第一章 操作系统概述

2024年6月22日 10:54

- [x] **什么是操作系统？它有什么基本特征？有什么重要作用？**
  - 操作系统是位于硬件层之上、所有其他系统软件层之下的一个系统软件，是管理系统中各种软硬件资源，使用户能方便使用计算机系统的程序集合
  - 并发性：所谓程序并发，是指在计算机系统中同时存在多个程序。从宏观上看来，这些程序是同时向前推进的
  - 共享性：所谓资源共享是指操作系统与多个用户程序共用系统中的各种资源，这种共享是在操作系统的控制下实现的。
  - 虚拟性：所谓虚拟（virtual），是利用某种技术把一个物理实体变为若干个逻辑实体。物理实体是实际存在的，而逻辑实体则是虚化的。
  - 异步性：在操作系统之上，宏观上同时运行的程序有多个，这些程序（连同操作系统程序）是交替执行的。
  - 作用：处理机管理，存储器管理，I/O设备管理，文件系统和用户接口

- [x] **操作系统有哪些类型？各自有什么特点？**
  - 多道批处理操作系统：
    - 是以脱机操作为标志的操作系统， 特别适合于处理运行时间比较长的程序。
    - 在使用这种系统时，用户无法对其程序的运行状况施行交互性控制
    - 多道批处理操作系统具有两个特性： ①多道。内存中同时存在多个正在处理的作业，而且外存储器输入井中还有多个尚待处理的作业。 ②成批。作业逐批地进入系统，逐批地处理，逐批地离开系统。作业与作业之间的过渡由操作系统控制，无须用户干预。
  - 分时操作系统：
    - 是以联机操作为标志的操作系统，特别适合于程序的动态调试和修改。
    - 在一个分时系统中，一个主机同多个交互终端相连，这些终端既可能是本地的，也可能是远程的。每个终端上可以有一个用户，系统以对话的方式与终端用户交互
    - 分时操作系统为终端用户提供一组交互终端命令，它是用户与操作系统之间交互的界面。
    - 这类系统是采取分时的方法为多个终端用户提供服务的，它将时间划分为若干个片段，称为时间片，并以时间片为单位轮流地为各个交互终端用户服务。
    - 分时操作系统具有以下3个重要的特性。 ①多路性。又称多路调制性，即一个主机可以同时与多个终端相连。 ②交互性。又称交往性，即系统以对话的方式为各个终端用户服务。③独占性。由于计算机的运行速度很快，相比之下手动操作的速度较慢，因而每个用户感觉仿佛独占整个计算机系统，而不知道其他用户的存在，即每个终端用

户实际上都拥有一台完全属于自己的虚拟机。

- 实时操作系统：
  - 实时，是指系统能够对外部请求做出及时的响应。实时操作系统按其应用范围可以分为实时控制和实时信息处理两大类别。
  - 实时操作系统应当具有两个基本特性。 ①及时性。即能够对外部请求做出及时的响应和处理。 ②可靠性。
- 通用操作系统：
  - 同时具有分时、实时和批处理功能的操作系统称为通用操作系统
  - 在通用操作系统中，可能同时存在3类任务，即实时任务、分时任务、批处理任务。这 3 类任务通常按照其急迫程度加以分级：实时任务级别最高，分时任务级别次之，批处理任务级别最低。
- 单用户操作系统：
  - 单用户操作系统是为个人计算机所配置的操作系统，这类操作系统最主要的特点是单用户，即系统在同一段时间内仅为一个用户提供服务。
- 网络操作系统：
  - 用于实现网络通信和网络资源管理的操作系统称为网络操作系统
- 分布式操作系统：
  - 分布式操作系统分为两类：一类建立在多机系统的基础之上，称为紧耦合分布式系统；另一类建立在计算机网络的基础之上，称为松散耦合分布式系统。
  - 分布式操作系统是网络操作系统的更高级形式，它保持网络操作系统所拥有的全部功能， 同时具备如下特征：①统一的操作系统。 ②资源的进一步共享。 ③可靠性。 ④透明性。
- 多处理器操作系统：
  - 具有公共内存和公共时钟的多CPU系统称为多处理器操作系统，也称为紧耦合系统
  - 多CPU系统中的多个CPU若型号和地位相同，没有主从关系，则称之为对称多处理（SMP）。对称多处理是多处理器操作系统的主要形式。
- 集群操作系统
- 云计算操作系统
- 嵌入式操作系统：
  - 提供一个支持多道程序设计的环境的操作系统称为嵌入式操作系统
  - 嵌入式操作系统与一般操作系统相比在以下几方面具有比较明显的差别。
    - 可裁减性
    - 可移植性
    - 可扩展性
- 多媒体操作系统
- 智能卡操作系统

• 为构建一个高效、可靠的操作系统，硬件需要提供哪些支持？

- 定时装置
- 堆与栈
- 寄存器
- 特权指令与非特权指令
- 处理器状态及状态转换
- 地址映射机构
- 存储保护设施
- 中断装置
- 通道与DMA控制器

