1）执行序列：1，2，4，3，1

（2）平均周转时间为 60 分钟

*周转时间：完成时间-提交时间（从**收容状态**就开始算起了！）

*响应时间：完成时间-开始时间

*平均周转时间最短的方法是最短作业优先调度算法SJF

*（IV）调度

一般来说，作业进入系统到最后完成，可能经历三级调度：高级调度、中级调度和低级调度。

1.高级调度（作业调度）

按一定的原则从外存上处于后备队列的作业中挑选一个（或多个）作业，给他们分配内存等必要资源，并建立相应的进程(建立PCB),以使它（们）获得竞争处理机的权利。高级调度是辅存（外存）与内存之间的调度。每个作业只调入一次，调出一次。作业调入时会建立相应的PCB,作业调出时才撤销PCB。高级调度主要是指调入的问题，因为只有调入的时机需要操作系统来确定，但调出的时机必然是作业运行结束才调出。

2.中级调度（内存调度）

为了使内存中的内存不至于太多，有时需要把某些进程从内存中调到外存。在内存使用情况紧张时，将一些暂时不能运行的进程从内存中对换到外存中等待。当内存有足够的空闲空间时，再将合适的进程重新换入内存。

3.低级调度（进程调度）

主要任务是按照某种方法和策略从就绪队列中选取一个进程，将处理机分配给它。进程调度是操作系统中最基本的一种调度，在一般的操作系统中都必须配置进程调度。进程调度的频率很高，一般几十毫秒一次。

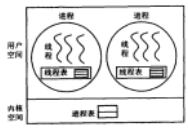
①基本概念、产生&消失

2023年2月22日　10:46

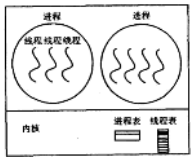
①基本概念
*进程=程序+执行
*组成结构：

*和程序的区别&联系：
*区别：
1. 程序是静态的，是有序代码的集合；进程是动态的，是程序的一次执行。
2. 程序的永久的，没有生命周期，可长久保存；进程是暂时的，有生命周期，是一个动态不断变化的过程。
3. 进程是操作系统资源分配和保护的基本单位；程序没有此功能。
4. 程序与进程的结构不同。
*联系：
一个程序可以生成多个进程；一个进程也可包含多个程序。

*线程:
*定义：线程是操作系统能够进行运算调度的最小单位。线程包含在进程之中，是进程中的实际运作单位。一个进程中可以并发多个线程，每条线程并行执行不同的任务。
*分类：
*用户级线程ULT：
*应用程序管理线程，核心感觉不到线程的存在
*当线程执行系统调用或者阻塞时，整个进程都被阻塞，不能充分利用多处理机

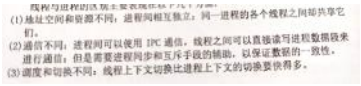


*核心级线程KLT（内核线程）：
*同一进程控制权转移时，用户级与核心级的切换开销很大
*某一线程阻塞时，其他线程和进程都不会阻塞



*与进程的关系:
1. 一个进程可以有多个线程，但至少有一个线程。
2. 线程只能在所属进程的内存空间内活动，他们共享该进程所有资源。
3. 进程将CPU资源分配给线程，即真正在处理器运行的是线程。
4. 线程在执行过程中需要协作同步，不同进程的线程间要利用消息通信的办法实现同步。

*与进程的区别:


(1)地址空间和资源不同，进程间相互独立；同一进程的各个线程之间都共享它们。
(2)通信不同：进程间可以使用 IPC 通信，线程之间可以直接读写进程数据段来进行通信；但是需要进程同步和互斥手段的辅助，以保证数据的一致性。
(3)调度和切换不同：线程上下文切换比进程上下文的切换要快得多。

*实现方式:
1.实验内核线程实现
2.实验用户线程实现
3.实验用户线程加轻量级进程混合实现

*同一进程的各个线程地址空间不同!

②进程的产生与消失
*PCB结构：

类型	内容	作用
标识信息	• 进程标识符（ID）； • 父进程标识符（ID）； • 用户标识（User ID）。	标识一个进程
状态（现场）信息	• 控制和状态寄存器，程序计数器（PC），CPU状态寄存器； • 用户可见寄存器； • 栈指针。	记录处理机上下文信息，以备中断恢复之用
控制信息	• 调度和状态信息，进程状态，优先级，调度相关信息； • 数据结构：进程的组织方式（队列，环…） • 进程间通信，进程通信的信息，如消息队列，互斥，同步； • 进程特权； • 存储管理，虚存段表/页表指针； • 资源使用权和使用情况，进程控制资源，如打开文件等。	用于进程的调度管理

三种类型的具体作用（上图内容的扩充，供简答题参考）：
标识信息：唯一的标识一个进程，主要包含进程标识、用户标识、父进程标识。
现场信息：记录进程使用处理器时的各种现场信息。主要有CPU通用寄存器的内容、CPU状态寄存器内容及栈指针等信息。

控制信息：操作系统对进程进行调度管理时用到的信息，主要有进程状态、调度信息、数据结构信息、队列指针、位置信息、通信信息、特权信息、存储信息等。

*系统管理控制块的方法：进程控制块PCB在内存中是以表的形式存在的，操作系统对PCB进行集中统一的管理，所有的PCB集中在一个**固定的存储空间**上，形成了**PCB表**。PCB之间是以**双向链式队列**的形式关联的

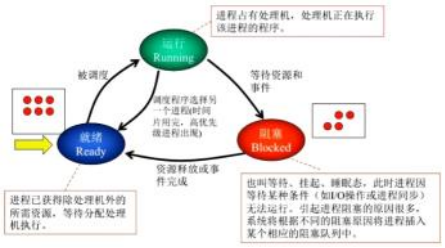
- 进程的产生
  - 系统初始化
  - 用户执行程序（命令，双击）
  - 程序启动程序（子进程）
  - 批处理系统，作业初始化
- 进程的消失
  - 寿终，运行结束而退出
  - 自杀，因错误而自行终止
  - 他杀，被其他进程/用户强行终止
  - 处决，因异常而被系统强行终结

②进程的执行&控制

2023年3月22日　　17:01

进程的执行与控制

（I）进程的基本状态及其转换



*转换原因（上图内容的扩充，供简答题参考）：

(1)就绪->执行

处于就绪状态的进程，当被进程调度程序选中，且为之分配了处理器后，该进程便由就绪状态转变成执行状态。

(2)执行->就绪

处于执行状态的进程在其执行过程中，因分配给它的时间片已用完或处理器被更高优先级进程抢占而不得不让出处理器，便从执行状态转变成就绪状态。

(3)执行->阻塞

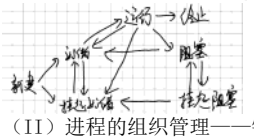
正在执行的进程因等待系统分配资源或后写事件发生而无法继续执行时，便从执行状态变成阻塞状态

(4)阻塞->就绪

处于阻塞状态的进程，若其等待的事件已发生或等待的资源可用时，由阻塞状态转变为就绪状态。

*进程阻塞时会引发一次处理器调度

*加上挂起就绪、挂起阻塞状态（外存中的运行、阻塞状态）后的状态转移图：



（II）进程的组织管理——链式队列

（III）进程控制

通过操作系统内核中的原语实现：

- 原语
  - 由若干条指令构成的可完成特定功能的程序段，必须在管态下运行，它是一个“原子操作(atomic operation)”过程，执行过程不能被中断。
  - 原语的原子性主要是通过屏蔽中断或原语固化保证的。
- 原语分类
  - 进程创建、撤销、阻塞、唤醒、挂起、激活原语。

③进程调度

2023年3月22日　　17:03

①基本概念

❖ 就是按照一定的算法，从就绪队列中选择某个进程占用CPU的方法。

*需要考虑的因素：系统对响应时间T的要求、就绪队列中的进程数目、系统的处理能力（用户的忍耐度）

②交互式系统（分时系统）下的调度策略

1.时间片轮转法（RR）（不会加快某个程序的执行效率！）

➢ 思想：系统规定一个时间长度(时间片)作为允许一个进程运行的时间，如果在这段时间该进程没有执行完，则必须让出CPU给下一个进程使用，自己则排到就绪队列末尾，等待下一次调度；如果时间片结束之前被阻塞或结束，则CPU立即切换。

2.优先级调度算法

*分为静态（优先级固定）、动态（优先级会变）两种，内容同作业管理中的优先级调度算法

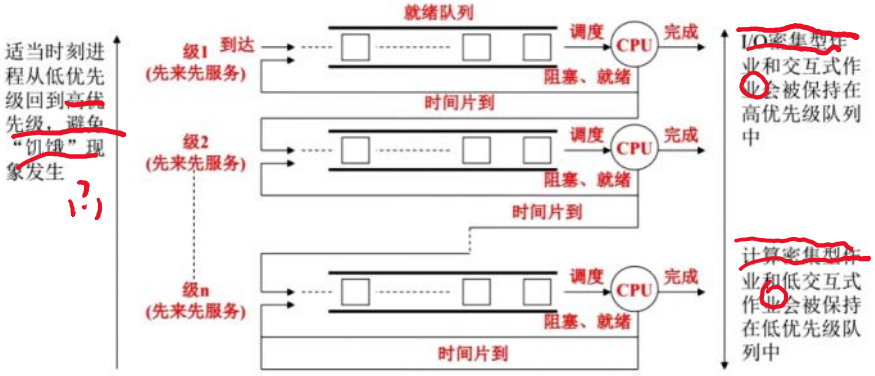
*进程处于临界区（一段用于同步的数据共享区）中时也可能可以进行处理机调度

3.多级反馈队列调度算法（MFQS）

*基本思想：

- 系统中维持多个不同优先级的就绪队列。
- 每个就绪队列具有不同长度的时间片。优先级高的就绪队列里的进程，获得的时间片短；优先级低的就绪队列里的进程，获得的时间片长。
- 新进程进入时加入优先级最高的就绪队列的末尾。