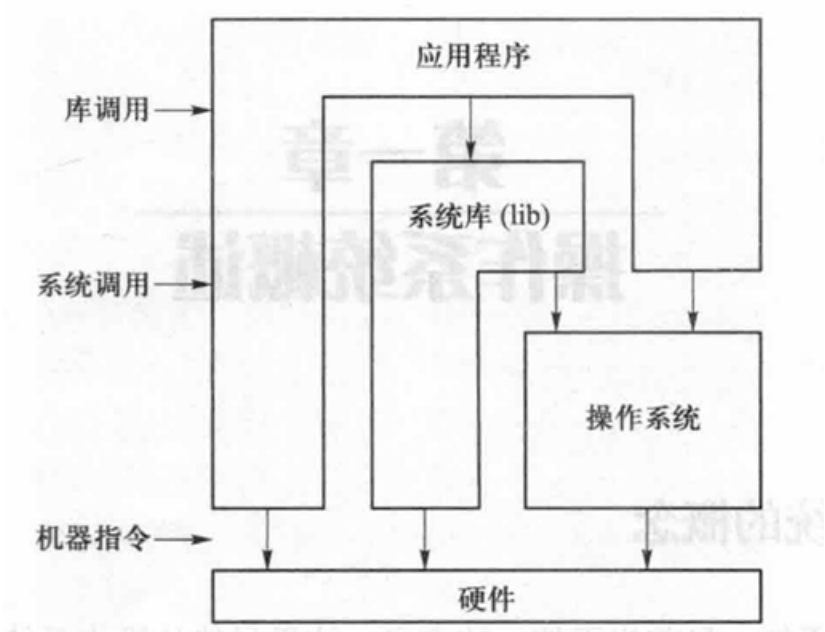
描述操作系统的处理器状态是如何转换的

- 目态到管态的转换：由于修改处理器状态字指令属于特权指令，只能在管态下执行，因而目态程序无法直接控制处理器状态的转换。处理器状态由目态转换为管态的唯一途径是中断。中断发生时，中断向量中的处理器状态字应标识处于管态，这个标识一般是由操作系统初始化程序来设置的。
- 管态到目态的转换：管态到目态的转换可以通过修改程序状态字（置PSW）来实现。由于操作系统运行于管态， 用户程序运行于目态，因而这种状态转换伴随着由操作系统到用户程序的转换。

描述中断装置的功能

- 发现中断：中断发生时能够识别。有多个中断事件同时发生时，按优先级别响应最高者。
- 响应中断：将目前运行进程的中断向量PSW和PC压入系统栈，然后根据中断原因到指定的内存单元将新的中断向量取出并送到寄存器中，从而控制转到相应的中断处理程序。

画图解释应用程序、系统库、操作系统、硬件之间的调用关系



- 应用程序以目标代码的形式运行时，可以与操作系统和硬件直接打交道（调用操作系统或执行硬件指令），操作系统之上的系统库可以被应用程序调用，系统库中的函数又可以调用操作系统

- **分时系统和实时系统有什么不同?**

▶ 分时系统通用性强，交互性强，及时响应性要求一般（通常数量级为秒）；

▶ 实时系统往往是专用的，系统与应用很难分离，常常紧密结合在一起，实时系统并不强调资源利用率，而更关心及时响应性（通常数量级为毫秒或微秒）、可靠性等

▶ **硬件将处理器划分为两种,即管态和目态,这样做会给操作系统的设计带来什么好处**?

▶ 便于设计安全可靠的操作系统。管态和目态是计算机硬件为保护操作系统免受用户程序的干扰和破坏而设置的两种状态。通常操作系统在管态下运行,可以执行所有机器指令;而用户程序在目态下运行,只能执行非特权指令。如果用户程序企图在目态下执行特权指令,将会引起保护性中断,由操作系统终止该程序的执行,从而保护了操作系统。

- **何谓特权指令?如果允许用户进程执行特权指令,会带来什么后果?举例说明。**

▶ 在现代计算机中,一般都提供一些专门供操作系统使用的特殊指令,这些指令只能在管态执行,称为**特权指令**。这些指令包括停机指令,置PSW指令,中断操作指令(开中断、关中断、屏蔽中断),输入输出指令等。

▶ 用户程序不能执行这些特权指令。如果允许用户程序执行特权指令,就有可能干扰操作系统的正常运行,甚至有可能使整个系统崩溃。

- **中断向量在计算机中的存储位置是由硬件决定的还是软件决定的?**

▶ 中断向量在计算机中的存储位置是由硬件决定的

- **中断向量的内容是由操作系统程序决定的,还是由用户程序决定的?**

▶ 由操作系统程序决定的。向量的内容包括中断处理程序的入口地址和程序状态字(中断处理程序运行环境),中断处理程序是由操作系统装入内存的,操作系统将根据装入的实际地址和该中断处理程序的运行环境来填写中断向量。

- **中断向量内的处理器状态字应当标明管态还是目态?为什么?**

- 答:应当标明是管态。该状态由系统初始化程序设置,这样才能保证中断发生后进入操作系统规定的中断处理程序。

• 系统如何由目态转换为管态?如何由管态转换为目态?

- 目态程序被中断时,现行PSW被压入系统栈,中断向量PSW被送入寄存器。由于后者状态位为管态,系统状态由目态转换为管态。
- 当处于管态的中断处理程序执行完且没有嵌套中断时,将系统栈中的PSW弹出送入寄存器。由于后者状态位为目态,即实现了由管态到目态的转换。

• 中断与程序并发之间的关系是什么?

- 中断是程序并发的前提条件。如果没有中断,操作系统不能获得系统控制权,也就无法按照调度算法对处理器进行重新分配,一个程序将一直运行到结束而不会被打断。

• 根据用途说明“栈”和“堆”的差别。

- 栈是一块按后进先出(Last In First Out,LIFO)规则访问的存储区域,用来实现中断嵌套和子程序嵌套(保存调用参数和返回断点)。堆虽然是一块存储区域,但是对堆的访问是任意的,没有后进先出的要求,堆主要用来为动态变量分配存储空间。

• 何谓系统栈?何谓用户栈?系统栈有何用途?用户栈有何用途?

- 系统栈是内存中属于操作系统空间的一块固定区域,其主要用途为:
  - (1)保存中断现场,对于嵌套中断,被中断程序的现场信息被依次压入系统栈,中断返回时逆序弹出;
  - (2)保存操作系统子程序之间相互调用的参数、返回值、返回点,以及子程序(函数)的局部变量。
- 用户栈是用户进程空间中的一块区域,用于保存用户进程的子程序之间相互调用的参数、返回值、返回点,以及子程序(函数)的局部变量。

• 为何无法确定用户堆栈段的长度?

- 用户堆栈段的长度主要取决于以下因素:
  - (1)用户进程(线程)中子程序(函数)之间的嵌套调用深度;
  - (2)子程序参数和局部变量的数量及类型;
  - (3)动态变量的使用。
- 这些在进程(线程)运行前无法确定,由此导致用户堆栈段的长度无法预先准确确定。

- **为何堆栈段的动态扩充可能导致进程空间的变迁?**

► 答:堆栈段的扩充需要在原来进程空间大小的基础上增添新的存储区域,而且通常要求新的存储区域是原来存储区域的连续区域。如果原来存放位置处的可扩展区域已经被其他进程占用,则需要将整个进程空间搬迁到另外一个区域,以实现地址空间扩展要求。

- **试述批处理操作系统与分时操作系统的差别。**

► 批处理系统采用脱机操作模式，追求系统效率,界面是作业控制语言(JCL);分时系统采用联机操作模式,追求交互性,界面是交互终端命令或GUI。

- **何谓作业?它包括哪几个部分?各个部分的用途是什么?**

► 所谓作业是指用户要求计算机系统为其完成的计算任务的集合。一个作业通常包括程序、程序所处理的数据以及作业说明书,程序用来完成特定的功能,数据是程序处理的对象,作业说明书用来说明作业处理的步骤和控制意图。

- **为什么构成分布式系统的主机一般都是相同的或兼容的?**

► 这样更有利于进程的动态迁移。如果主机不兼容,则在一台主机上能够运行的进程,因所用指令系统不同在另一台主机上可能无法运行,导致进程难以在不同主机之间迁移,使得分布式操作系统难以实现负载平衡。

- **为什么嵌入式操作系统通常采用微内核结构?微内核结构包括哪些内容?**

► 答:嵌入式操作系统与一般操作系统相比具有比较明显的差别:
  ► (1)嵌入式操作系统规模一般较小。因为其硬件配置一般较低,而且对操作系统提供的功能要求也不高;
  ► (2)应用领域差别大。对于不同应用,其硬件环境和设备配置情况有着明显的差别。
► 所以,嵌入式操作系统一般采用微内核(micro kernel)结构。
► 微内核包括如下基本成分:
  ► (1)处理器调度;(2)基本内存管理;(3)通信机制;(4)电源管理。
► 在这些基本成分之上可进行扩展,以适应不同的应用目标。

- **微内核结构有哪些优点和缺点?**

► 微内核结构的明显优点是可靠性高,可移植性好,适用范围广。
► 缺点是效率低,因为许多系统功能(如文件系统)被放置在核外,调用这些功能会导致两次上

下文切换。

- **程序状态字包含哪些主要内容?**
  - (1)程序基本状态
  - (2)中断码
  - (3)中断屏蔽位

- **什么是系统调用，为什么要使用“系统调用”机制**
  - 系统调用是指用户在程序中调用操作系统所提供的一些子功能，是操作系统提供给应用程序使用的接口，系统调用可视为特殊的公共子程序。
  - 用户程序不能直接执行对系统影响非常大的操作，必须通过系统调用的方式请求操作系统代为执行，以便保证系统的稳定性和安全性，防止用户程序随意更改或访问重要的系统资源，影响其他进程的运行。

- **导致一个进程创建另一个进程的典型操作有哪几种?**
  - 用户登录：系统为用户创建一个进程，并插入就绪队列
  - 作业调度
  - 提供服务：系统为用户请求创建一个进程
  - 应用请求：用户程序自己创建进程

输入井和输出井分别为磁盘或磁鼓上的两个区域，输入井用于保存已经输入但尚未处理的 作业；输出井用于保存处理完毕但尚未输出的结果。设置输入井和输出井的目的主要有两个： 协调输入输出设备速度与处理器速度之间的差异；为作业调度提供有利条件，如果没有输入井， 系统只能按照自然次序处理作业，设置输入井后，系统可以根据调度的需要在输入井中选择进 入内存的作业，使得内存中运行的作业搭配合理。