*图解：



多级反馈队列调度算法是目前最为通用的CPU调度策略，经过适当的配置和改进即可适应不同的系统！

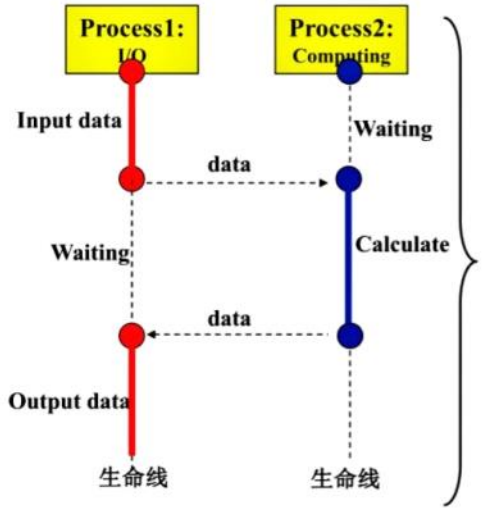
④进程间的相互作用

2023年3月22日　　17:03

进程间的相互作用

*介绍

1.并发

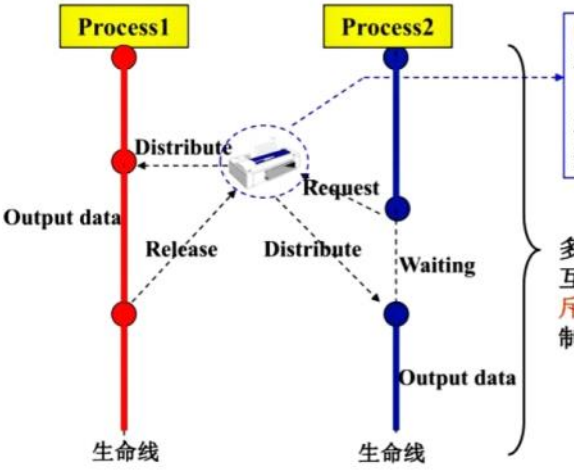


进程之间这种相互合作、协同工作的关系称为进程的同步。

简单说来就是：多个相关进程在执行次序上的协调。

制约关系：直接制约。

2.并发时可能产生的竞争



临界资源：也称独占资源，是指在一段时间内只允许一个进程访问的资源。例如打印机，磁带机，也可以是进程共享的数据、变量等。

多个进程因为争夺临界资源而相互排斥执行的过程称为进程的互斥。

制约关系：间接制约。

*解决完全同时竞争带来的问题：

1.加锁法——自旋锁

- 做法：设置一个共享变量W (锁)，初值为0。当一个进程想进入其临界区(使用临界资源的程序段)时，它首先测试这把锁：如果锁的值为0，则进程将其置为1并进入临界区。若锁已经为1，则进程等待直到其变成0。
- 实现：
  - 加锁原语LOCK(W)：L: if W=1 then goto L else W=1;
  - 解锁原语UNLOCK(W)：W=0;

现象：进程会一直处于忙等待状态(Busy waiting)。

*适合所需等待时间较短的情况，较长时则会产生CPU的无效浪费

2.信号量和P(down)、V(up)操作

*信号量：

*表示资源的实体，是一个与队列有关的整型变量

*其值只能通过初始化操作和P、V操作来访问

信号量的类型：

•公用信号量，用于进程间的互斥，初值通常为1；

信号量的类型：

•公用信号量：用于进程间的互斥，初值通常为1；

•私有信号量：用于进程间的同步，初值通常为0或n。

*值正负性的意义：

*为正值+N则表示当前仍可分配N个资源

*为0则表示所有资源都已分配出去，且当前没有进程处于等待状态

*为负值-N则表示所有资源都已分配出去，且当前有N个机场呢处于等待状态

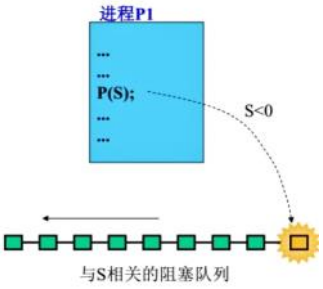
*一般来说，有n个进程共享互斥段、互斥段中最多同时运行m个进程的情况下，信号量初值设为m，其变化范围为m,m-1,…,1,0,…,-(n-m)

*P/down操作：

*尝试，即请求分配一个资源或进行测试

*原语操作：

```
P(S):  //S为信号量
{
   S = S - 1;
   if (S < 0)
   {
      调用进程被阻塞，
      进入S的等待队列；
   }
}
```

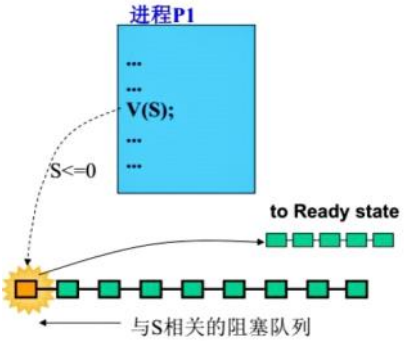


*V/up操作：（和P/down操作相反）

*增量/升高，即释放/增加一个单位资源

*原语操作：

```
V(S):  //S为信号量
{
   S = S + 1;
   if (S <= 0)
   {
      从S的等待队列中唤醒一个进程
      使其进入就绪状态；
   }
}
```



*典型问题：

1生产者-消费者问题

信号量设置：

```
semaphore mutex=1   //互斥
semaphore empty=N   //缓冲区空闲数, 同步
semaphore full=0    //产品数量，同步
```

```
生产者：
while(1) {
   produce;
   P(empty);
   P(mutex);
   insert to buffer;
   V(mutex);
   V(full);
}
```

```
消费者：
while(1) {
   P(full);
   P(mutex);
   get from buffer;
   V(mutex);
   V(empty);
   consume;
}
```

2读者-写者问题

设置信号量：

```
int  readnum=0; //计数，用于记录读者的数目
semaphore mutex=1; //公用信号量，用于readnum互斥
semaphore write=1; //公用信号量，用于Data访问的互斥
```

```
读者进程代码：

P(mutex); //对readnum互斥
readnum ++; //读者数目加1
if (readnum==1)  //第一个读进程
   P(write); //申请使用data资源
V(mutex); //释放readnum
reading;
P(mutex); //对readnum互斥
readnum --;
if (readnum==0)  //最后一个读进程
   V(write); //释放data资源
V(mutex); //释放readnum
```

```
写者进程代码：

P(write); //申请使用data资源
writing;
V(write); //释放data资源
```

*注意事项：

PV总是成对出现，互斥操作时他们处于同一进程中，同步操作时处于不同进程中。

3.管程

*定义&特点：

- 管程的定义：
  - 关于共享资源的一组数据结构和在这组数据结构上的一组相关操作。
- 管程的思想——集中式同步机制
  - 将共享变量以及对共享变量能够进行的所有操作集中在一个模块中，以过程调用的形式提供给进程使用。
- 使用的互斥性：任何一个时刻，管程只能有一个活跃进程，进入管程时的互斥由编译器负责完成。

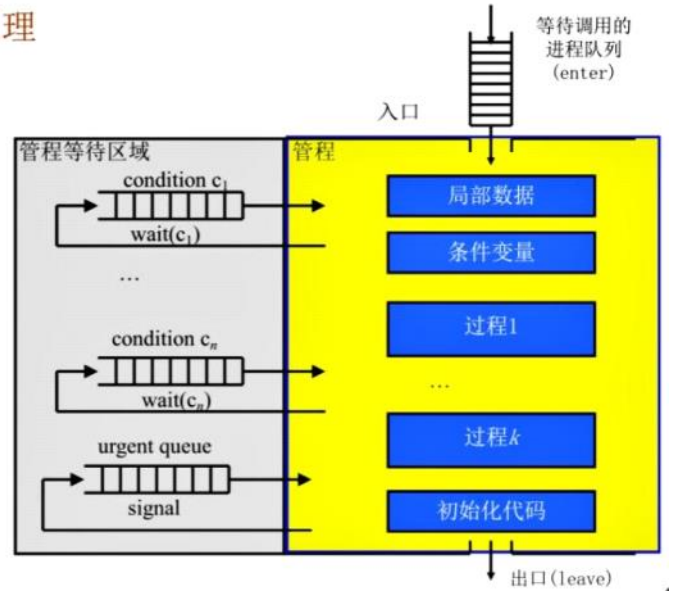
*工作原理

管程的工作原理

条件变量(c)：是管程内的一种数据结构，只在管程中才能被访问，通过两个原语wait,signal操作来控制它。

wait(c)：在c上阻塞调用进程，直到另一个进程在该条件变量上执行signal()。

signal(c)：在c上执行signal操作，如果存在其他进程由于c而被阻塞，唤醒之。



⑤进程通信

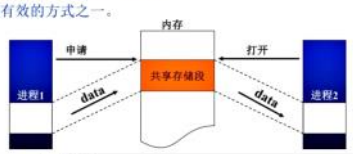
2023年3月22日　　17:04

进程通信（IPC）

- 进程通信(IPC)：简单来说就是在进程间传输数据/交换信息。
- 根据交换信息量的多少和效率的高低，分为：

1.共享内存模式

- 使用一段共享的内存区域交换数据。
- 最为快捷有效的方式之一。
- 原理：



优点：
- 能够传输大量复杂数据

局限：
- 系统本身不提供对内存共享存储区的互斥控制，因此互斥要通过其它机制实现；
- 数据的发送方不关心数据由谁接收，数据的接收方也不关心数据是由谁发送的，存在安全隐患。容易造成病毒的传播。

2.信息传递模式

- 消息传递的方式
  - 直接通信方式：点到点的发送。

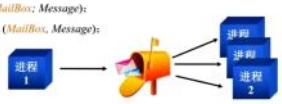
    Send (*DestProcessName*, *Message*);

    Receive (*SourceProcessName*, *Message*);

  - 间接通信方式：以信箱为媒介进行传递，可以广播。

    Send (*MailBox*; *Message*);