第二章 进程、线程与作业

2024年6月22日 11:57

- **单道程序设计有哪些缺点?**
  - 设备资源利用率低
  - 内存资源利用率低
  - 处理器资源利用率低

- **分析多道程序设计对于系统的资源利用率所带来的影响**
  - 设备资源利用率提高：如果允许若干个搭配合理的程序同时进入内存，这些程序分别使用不同的设备资源，则系统中的各种设备资源都会被用到并经常处于忙碌状态
  - 内存资源利用率提高：允许多道程序同时进入系统可以避免单道程序过短而内存空间过大所造成的存储空间的浪费，从而提高内存资源的利用率。同时，进入系统的多个程序可以保存在内存的不同区域中。
  - 处理器资源利用率提高：如果将两道程序同时放入内存，在一个程序等待输入输出操作完成期间，处理器执行另一个程序，这样便可提高处理器的利用率

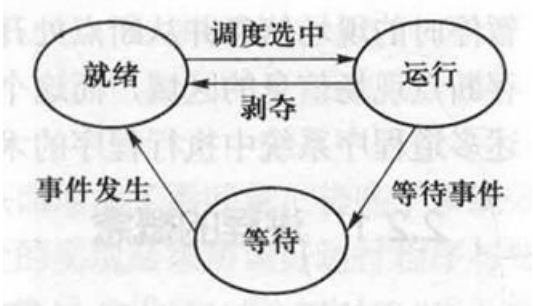
- **进程在其生存周期内可能处于哪些基本状态？三者之间如何转换？请画图说明**
  - ①运行（run）态：进程占有处理器资源，正在运行。
  - ②就绪（ready）态：进程本身具备运行条件，但是由于处理器的数量少于可运行进程的数量，暂未投入运行，即相当于等待处理器资源。
  - ③等待（wait）态：也称为挂起（suspend）态、阻塞（block）态、睡眠（sleep）态。进程本身不具备运行条件，即使分给其处理器也不能运行。进程正在等待某一事件的发生，如等待某一资源被释放，等待与该进程相关的数据传输的完成信号等。
  - 

图 2-3 进程间的基本状态转换关系
  - 具体地说，当一个就绪进程获得处理器时， 其状态由就绪态变为运行态；
  - 当一个运行进程被剥夺处理器资源时，如用完系统分给它的时间片，或者出现高优先级别的其他进程，其状态由运行态变为就绪态；

- ► 当一个运行进程因某事件受阻时，如所申请资源被占用、启动数据传输未完成，其状事件发生态由运行态变为等待态；
- ► 当所等待的事件发生时，如得到被申请资源、数据传输完成，其状态由等待态变为就绪态。

• 进程由哪几个部分构成？他们分别属于什么空间？

- ► 进程由两个部分组成，即进程控制块和程序，其中程序包括代码和数据等。
- ► 进程控制块属于操作系统空间，而程序则属于用户空间。

• 进程队列有哪几种类型？进程在不同队列中如何变化？请画图说明

- ► 就绪队列、等待队列、运行队列
- ►

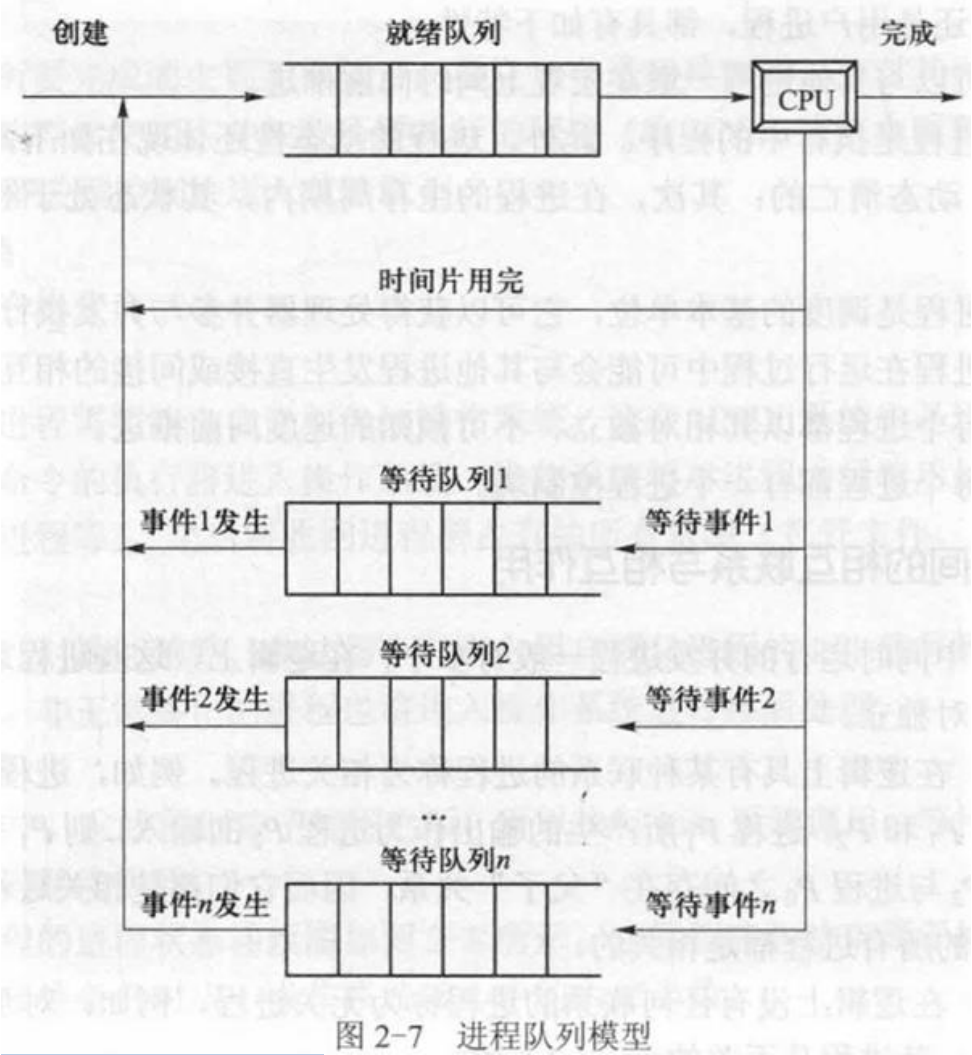


图 2-7　进程队列模型

- ► 进程初创后进入就绪队列；CPU空闲下来时从就绪队列中选取进程使其运行，获得处理器的进程用完所分得的时间片后回到就绪队列；运行进程因某一事件受阻进入相应的等待队列，等待进程在所等事件发生后进入就绪队列，运行进程正常结束后离开系统。

• 从操作系统的角度看，进程有哪些类型？进程有哪些特性？分别进行解释

- 从操作系统的角度来看，可以将进程分为系统进程和用户进程两大类。
- ①并发性。可以与其他进程一道在宏观上同时向前推进。
- ②动态性。进程是执行中的程序。此外，进程的动态性还体现在如下两个方面。首先， 进程是动态产生、动态消亡的；其次，在进程的生存周期内，其状态处于经常性的动态变化之中。
- ③独立性。进程是调度的基本单位，它可以获得处理器并参与并发执行。
- ④交互性。进程在运行过程中可能会与其他进程发生直接或间接的相互作用。
- ⑤异步性。每个进程都以其相对独立、不可预知的速度向前推进。
- ⑥结构性。每个进程都有一个进程控制块。

• 进程间相互作用的方式有哪些？请举例解释他们之间的差异

- ①直接相互作用：进程之间不需要通过媒介而发生的相互作用，这种相互作用通常是有意识的。例如，进程P1将一个消息发送给进程P2，进程P1的某一步骤S1，需要在进程P2的某一步骤又执行完毕之后才能继续等。直接相互作用只发生在相关进程之间。
- ②间接相互作用：进程之间需要通过某种媒介而发生的相互作用，这种相互作用通常是无意识的。例如，进程P1欲使用打印机，该设备当前被另一进程P2所占用，此时进程P1只好等待，以后进程P2用完并释放该设备时，将进程P1唤醒。间接相互作用可能发生在任意进程之间。

• 画出体现生灭过程的进程状态转换图

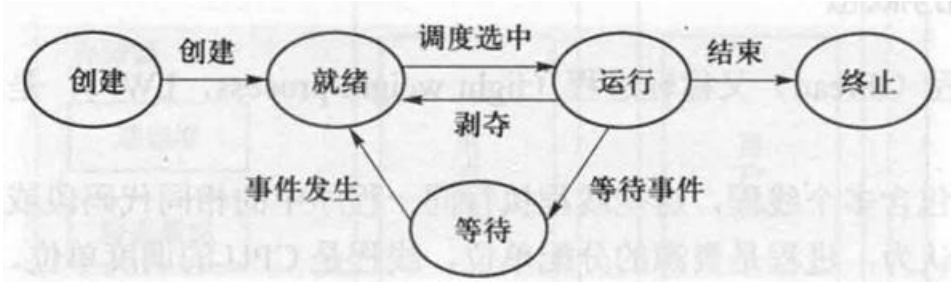


图 2-8　体现生灭过程的进程状态转换图

• 描述进程与程序的联系和差别

- 进程与程序的联系：程序是构成进程的组成部分之一，一个进程存在的目的就是执行其所对应的程序。如果没有程序，进程就失去了其存在的意义。
- 进程与程序的差别：
  - ①程序是静态的，而进程则是动态的。
  - ②程序可以写在纸上或在某种存储介质上长期保存，而进程具有生存周期，创建后存在， 撤销后消亡。
  - ③一个程序可以对应多个进程，但是一个进程只能对应一个程序。

- **什么是进程？什么是线程？它们的关系是什么？**
  - **进程**：进程是具有一定独立功能的程序关于一个数据集合的一次运行活动。
  - **线程**：线程又称轻进程（LWP），是进程内的一个相对独立的执行流。
  - 一个进程可以包含多个线程，这些线程执行同一程序中的相同代码段或不同代码段，共享数据区和堆。一般认为，进程是资源的分配单位，线程是CPU的调度单位。一个线程只能属于一个进程，而一个进程可以有多个线程；资源分配给进程，同一进程的所有线程共享该进程的所有资源；处理机分给线程，即真正在处理机上运行的是线程；线程在运行过程中，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。

- **与进程相比，线程有哪些优点？**
  - 与进程相比，线程具有如下优点：
    - ①上下文切换速度快。由同一进程中的一个线程切换到另一个线程只需改变寄存器和栈， 包括程序和数据在内的地址空间不变。
    - ②系统开销小。创建线程比创建进程所需完成的工作少，因而对于客户请求，服务器动 态创建线程比动态创建进程具有更高的响应速度。
    - ③通信容易。由于同一进程中的多个线程的地址空间共享，一个线程写到数据空间的信 息可以直接被该进程中的另一线程读取，方便快捷。