    Receive (*MailBox*, *Message*);



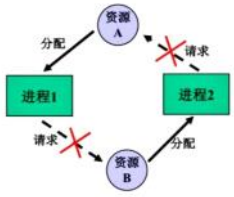
3.共享文件模式：管道（pipe）

- 管道是一种信息流缓冲机构(线性字节数组)，使用文件读写方式进行访问。但管道并不是文件。
- 以先进先出(FIFO)方式的单向传送数据。
- 匿名管道(Anonymous Pipes)
  - 通常用来在父进程和子进程之间传输数据。匿名管道总是本地的，不能在网络之间传递数据。
- 命名管道(Named Pipes)
  - 命名管道是一种有名称的，可在同一台计算机的不同进程之间或跨越一个网络的计算机的进程之间传输数据。

①基本概念

2023年3月22日　　16:36

*定义（Deadlock）



在多道程序中，由于多个并发进程共享系统资源，如果使用不当可能会造成
一种僵局，即当某个进程提出资源的使用请求后，使得系统中一些进程处于
无休止的阻塞状态，在无外力的作用下，这些进程将无法继续执行下去，这
就是**死锁**。

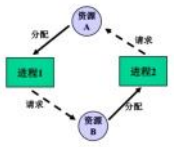
*死锁是一种小概率事件！
*死锁发生时一定会检测到进程资源构成的环！
*死锁与时间无关！

②环境&条件

2023年4月20日　　12:28

*环境&条件

- 死锁产生的必要条件(Coffman 1971年提出)
  - 资源互斥使用(资源独占)
  - 非剥夺控制(不可强占)
  - 零散请求与保持
  - 循环等待



- 死锁的危害
  - 一旦发生：轻则系统资源利用率下降，重则系统崩溃。

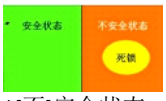
CPU不可能引发死锁（因为它不是资源互斥的）！

组织环中对互斥+非剥夺的资源零散请求为保持.

③解决策略

2023年4月20日　　12:27

（I）置之不理法（鸵鸟政策）
（II）事后处理法
*但不是所有情况都能容忍
（III）积极防御法
手段：预防（使其根本不可能发生）、避免（存在发生的可能但是绕开）
1.预防
*破坏互斥条件
*破坏不可剥夺条件：允许一个进程还未执行完成时释放已经占有的资源（被剥夺使用权）
*破坏零散请求（有的地方又称保持和等待）条件：进程创建时就由系统分配了所有需要的资源，进程执行完后释放资源（如资源静态分配法）
*破坏循环等待条件：给资源编号
这些预防方法的效果：由于对资源的申请加上了诸多的限制，因此这种策略虽有一定的效果，但会导致资源利用率或进程运行效率降低，很难令人满意。
2.避免
*思想：在分配资源之前先确定不会发生死锁
*



*[不]安全状态：当多个进程动态地申请资源时，系统将按照某种顺序逐次地为每个进程分配所需资源，使每个进程都可以在最终得到最大需求量后，依次顺利完成。反之，如果不存在这样一种分配顺序是的进程都能顺利完成，则称系统处于不安全状态。
*单/多银行家算法：
*多银行家算法思想和单银行家算法一样，但分别用向量、整数来运算
*规范：

进程	最大资源需求量			已分配资源数量			剩余需求量			系统剩余资源		
	A	B	C	A	B	C	A	B	C			
P1	5	5	9	2	1	2	3	4	7	2	3	3
P2	5	3	6	4	0	2	1	3	4			
P3	4	0	11	4	0	5	0	0	6			
P4	4	2	5	2	0	4	2	2	1			
P5	4	2	4	3	1	4	1	1	0			

进程	最大资源需求量			已分配资源数量			剩余需求量			系统剩余资源		
	A	B	C	A	B	C	A	B	C			
P1	5	5	9	2	1	2	3	4	7	4	3	7
P2	5	3	6	4	0	2	1	3	4			
P3	4	0	11	4	0	5	0	0	6			
P4												
P5	4	2	4	3	1	4	1	1	0			

④死锁的检测和解除

2023年4月20日　　12:27

（I）检测

1.永久资源的死锁检测

*资源分配图：

(1) 圆表示一个进程。
(2) 方块表示一个资源类，其中的圆点表示该类型资源中的单个资源。
(3) 从资源指向进程的箭头表示资源被分配给了这个进程。
(4) 从进程指向资源的箭头表示进程申请一个这类资源。

*应满足条件：

(1) 资源 R_j 分配给各进程的资源数目不能大于 W_j，即对于各类资源 R_j 的分配应满足：

$$\sum_i |(R_j, P_i)| \leqslant W_j$$

(2) 任何一个进程 P_i 对某类资源 R_j 的申请量和已分配数量之和，不应大于该类资源的总数 W_j，即

$$|(P_i, R_j)| + |(R_j, P_i)| \leqslant W_j$$

P_i:表示第i个进程，R_j表示第j个资源；
$|(P_i, R_j)|$表示P_i申请R_j资源的个数；
$|(R_j, P_i)|$表示R_j已分配给P_i资源的个数。
W_j表示R_j类资源的个数

*死锁定理：

资源分配图中的所有进程如果都能化简成孤立结点，则这个资源图就是可完全化简的（completely reducible）；反之，就是不可完全化简的（irreducible）。

死锁定理：如果一个系统状态为死锁状态，当且仅当资源分配图是不可完全化简。也即，如果资源图中所有的进程都成为孤立结点，则系统不会死锁；否则系统状态为死锁状态。

*检测步骤：

(1) 检查图中有无环路，如果没有，系统不会发生死锁，结束检测；如果有环路，进行第(2)步。
(2) 若环路中涉及的每个资源类中只有一个资源，系统一定死锁；若每个资源类中有多个资源，进行第(3)步。
(3) 在环路中查找非阻塞且非独立的进程 P_i，应满足：

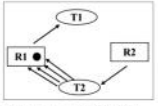
$$|(P_i, R_j)| + \sum_k |(R_j, P_k)| \leqslant W_j$$

即它可以在有限的时间里将获得申请的资源执行完毕，从而释放进程占据的所有资源。找到后，把与该进程相连的所有有向边去掉，形成孤立节点。如此反复执行步骤(3)，直到没有进程可被化简。

2.临时资源的死锁检测

*概念

- 临时性资源：即可消耗的资源。如信号、消息、邮件等。
- 特点：没有固定数目；不需要释放。
- 表述方式——重定义的资源分配图
  - 规则：
    ① 圆表示一个进程；
    ② 方块表示一个资源类，其中的圆点表示该类型资源中的单个资源；
    ③ 由进程指向资源的箭头表示该进程申请这种资源，一个箭头只表示申请一个资源；
    ④ 由资源类指向进程的箭头表示该进程产生这种资源，一个箭头可表示产生一到多个资源，每个资源类至少有一个生产者进程。

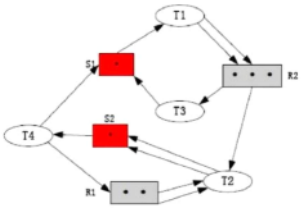


*临时资源与永久资源的区别在于生产者进程可以源源不断地生产临时资源，供需要的进程使用

*永久资源跟临时资源一样，也用框里面标注点来表示，但是框里填充颜色不相同（比如可以打个阴影）

*进程占有n个永久/临时资源的表示方法同产生，即资源的n个箭头都指向进程

*示例：



*判断方法：

判断系统是否死锁的关键在于判断生产者进程的状态，若生产者进程不被阻塞，则可以认为它总会生产出该类资源，也就是说，申请这类资源的所有申请者进程都可以得到满足。

*检测步骤：

① 从那些没有阻塞的进程入手，删除那些没有阻塞的进程的请求边，并使资源类中资源数（图中黑点的数目）减1。

② 重复以上步骤直至：

a. 图中所有的请求边都已经删除，则不会死锁

b. 图中仍然存在请求边但无法再化简，系统死锁

（II）解除

- 重新启动
  - 这是一种常用但比较粗暴的方法，虽然实现简单，但会使之前的工作全部白费，造成很大的损失和浪费。
- 撤消进程
  - 死锁发生时，系统撤消造成死锁的进程，解除死锁。
  - 一次性撤消所有的死锁进程。损失较大。
  - 逐个撤消，分别收回资源。具体做法：系统可以先撤消那些优先级低的、已占有资源少或已运行时间短的、还需运行时间较长的进程，尽量减少系统的损失。
- 剥夺资源
  - 死锁时，系统保留死锁进程，只剥夺死锁进程占有的资源，直到解除死锁。选择被剥夺资源进程的方法和选择被撤消进程相同。
- 进程回退
  - 死锁时，系统可以根据保留的历史信息，让死锁的进程从当前状态向后退回到某种状态，直到死锁解除。
  - 实现方法：可以通过结合检查点或回退（Checkpoint/Rollback）机制实现。进程某一时刻的瞬间状态叫做检查点，可以定期设置检查点。一旦系统检查到有某个进程卷入了死锁，系统查看保存的检查点信息，重新建立该进程的状态，从上次检查点的位置重新执行。

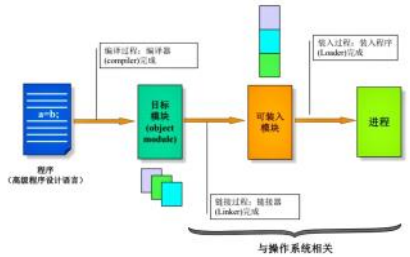
①计算机存储体系结构

2023年4月3日　　17:17

*存储器是计算机体系结构重要组成部分，功能是保存指令和数据
*寄存器与外存（不是内存！）之间的访问速度差异在百万倍量级

②程序的转化过程

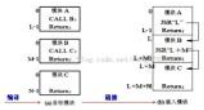
2023年4月10日　　17:17



*编译过程不属于操作系统的功能范畴

（I）链接过程

- 链接：将编译后的目标模块以及需要的相关部分（如库函数）链接在一起，形成一个完整的装入模块（可执行程序）的过程。
- 程序链接的三种方式
  - 静态链接
  - 装入时动态链接
  - 运行时动态链接