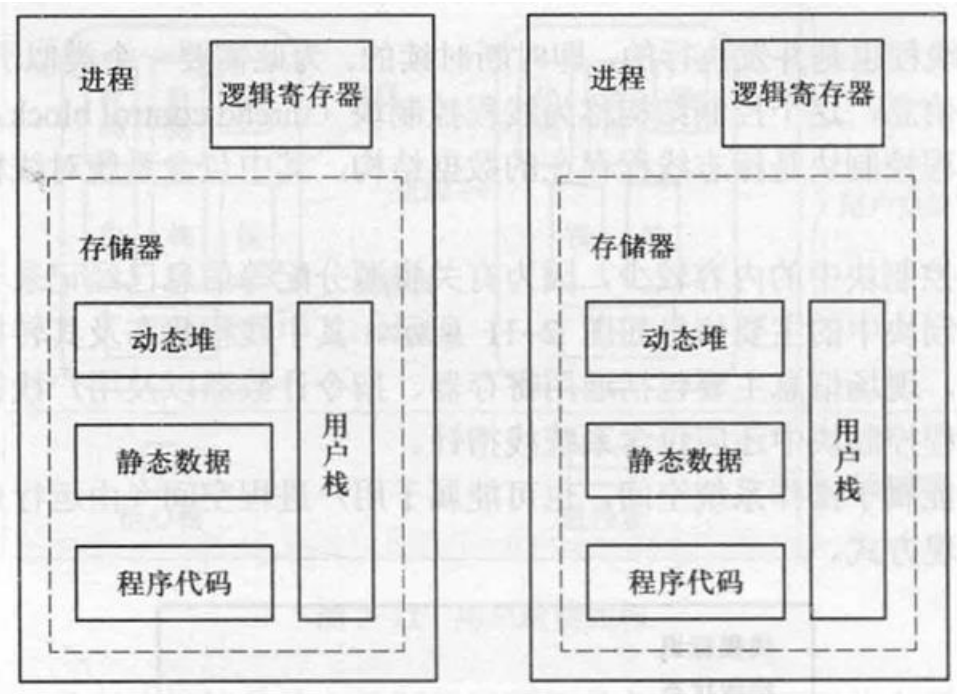
- **画图呈现多进程结构与多线程结构，说明什么时候能用多线程结构来实现多进程**
  - 

    图 2-9　多进程结构

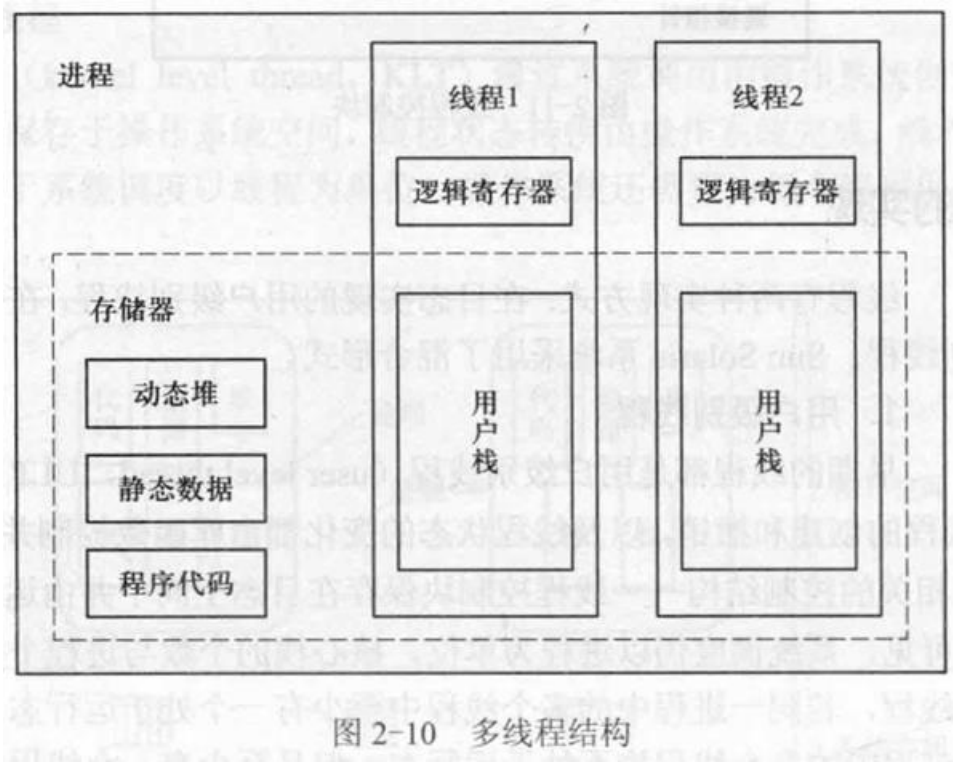
- 

图 2-10　多线程结构

- 如果这两个进程具有一定的逻辑联系，比如二者是执行相同代码的服务程序，或者二者为协同进程，则可以用多线程结构实现

- **线程有哪些实现方式？各自有什么优缺点？**
  - 线程有两种实现方式：在目态实现的用户级别线程，在管态实现的核心级别线程。
  - 用户级别线程的优点在于：线程不依赖于操作系统，可以采用与问题相关的调度策略，灵活性好；同一进程中的线程切换不需要进入操作系统，因而实现效率较高。
  - 用户级别线程的缺点在于：同一进程中的多个线程不能真正并行，即使在多处理器环境中；由于线程对操作系统不可见，调度在进程级别，某进程中的一个线程通过系统调用进入操作系统受阻，该进程的其他线程也不能运行。
  - 核心级别线程的优点是并发性好，在多处理器环境中同一进程中的多个线程可以真正并行执行。
  - 核心级别线程的缺点是线程的控制和状态转换需要进入操作系统完成，系统开销比较大。