（II）程序的装入

*当程序被装入内存时，程序的逻辑地址被转换成内存的物理地址这一过程称为地址重定位

1.绝对装入/固定地址再定位

程序的装载地址（物理地址）是在编译链接时直接生成的。

此时程序的逻辑地址与装入内存的物理地址一致。

优点：装入过程简单。
缺点：编译时必须知道模块装载到内存的位置，实现较为困难，不适合于多进程并发环境。

以下2&3统一归为地址重定位/可重定位装入：

2.静态重定位

允许程序装入内存中的任意位置，因此地址的重定位是在装入时完成的，且只进行一次，在程序的执行期间地址将不再发生变化，因此也称为静态重定位。

→指装入内存时完成地址链接

优点：支持多个程序并发运行；编译器无需考虑物理地址。

缺点：程序在执行期间不能在内存中调整位置，降低了内存的利用率。

3.运行时重定位/动态重定位

程序在装入内存时，不修改程序的逻辑地址值，程序在访问物理内存之前，再实时地将逻辑地址转换成物理地址。

→程序运行过程中

优点：代码可以在内存中移动，灵活，在现代操作系统中得到广泛应用

③分区存储管理方案

2023年4月10日　　17:18

0基本概念

- 连续分配方式
  - 分区存储管理方案
- 离散分配方式
  - 页式存储管理方案
  - 段式存储管理方案
  - 段页式存储管理方案
- 内存扩充技术（离散不连续方式）
  - 交换和覆盖技术
  - 虚拟存储管理方案

*碎片：

*内碎片：内部碎片就是已经被分配出去（能明确指出属于哪个进程）却不能被利用的内存空间；

*外碎片：外部碎片指的是还没有被分配出去（不属于任何进程），但由于太小了无法分配给申请内存空间的新进程的内存空闲区域。

①连续分配方式

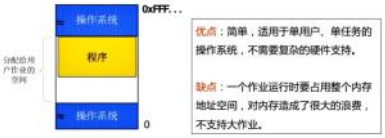
*局限：

作业存储时必须连续存放。这样当一个作业大于当前最大的空闲分区容量时，即使主存中有空闲区域也不能够利用从而降低了存储空间利用率。

（I）连续分配方式（数组实现）

*不支持并发

- 特点：内存中一次只能装入一个用户程序，程序独占整个用户区。



（II）固定分区管理

- 实现方法：

（III）动态分区管理（链表实现）

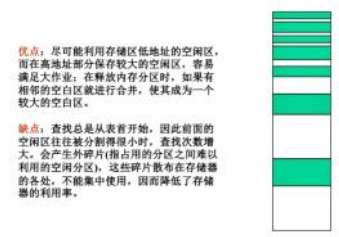
思想：预先不划分内存，当作业需要装载时系统从内存中分出一块，其大小等于该作业的大小，然后将剩下的部分再作为空白块给下一次分配使用。

*分区分配算法：

对大作业有利的是：邻近适应算法、最佳适应算法

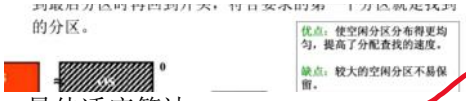
1.最先（首次）适应算法

- 分配方法：将所有的空闲分区按照地址递增的顺序排列，头开始查找，符合要求的第一个分区就是要找的分区。

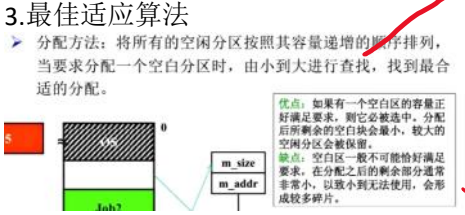


2.邻近（下次/循环）适应算法

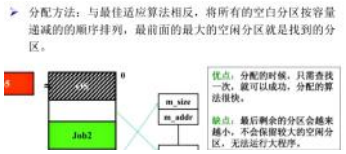
- 分配方法：按分区的先后次序，从上次分配的分区起查找，到最后分区时再回到开头，符合要求的第一个分区就是找到的分区。



3.最佳适应算法

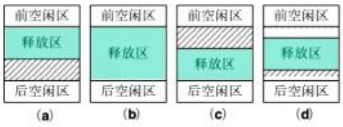


4.最坏适应算法



按大小.

*分区释放方式：

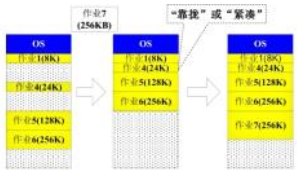


规则：相邻合并，否则插入

前三者有合并（即链表的增删等操作）过程。

*解决碎片问题的策略——紧凑

基本思想：移动所有被分配的分区，使之成为一个连续区域，而留下一个较大的空白区。



*并不能消除碎片

②离散分配方式

*快表：页面变换表/页表等等

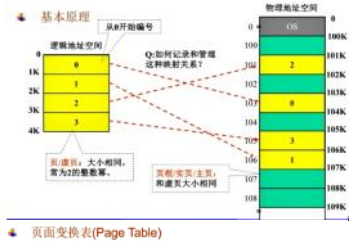
*硬件单元在CPU中封装

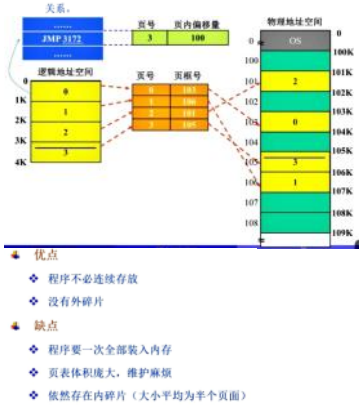
*各行具有并行查找能力

*管理通过操作系统进行

*命中率可达90%以上

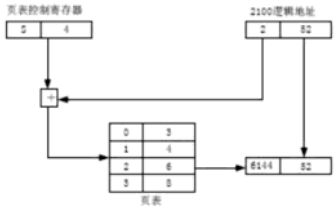
（I）页式存储管理——等分内存空间



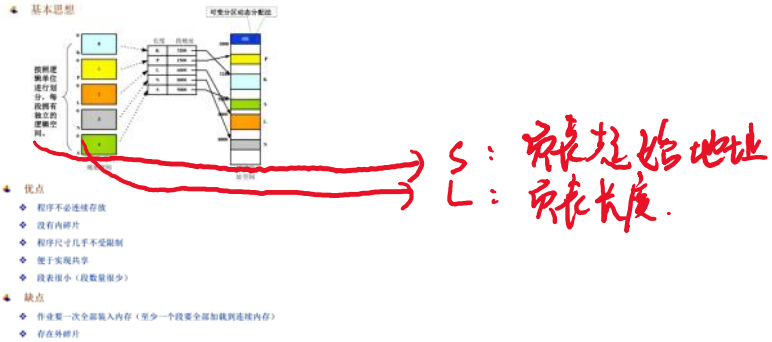


*局限性：不便于实现共享、一个程序只有一个虚拟地址空间

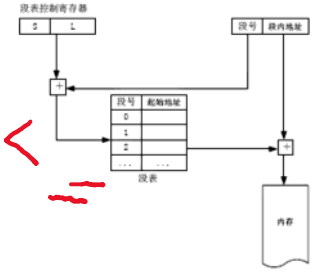
*访问一个地址空间时，实际需要访问内存两次（查页表→找物理地址）
*采用动态地址重定位技术
*缺页不能通过操作系统的预测发现
*若题目没说，默认页面大小等于主存分块的大小，逻辑地址以KB为单位划分
*表示地址变换过程的地址变换图规范：



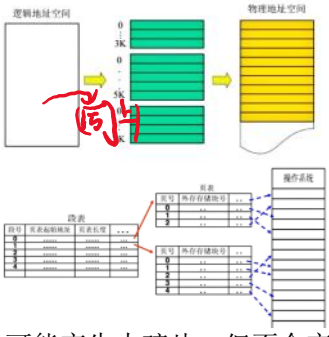
（II）段式存储管理



*段表长度较短，是因为段的尺寸较大
*访问一个地址空间时，实际也需要访问内存两次（同上，查段表→找物理地址）
*表示地址变换过程的地址变换图规范：



（III）段页式存储管理



*可能产生内碎片，但不会产生外碎片
*每访问一条指令需要访问内存3次（段表、页表、内存地址对应内容）
（IV）比较

内容	页式存储管理	段式存储管理
划分依据	系统管理需要	用户应用需要
页/段大小	各页面大小相同	段的大小不固定
逻辑地址	只有一个逻辑地址空间	每个段一个独立的逻辑地址空间
页表/段表	页面较多，页表较长，查找费时	段较少，段表较短，查找速度快
碎片	存在内碎片	存在外碎片
内存共享	不支持	支持
存储扩充	不支持	不支持

*n级页表索引结构：访问一条指令或数据需要访问n+1次内存

*内外碎片结论整理：
固定分区管理方案：内碎片
可变分区管理方案：外碎片

页式管理方案：内碎片
段式管理方案：外碎片
段页式管理方案：内碎片

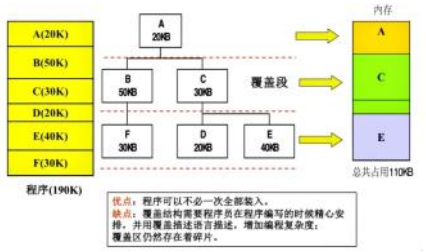
④内存扩充技术

2023年4月17日　　17:19

①基本概念

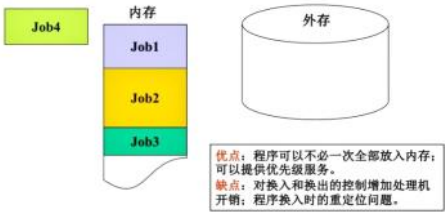
- 提出原因
  - 在基本的存储管理系统中，当一个作业的程序地址空间大于内存可以使用的空间时，该作业就不能装入运行，并发运行进程数受到了内存空间的限制。
- 内存扩充技术
  - 就是借助大容量的辅存在逻辑上实现内存的扩充，来解决内存容量不足的问题。