- **为何引入多道程序设计?在多道程序系统中,内存中作业的道数是否越多越好?请说明原因。**
  - 引入多道程序设计技术是为了提高计算机系统资源的利用率。
  - 在多道程序系统中,内存中作业的道数并非越多越好。一个计算机系统中的内存、外设等资源是有限的,只能容纳适当数量的作业,作业道数增加将导致对资源的竞争激烈,进程切换频繁,系统开销增大,从而导致作业的执行缓慢,系统效率下降。

- **多道程序设计带来哪些问题?如何解决?**

► (1)处理器资源管理:将CPU资源按照调度原则分派给可以运行的程序;

► (2)内存资源管理:进程空间既相互独立,又可共享;

► (3)设备资源管理:多个进程使用同一个设备时不发生冲突。

- **什么是多道程序设计技术？如何在一个CPU的情况下实现该技术?**

► 多道程序设计就是将多个用户程序同时装入内存，然后在操作系统的控制下，多个程序交替或同时运行。

► 在一个CPU的情况下，可让多个程序轮流使用CPU和I/O设备，从而形成一个程序使用CPU时，其他的程序在进行I/O操作，以达到多个程序同时运行并提高CPU和外设的使用率的效果。

- **有人说,用户进程所执行的程序一定是用户自己编写的。这种说法对吗?如果不对,试举例说明之。**

► 这种说法不对。例如,C语言编译程序以用户进程身份运行,但C语言编译程序一般并不是用户自己编写的。此外,还有调试程序、字处理程序等工具软件。

- **什么是进程上下文?进程上下文包括哪些成分?哪些成分对目态程序是可见的?**

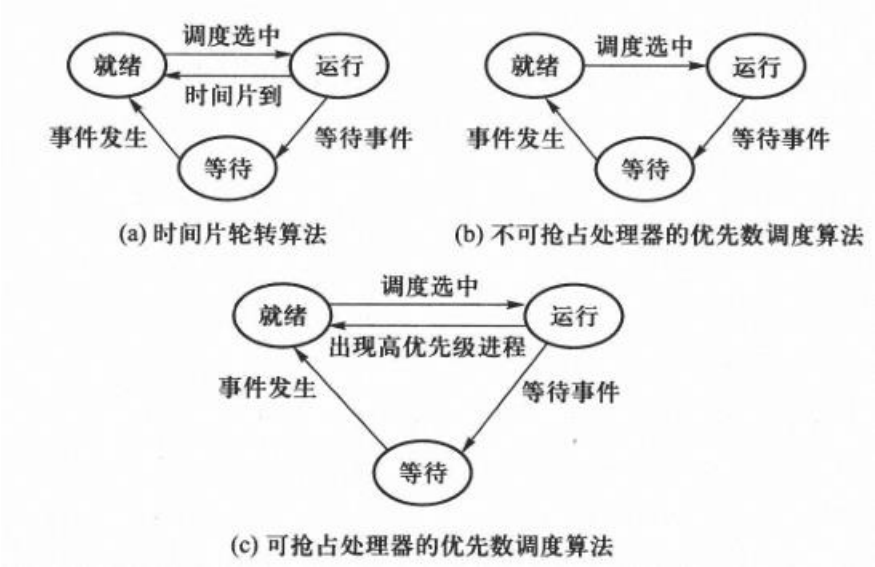
► 进程是在操作系统支持下运行的,进程运行时操作系统需要为其设置相应的运行环境,如系统堆栈、地址映射寄存器、打开文件表、PSW与PC、通用寄存器等。进程的物理实体与支持进程运行的物理环境合称为进程上下文。

► 进程上下文包括以下三个组成部分:

  - (1)用户级上下文。是由用户进程的程序块、用户数据块(含共享数据块)和用户堆栈组成的进程地址空间。
  - (2)系统级上下文。包括进程控制块、内存管理信息、进程环境块,以及系统堆栈等组成的进程地址空间。
  - (3)寄存器上下文。由程序状态字寄存器、各类控制寄存器、地址寄存器、通用寄存器、用户堆栈指针等组成。

► 其中,用户级上下文和部分寄存器上下文对目态程序是可见的。

- **对于时间片轮转进程调度算法、不可抢占处理器的优先数调度算法、可抢占处理器的优先数调度算法,分别画出进程状态转换图**

- 

- [x] **什么是进程控制块?进程控制块一般包含哪些内容?**
  - 进程控制块是标识进程存在的数据结构,其中包含系统管理进程所需要的全部信息。
  - PCB一般包括如下信息:
    - 进程标识　用户标识　进程状态　调度参数　现场信息　家族联系　程序地址　当前打开文件　消息队列指针　资源使用情况　进程队列指针