②覆盖技术



③交换技术

- 原理



④覆盖&交换技术的区别

内容	覆盖技术	交换技术
适用情况	作业内部	作业之间
对程序结构的影响	有	无

⑤虚拟存储技术

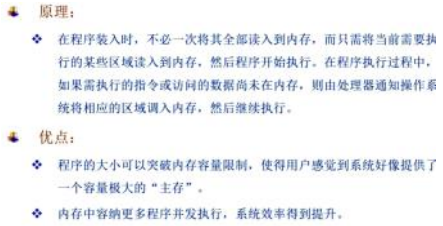
2023年4月17日　　17:23

①基本概念

（I）程序执行规律

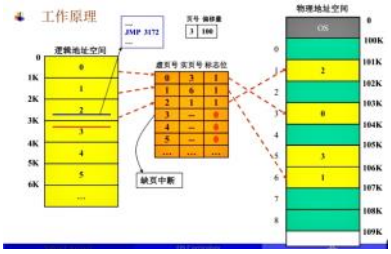
*程序执行的**局部性原理**：作业一般不会执行到所有程序的指令，也不会存取绝大部分数据

（II）原理&优点

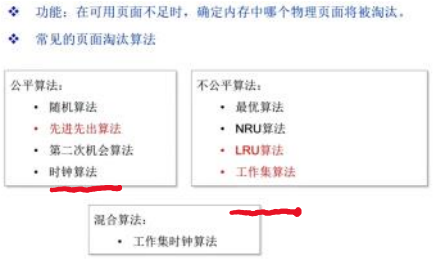


*缺页中断时，系统将寻找一个空白页将程序装入

②工作原理



③页面淘汰算法（同计组，但格式规范不同）



*缺页率：未命中次数/总次数

*格式规范：

时刻	1	2	3	4	5	6	7	8	9	10	11	12
P	4	3	2	1	4	3	5	4	3	2	1	5
M	4'	3' 4	2' 3 ④	1' 2 ⑧	4' 1 ②	3' 4 ①	5' 3 4	5 3 4	5 3 ④	2' 5 ⑧	1' 2 5	1 2 5
F	+	+	+	+	+	+	+			+	+	

缺页中断次数F=9，而缺页率f=9/12=75%

*增加缺页率可能的方法：增加主存容量M

*Belady现象：可用页面增大，缺页率反而升高

④颠簸/抖动

（I）定义

虚存中，页面在内存与外存之间频繁调度，进程的运行将很频繁地产生缺页中断，此时系统效率急剧下降，甚至导致系统崩溃。这种频率非常高的页面置换现象称为抖动。

（II）原因

*直接原因：页面淘汰算法不合理

*根本原因：分配给进程的物理页面太少、物理外存大小不足

（III）解决方法

*给定更合适的页面淘汰算法（不一定奏效）

*分配给个更多的物理内存页面（一般会有效果，而且效果较好，但给多了并发度就会降低）

*较好的方法是使用工作集机制处理。

①基本概念

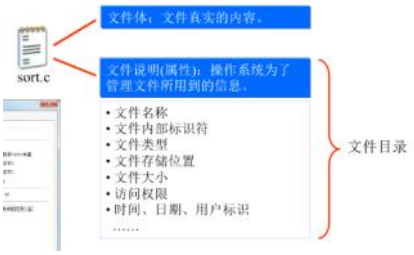
2023年5月6日　　16:05

*文件

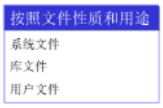
定义：是记录在外存上的，具有符号名的，在逻辑上具有完整意义的一组相关信息项的集合。

从用户的角度看，文件是逻辑外存的最小分配单元，即信息(数据)只能以文件的形式写入外存。

*组成部分



*类型



②文件的结构

2023年5月6日　　16:21

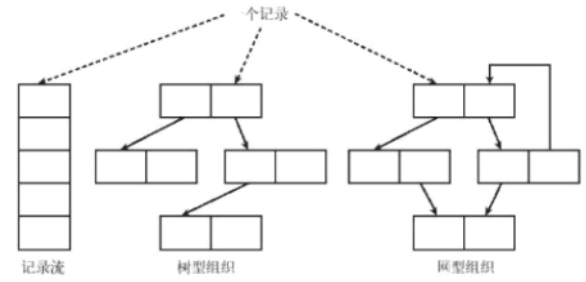
*逻辑

①逻辑结构

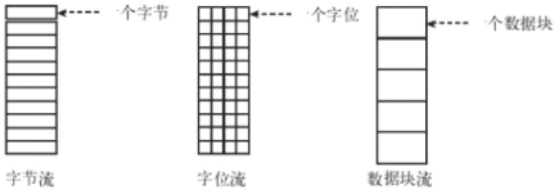
*定义：是用户所观察到的文件内容组织形式，它独立于物理存储设备。

*分类：

*有结构的文件（关系导向型结构）-记录式：



*无结构的文件（非关系导向型结构）-流式：

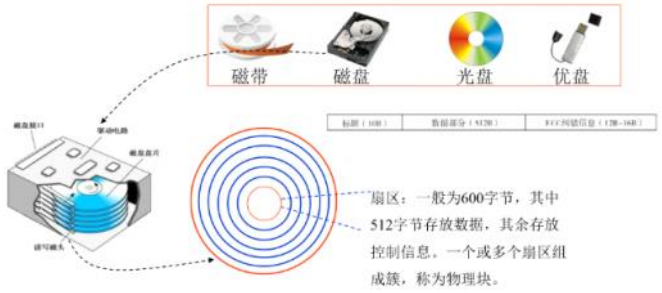


②物理结构

*定义：是指文件的内部组织形式，即文件在物理存储设备上的存放方法。

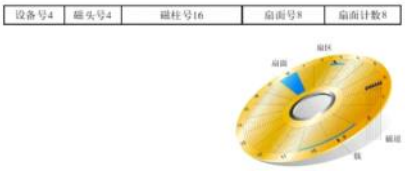
（Ⅰ）结构组成

*图解：



*操作系统管理磁盘数据的基本单位：

- 数据块 / 磁盘块 (block)
- 通常为扇区的整数倍（2的幂次方倍）
- 操作系统给出的访问地址：逻辑地址（LBA, Logical Block Address）

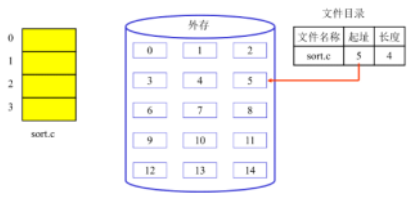


（II）操作系统从磁盘读取数据的过程

（III）几种常见的物理存储方式

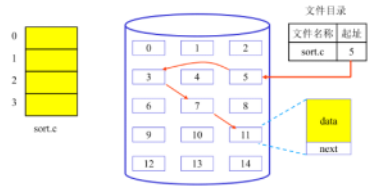
1.连续存储（顺序结构）

- 它将逻辑上连续的文件信息依次存放在编号连续的物理块上。



2.链接结构

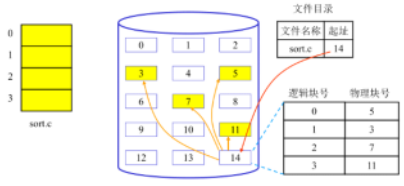
- 将逻辑上连续的文件信息存放在不连续的物理块上，每个物理块设有一个指针指向下一个物理块。



*适合随机访问且易于文件扩展

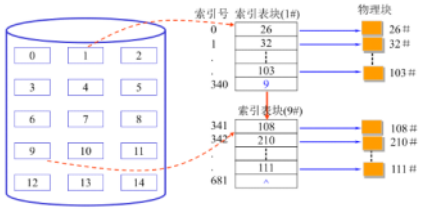
3.索引结构

- 将逻辑上连续的文件信息(记录)存放在不连续的物理块中，系统为每个文件建立一个专用数据结构——索引表，索引表中存放文件的逻辑块号和物理块号的对应关系。

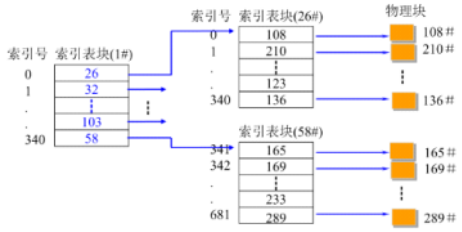


*索引表的组织方式

*链接文件方式：将多个索引表块按链接文件的方式串联起来。

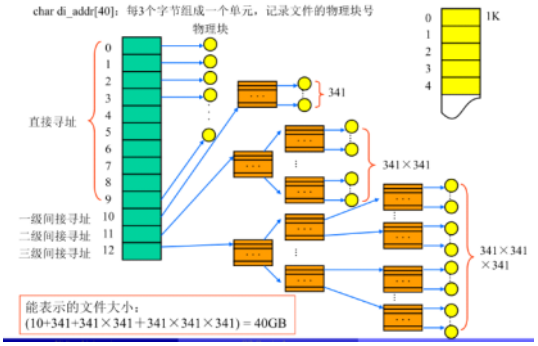


*多重索引方式：将一个大文件的所有索引表的地址放在另一个索引表中。



*非对称多重索引实例——UNIX文件系统（需要掌握！）

*类似于计组的多级寻址：（下图中1列连续的黄圆需要1个索引块）



*直接盘块：最终存放文件的盘块（即上图中的黄圈）

*间接盘块：存放直接盘块或者间接盘块的盘块（即上图中的橙块）

*n级物理盘块（n位1/2/3）：上图n级间接寻址的黄圈

**Hash文件

- 采用计算寻址结构，它由主文件和溢出文件组成。

③文件的存取方式

- 文件的存取方式是指读写文件存储器上的一个物理块的方法。
- 分类：
  - 顺序存取：指对文件中的信息按顺序依次读写的方式。
  - 随机读取
    - ① 直接存取法：允许用户随意存取文件中任意一个物理记录。
    - ② 按键存取法：根据文件中各记录的某个数据项内容来存取记录的，这种数据项称之为“键”。

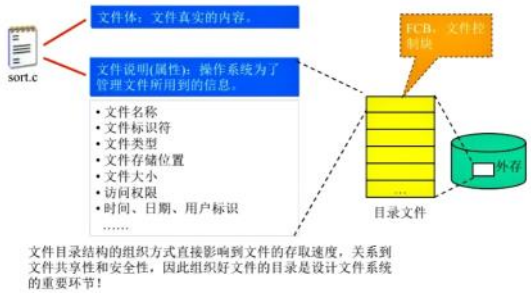
*文件结构、文件存取方式与文件存储介质的关系

存储设备	磁带	磁盘		
文件结构	连续	连续	串联	索引
存取方式	顺序	顺序、直接	顺序	顺序、直接

③文件的目录

2023年5月8日　　17:04

①文件目录的内容



②文件的目录结构

（I）一级目录结构

缺点：

- **重名问题**
- **只支持单用户**
- **文件共享问题**

（II）二级目录结构

（III）多级目录结构

*森林型/树形：可以解决重名问题，但并不能节省磁盘空间！

③文件的查找——线性检索法

*逐级查找~

④文件的共享

2023年5月8日　　17:59

①定义

文件共享是指不同用户或进程使用同一文件，实际上文件的实体只有一个

②文件共享类型

（I）硬链接

可使用ln命令实现

- # ln [options] oldfilename newfilename
- 例如：#ln /bin/ls /usr/dir

硬链接两大局限：

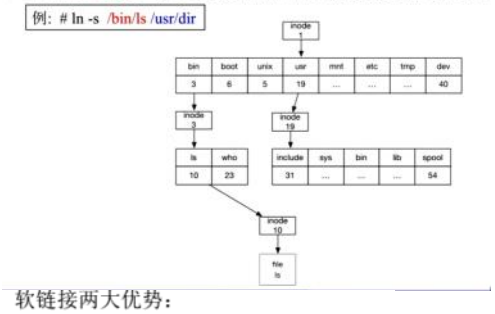
- 不能用于连接目录
- 也不能跨越文件系统的范围

*硬链接与目标文件共用一个索引节点，建立/删除会使引用计数增加/减少

（II）软链接

- 也称符号链接，符号链接文件的内容为被链接文件的路径名。

例: # ln -s /bin/ls /usr/dir



软链接两大优势：

- 可用于连接目录
- 可能跨越文件系统甚至网络建立链接

*软链接有自己单独的索引节点，不会影响目标文件的引用计数

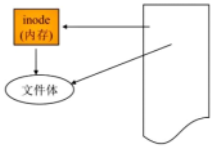
⑤文件的操作

2023年5月17日　　16:49

（1）打开文件机构

- 操作系统在内存设置了一个精炼的文件机构，用于实现灵活方便高效的文件操作，这套机构称为打开文件机构。
- 打开文件机构的组成：
  - 内存文件控制块（内存索引节点）
  - 系统打开文件控制块
  - 用户打开文件表

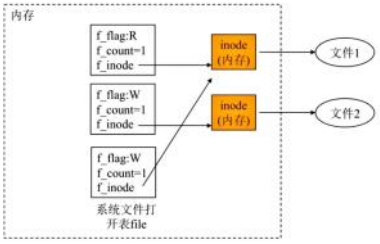
1.内存文件控制块



- 当打开某一个文件时，如果找不到其相应的内存inode，就在内存中将外存inode中的主要部分复制进去（适当变化），形成内存inode。
- 当需要查询、修改文件的控制信息时，直接在内存inode中进行。当关闭文件时，如果内存inode被修改过，则更新对应的外存inode信息。

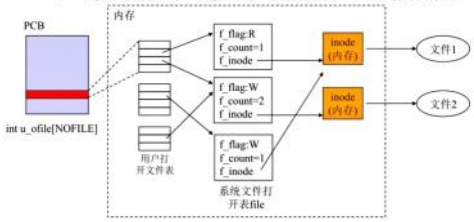
2.系统打开文件控制块

- 系统打开文件表用于记录所有打开文件的控制信息。



3.用户打开文件表

- 文件通常是通过进程来操作的，因此每一个文件都有一个用户(进程)。同样每一个进程可打开多个文件。
- 在操作系统中，每一个进程都有一张打开文件表——用户打开文件表。



*打开一个文件的过程：

(1) 系统要为其分配一个内存 I 节点，并将该文件的外存 I 节点中的主要部分复制进去，并填入外存 I 节点号。

(2) 分配一个空闲打开文件控制块 file，并使 f_inode 指针指向内存 I 节点。

(3) 在进程 user 结构的 u_ofile 中找到一个空闲项，填入 file 结构的地址，并将 u_ofile 数组索引值作为打开文件描述字返回给用户。

*示例：

3. 在UNIX系统中，若有如下三种情况：

(1) P1进程执行如下代码：

fd1=open("/bin/test",o_RDONLY);

fd2=open("/usr/bin/wangproc", o_RDWR);

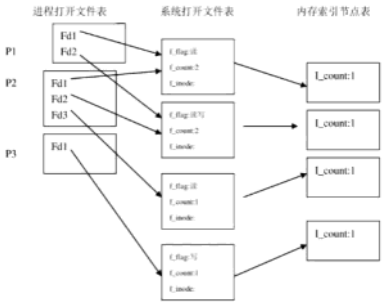
(2) P1进程创建的子进程 P2 执行如下代码：

fd3=open("/usr/bin/test.c",o_RDONLY);

(3) P3进程执行如下代码：

fd1=open("/etc/test",o_WRONLY);

请画出进程打开文件表u_ofile、系统打开文件表file和内存索引节点表i_node之间的关系图。


进程打开文件表
系统打开文件表
内存索引节点表
P1
P2
P3
Fd1
Fd2
Fd1
Fd2
Fd3
Fd1
I_count:1
I_count:1
I_count:1
I_count:1

⑥外存空间管理

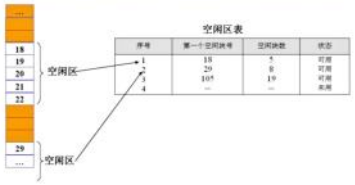
2023年5月17日　　16:49

0概述

- 外存具有较大的存储空间，且被多用户共享，用户执行程序经常要在磁盘上存储、删除文件，因此，文件系统必须对磁盘空间进行有效管理。
- 外存空闲空间管理的数据结构通常称为磁盘分配表(Disk Allocation Table)。常用的空闲空间的管理方法有：
  - 空闲区表
  - 位示图
  - 空闲块链

①空闲区表

- 将外存空间上一个连续未分配区域称为“空闲区”。操作系统为磁盘外存上所有空闲区建立一张空闲表，每个表项对应一个空闲区，空闲表中包含序号、空闲区的第一块号、空闲块的块数等信息。



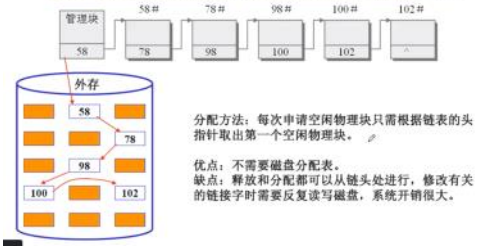
②位示图

- 这种方法是在外存上建立一张位示图(bitmap)，记录文件存储器的使用情况。每一位对应文件存储器上的一个物理块，取值0和1分别表示空闲和占用。

```
1001001101110001010111100001010
0011100100010101011101101000011
0011011011101010101101101011110
1101111100010010110001011010111
0001111101010101101011101011011
1000001111110111010100001101001O
```

③空闲块链

- 每个空闲物理块中有指向下一个空闲物理块的指针，所有空闲物理块构成一个链表，链表的头指针放在文件存储器的特定位置上（如管理块中）。

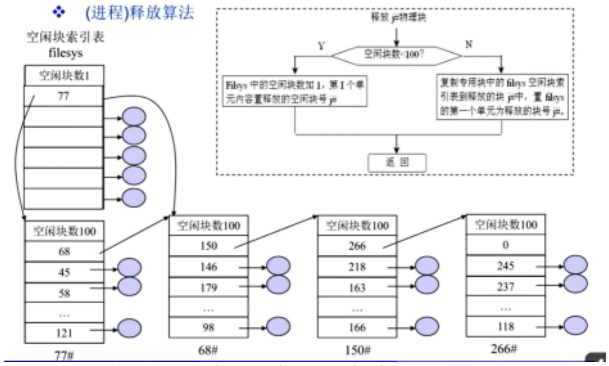


④成组链接法

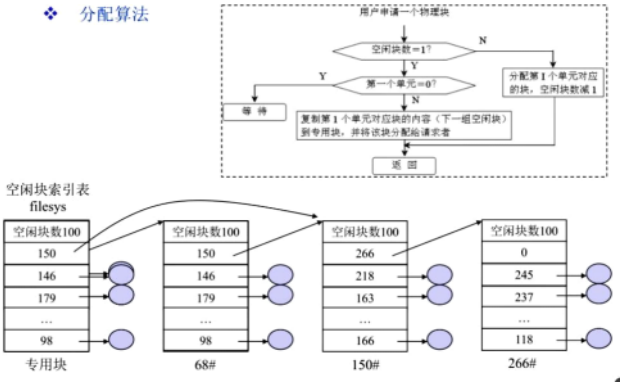
- 将空闲块分成若干组，每100个空闲块为一组。每组的第一个空闲块登记了下一组空闲块的物理盘块号和本组空闲块总数。

*最后一个节点（作为结束标记，功能类似于next指针）为最后一组中的第一个块号0（即相当于NULL），其虽被算进空闲块的计数里但实际上不属于空闲块，因此最后一组的空闲块数=第一个空闲块登记的空闲块总数-1！

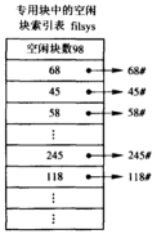
*进程释放算法（需能够说出完整过程！）



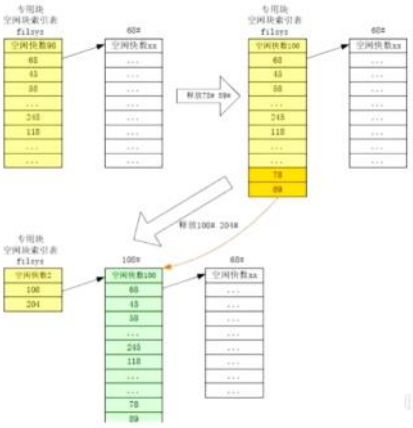
*进程分配算法（需能够说出完整过程！）



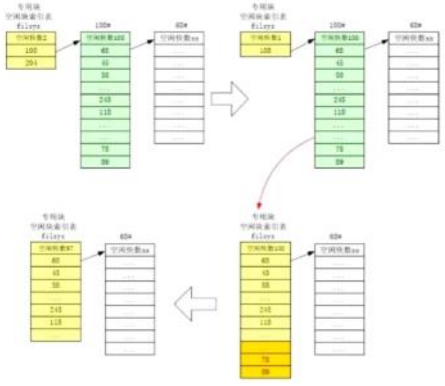
*规范示例：



*当用户释放了 78#，89#，108#和204#物理块，专用块中的空闲块索引表 filsys的变化情况又如何?



*当用户又申请5个物理块，专用块中的空闲块索引表 filsys 的变化情况又如何?

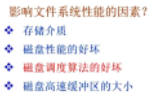


⑦文件系统的性能分析

2023年5月22日　　16:10

（0）基本概念

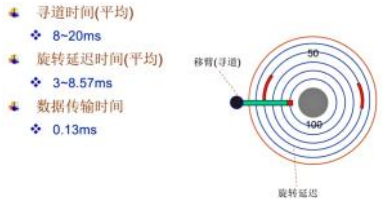
*影响因素



*扇区：一般为600字节，其中512字节存放数据，其余存放控制信息

*在操作系统角度，最小的存储单位是数据块或簇，簇是有一个或多个扇区组成的更大的单位。

（I）读写时间



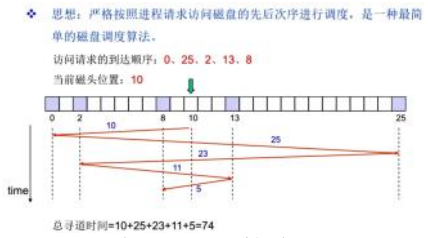
（II）移臂调度算法

*移臂顺序规范：

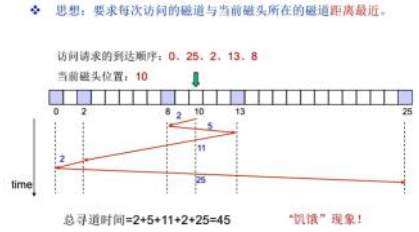
SSTF（最短查找时间）：

访问磁道号	移动距离
206	
205	1
225	20
246	21
278	32
286	8
296	10
332	36
398	66
414	16
491	77
168	323
94	74
总	684

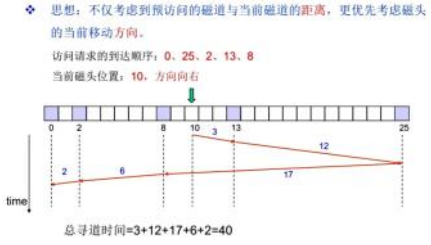
1.先来先服务算法FCFS



2.最短寻道时间优先算法SSF



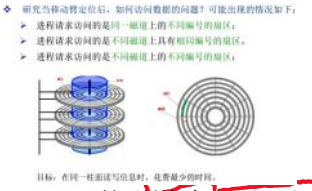
3.电梯调度算法SCAN



*三种算法比较

	FCFS	SSF	SCAN
平均花费时间	较大	较小	较小
稳定性	差	优	优
是否有“饥饿”现象	无	有	无

（III）旋转调度算法



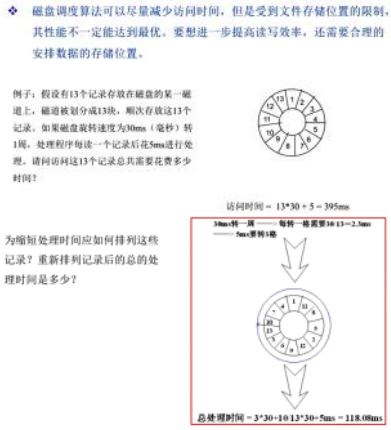
*虽然磁道确定了，但是由于存在多个数据块（即圆）所以依然有不同的磁道号（即此时的磁道号其实是磁头号）

*每个数据块都有一个磁头，但是任意时刻只能有一个磁头在工作

（IV）同时对移臂和旋转进行调度的优先次序

*柱面号（即磁道号）>扇区号>磁头/道号（实际上磁头号都不需要考虑）

（IV）信息的优化分布问题



①基本概念

2023年5月24日　　16:36

*负责计算机与外部的输入输出（I/O）工作的设备称为外部设备，简称外设（I/O设备）
*设备管理的目标

- 提高设备的利用率
  - 就是提高CPU与I/O设备之间的并行操作程度。
- 为用户提供方便、统一的界面
  - 方便：屏蔽外部设备的差异；
  - 统一：是指对不同的设备尽量使用统一的操作方式。

*提高设备的利用率可通过提升设备并行工作度达成

②外设的分类

2023年5月24日　　16:40

①按数据组织（存储、传输方式）

②按数据传输率分类

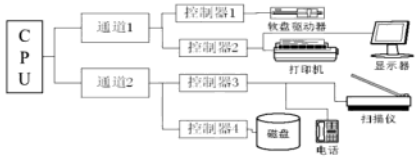
- 低速设备：指传输速率为每秒钟几个字节到数百个字节的设备。典型的设备有键盘、鼠标、语音的输入等；
- 中速设备：指传输速率在每秒钟数千个字节至数十千个字节的设备。典型的设备有行式打印机、激光打印机等；
- 高速设备：指传输速率在数百千个字节至数兆字节的设备。典型的设备有磁带设备、磁盘设备、光盘设备等。

③I/O系统&结构

2023年5月24日　　16:42

①系统定义
*计算机中负责管理I/O的机构（硬件和软件的组合）

②结构
*单、多总线结构
*通道系统



④I/O系统控制方式（逻辑I/O模式）

2023年5月24日　　16:44

①不同的控制方式

*程序控制I/O（直接控制方式/可编程I/O模式）

优点：简单。

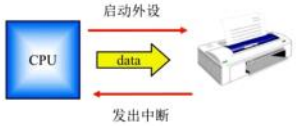
缺点：CPU的大部分时间都用于对硬件进行测试，效率低下。

```
copy_from_user(buffer,p,count)                    /*p为内核缓冲区*/
    For(i=0;i<count;i++){
        while(*printer_status_reg!=READY);
        *printer_data_register=p[i];              /*输出一个字符*/
    }
    Return_to_user();
```

打印一串字符

*中断驱动I/O

中断驱动I/O



优点：在外设进行数据处理时，CPU不必等待，可以继续执行该程序或其他程序。提高了CPU的利用率。中断技术使得CPU和外设之间的并行工作成为可能。

缺点：数据仍然需要通过CPU进行传输，由于CPU每次处理的数据量少，因此这种方式只适于数据传输率较低的设备。

打印代码

```
Copy_from_user(buffer,p,count)
Enable_interrupts();
while(*printer_status_reg!=READY);
*printer_data_register=p[0];
Scheduler();
```

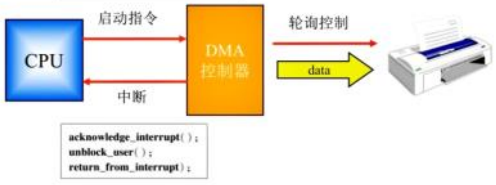
```
If(count==0){
    unblock_user();
}else{
    *printer_data_register=p[i];
    Count=count-1;
    i=i+1;
}
acknowledge interrupt();
return_from_intertupt();
```

中断处理代码

*直接存储访问I/O（DMA）

直接存储访问I/O(DMA, Direct Memory Access)

```
Copy_from_user(buffer,p,count);
Set_up_DMA_controller();
Scheduler();
```



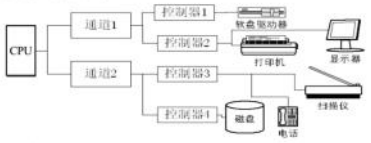

```
acknowledge_interrupt();
unblock_user();
return_from_interrupt();
```

优点：CPU只需干预I/O操作的开始和结束，而其中的数据读写无需CPU控制，适于高速设备。

*DMA工作过程：启动DMA控制器，完成数据交换，DMA发出中断信号，结束DMA控制。

*通道控制方式I/O

通道控制方式I/O



一个CPU可以连接若干个通道，一个通道可以连接若干个控制器，一个控制器可以连接若干个设备。

优点：解决了I/O操作的独立性和各部件工作的并行性。通道把中央处理机从繁琐的输入输出操作中解放出来。采用通道技术后，不仅能实现CPU和通道的并行操作，而且通道与通道之间也能实现并行操作，各通道上的外围设备也能实现并行操作，从而可达到提高整个系统的效率的根本目的。

*通道是专用的I/O系统，拥有自己的指令系统！（不同于指令集！）

*通道的种类：

*多路通道（可以同时通过多个）：字节、数组

*选择通道（任意时刻只能选一个）：数据

*对比图解：

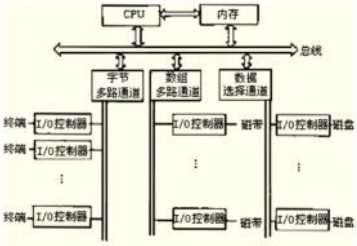


图 5-6 通道方式的数据传送结构

⑤I/O软件

2023年5月24日　　17:02

①组成部分

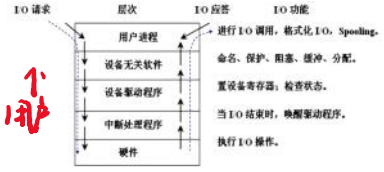
- I/O交通管制程序：
  - 负责各I/O设备之间的协调工作；
- I/O调度程序：
  - 负责设备的分配和调度；
- I/O设备处理程序：
  - 负责每类设备的具体操作。

②设计目标

- 设备独立
- 统一命名
- 错误处理
- 数据传输
- 缓冲管理
- 设备共享

……

③I/O软件的结构



（I）设备驱动程序层

- 设备驱动程序是直接同硬件(控制器)打交道的内核软件模块。它是整个操作系统中唯一知道(关注)具体硬件信息的模块。
- 一般而言，设备驱动程序的任务为：
  - 向上：接受来自与设备无关的上层软件的抽象请求；
  - 向下：进行与设备相关的操作。

*功能

- 从上层接收抽象的读写请求；
- 控制和监督各I/O控制器的正确执行，并确保读写操作完成；
- 初始化I/O设备，开/关设备；
- 缓冲区管理；

……

*工作过程

*特点

- 与I/O设备的硬件结构密切联系，是操作系统底层中唯一知道各种输入输出设备的控制器细节及其用途的部分。
- 驱动程序属于操作系统内核的组成部分，因此也是操作系统安全的巨大隐患。

（II）设备无关系统软件层

- 设备无关系统软件层
  - 负责实现对所有设备都具有共性的功能，并向上提供一个统一的接口。
  - 例如：缓冲管理，错误报告，分配与释放独享设备…
- 主要功能：
  - 统一界面：通过对下层(驱动程序层)规定一个标准化的功能清单，让所有的设备看上去都一样或者相似。
  - 缓冲管理：缓冲区是外设的常用组成部分，设置缓冲的目的是为了桥接不同速度的设备并提供灵活的安全机制。
- 错误处理：要负责处理两类错误——程序错误和I/O错误。
- 提供与设备无关的逻辑块：屏蔽底层各种I/O设备空间大小、处理速度和传输速率的差异，只向上层提供大小统一的逻辑块尺寸。
- 存储设备的块分配。
- 独占设备的分配与释放。

（III）用户空间I/O软件

- 就是具有I/O功能但在用户态下运行的软件(函数)或功能模块。
- 例如：

```
write(fd, buffer, nbytes);
open(…);
printf(…);
gets(…);
```

SPOOLing系统

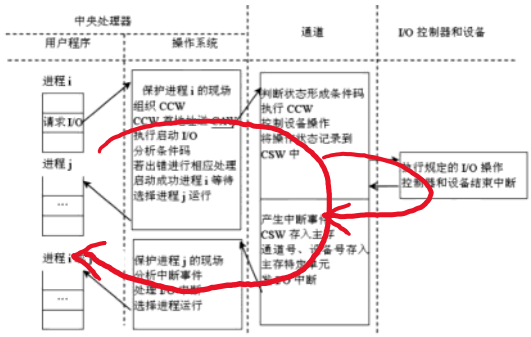
⑥具有通道的设备管理

2023年5月29日　　16:15

①基本概念

- 通道命令(Channel Command Word, CCW)：通道又称为I/O处理机，具有自己的指令系统，常常把I/O处理机的指令称通道命令。
- 通道程序：用通道命令编写的程序称通道程序，通道通过执行通道程序控制I/O设备运行。
- 通道地址字(Channel Address Word, CAW)：用来存放通道程序首地址的内存单元称通道地址字。
- 通道状态字(Channel Status Word, CSW)：是通道向操作系统报告工作情况的状态汇集。

②工作原理



*通道工作时，操作系统可以继续向后执行进程

*回到进程时，有一次**再调度**的过程

⑦缓冲技术

2023年5月29日　　16:27

①基本概念

- 缓冲区(buffer)是一种在设备和设备之间或设备与程序之间交换数据的区域。
- 设置缓冲区的目的：
  - 缓解缓冲两端设备(程序)速度的差异
  - 协调数据大小的不一致性
  - 实现应用程序与I/O的语义拷贝
- (几乎)所有的设备之间，都会使用缓冲区来交换数据。因此缓冲区的合理管理十分重要。

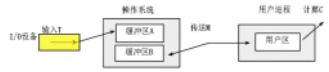
减少对CPU的中断频率，放宽对中断响应时间的限制
提高CPU和I/O设备之间的并行性

②分类

（I）单缓冲技术

*数据输入过程T可以与计算过程C并行，但是不能与传送过程M并行（缓冲区自身要求）！

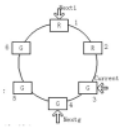
（II）双缓冲



*三种过程均可以并行

（III）环形缓冲

- 结构：



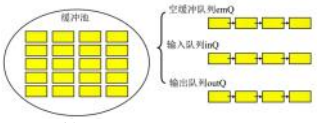
环形缓冲的两种现象

系统受限计算：Next追上Nextg

系统受限I/O：Nextg追上Next

（IV）缓冲池

- 可供多个进程共享的双向缓冲技术。

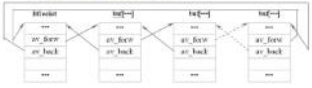


*缓冲管理队列

- 为方便对缓冲区进行管理，UNIX系统设置了三种队列。
  - 自由buf队列
  - 设备buf队列
  - NODEV设备队列
- 由于buf记录了与缓冲存储区有关的各种管理信息，所以缓冲区管理队列实际上就是对缓存控制块buf队列的管理。

*自由buf队列

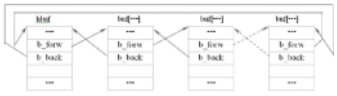
- 在Unix系统中，一个可被分配作它用的缓冲存储区，其相应的buf位于自由buf队列中。



自由buf队列采用FIFO管理算法。一个缓存被释放时，其相应的buf被送入自由buf队列的队尾；当要求分配一个缓存时，从队首取出一个buf，它所管理的缓存就可被“移作它用”。

*设备buf队列

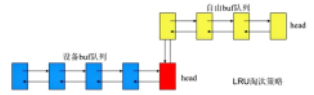
- 在Unix系统中，每类设备都有一个buf队列。



一个缓存被分配用于读、写某类块设备上的某一字符块时，其相应的buf就进入该类设备的buf队列，除非被“移作它用”，否则一直保留在该队列中。

*即就算已经被用完了仍会继续留在buf队列里

*缓冲控制块buf中设置两对指针的原因：



- 可以使得一个缓冲区同时处于两个队列中。
- 一个设备buf队列中的缓冲区当使用完后，便同时处于原设备buf队列中和自由buf队列中。
- 避免了重复、费时的I/O操作过程，从而提高了系统的I/O效率。

(队列太长时也需淘汰)

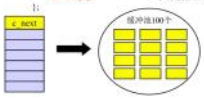
⑧字符设备的缓存管理

*用缓冲池管理

（I）组成部分

❖ 字符缓存用于解决CPU与字符设备间速度不匹配的矛盾，由于字符缓存很短，所以UNIX在实现上没有设置专门的缓存控制块，其字符缓存的结构如下：

```
struct cblock
{
    struct cblock *c_next;      /*字符缓存指针*/
    char info[6];               /*字符缓存信息区*/
};
```



*自由字符缓存队列

➢ 由空闲的字符缓存构成自由队列。

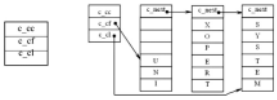
字符缓存的分配和释放都是在队首进行。

*注意操作均在队首（与②中缓冲池相区分）

*I/O字符缓存队列

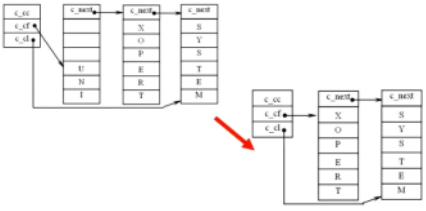
➢ 字符设备通过字符缓存进行输入或输出。各个正被使用的字符缓存按照它们的不同用途形成多个I/O队列，每个队列设置一个控制块，其结构如下：

```
struct list
{
    int c_cc;       /*字符计数器*/
    int c_cf;       /*缓存队列首指针*/
    int c_cl;       /*缓存队列尾指针*/
};
```



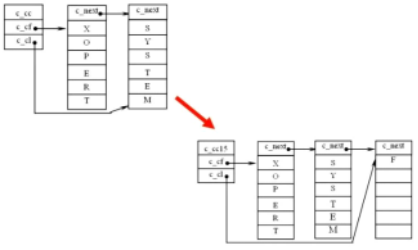
（II）具体方式

1.取字符&释放字符缓存



2.送字符&分配字符缓存

*即1中过程反过来：



⑧设备管理中的设备分配

2023年6月10日　　10:18

①操作系统中的设备分配

- 在多道程序环境下，系统中的设备供所有进程共享。
- 为防止诸进程对系统资源的无序竞争，特规定系统设备不允许用户自行使用，必须由系统统一分配。每当进程向系统提出I/O请求时，只要是可能和安全的，设备分配程序便按照一定的策略，把设备分配给请求用户（进程）。

②设备管理数据结构

（0）总体数据结构架构图

（I）设备控制表DCT

每个设备一张，描述设备的特性和状态，设备与控制器的连接情况等

（II）控制器控制表COCT

每个设备控制器一张，描述设备控制器的配置和状态

（III）通道控制表CHCT

每个通道一张，描述通道工作状态

（IV）系统设备表SDT

系统范围的数据结构，其中记录了系统中全部设备的情况，每个设备占一个表目