- [x] **试比较进程状态与该进程内部线程状态之间的关系**
  - 对于用户级别线程,若同一进程中的多个线程中至少有一个处于运行态,则该进程的状态为运行态;若同一进程中的多个线程均不处于运行态,但是至少有一个线程处于就绪态,则该进程的状态为就绪态;若同一进程中的多个线程均处于等待态,则该进程的状态为等待态。

- [x] **什么是线程控制块?线程控制块中一般包含哪些内容?**
  - 线程控制块(Thread Control Block,TCB)是线程存在的标志,其中保存有系统管理线程所需要的全部信息。
  - 一般TCB中的内容较少,因为有关资源分配等多数信息已经记录于所属进程的PCB中。TCB中的主要信息包括线程标识、线程状态、调度参数、现场、链接指针。其中,现场信息主要包括通用寄存器、指令计数器PC以及用户栈指针。对于操作系统支持的线程,TCB中还应包含系统栈指针。

- **同一进程中的多个线程有哪些成分是共用的,哪些成分是私用的?**
  - 同一进程中的多个线程共享进程获得的主存空间和资源,包括代码区、数据区、动态堆空间。线程的私有成分包括:(1)线程控制块;(2)一个执行栈;(3)运行时动态分给线程的寄存器。

- **试比较用户级别线程与核心级别线程在以下几个方面的差别和各自的优、缺点：(1)创建速度;(2)切换速度;(3)并行性;(4)线程控制块的存储位置**

► 用户级别线程由系统库支持。线程的创建、撤销以及线程状态的变化都由库函数控制并在目态完成,与线程相关的控制结构线程控制块保存在目态空间并由运行系统维护。由于线程对操作系统不可见,系统调度仍以进程为单位,核心栈的个数与进程个数相对应。

► 用户级别线程的优点在于:
  - (1)线程不依赖于操作系统,可以采用与问题相关的调度策略,灵活性好;
  - (2)同一进程中的线程切换不需进入操作系统,因而实现效率较高。

► 缺点在于:
  - (1)同一进程中的多个线程不能真正并行,即使在多处理器环境中;
  - (2)线程对操作系统不可见,调度在进程级别。如果某进程中的一个线程通过系统调用进入操作系统受阻,那么该进程的其他线程也不能运行。

► 核心级别线程通过系统调用由操作系统创建,线程的控制结构TCB保存于操作系统空间,线程状态转换由操作系统完成,线程是CPU调度的基本单位。另外,由于系统调度以线程为单位,操作系统还需要为每个线程保持一个核心栈。

► 核心级别线程的优点是并发性好,在多CPU环境中同一进程中的多个线程可以真正并行执行。缺点是线程控制和状态转换需要进入操作系统完成,系统开销比较大。

- **试比较Linux系统中fork()与clone()两个系统调用之间的差异**

► fork()用于创建子进程,子进程与父进程具有各自独立的地址空间。子进程地址空间内容通过从父进程处复制得到。

► clone()用于创建子进程(线程),父子之间可以共享存储空间。

- **何谓系统开销?试举3个例子说明之**

► 运行操作系统程序,实现系统管理所花费的时间和空间称为系统开销。例如,操作系统的内核要占用内存空间,页面调度时需占用设备资源并消耗处理器时间,进程切换时也要占用处理器时间。

- **归纳引入多线程（multithread）程序设计的原因**

- ①某些应用具有内在的多个控制流结构，这些控制流具有合作性质，需要共享内存。采用多线程易于对问题建模，从而得到最自然的解决算法。
- ②在需要多控制流的应用中，多线程比多进程在速度上具有更大优势。统计测试结果表明，线程的建立速度比进程的建立速度快100倍，进程内线程间的切换速度与进程间的切换速度也有数量级之差。
- ③采用多线程可以提高处理器与设备之间的并行性。在单控制流的情形下，启动设备的进程进入核心后将被阻塞，此时该进程的其他代码也不能执行。若此时无其他可运行程序，处理器将被闲置。多线程结构在一个线程等待时，其他线程可以继续执行，从而使设备和处理器

并行工作。

- ④在有多个处理器的硬件环境中，多线程可以并行执行，既可提高资源利用效率，又可提高进程推进速度。

• 什么是PCB？PCB的作用是什么？PCB包含哪些内容？

- ▶ PCB是进程控制块的简称，是操作系统中用于描述和控制并发进程的数据结构
- ▶ PCB的**作用**是描述和控制并发进程；是进程存在的唯一标志；
- ▶ PCB中一般**包括** 进程标识 用户标识 进程状态 调度参数 现场信息 家族联系 程序地址 当前打开文件 消息队列指针 资源使用情况 进程队列指针 等内容。

• 作业与进程有何不同？它们之间有什么关系？

- ▶ 不同：
  - 作业：是用户在一次上机活动中，要求计算机系统所做的一系列工作的集合。也称作任务（task）。
  - 进程：是一个具有一定独立功能的程序关于某个数据集合的一次可以并发执行的运行活动。
  - 作业是一个宏观的执行单位，它主要是从用户的角度来看待的。作业的运行状态是指把一个作业调入内存，然后产生若干个进程可以去竞争CPU。
  - 进程是微观的执行单位，它主要从系统的角度来看待的，它是抢占CPU和其他资源的基本单位。进程的执行状态是指一个进程真正占用了CPU。
- ▶ 关系：一个作业调入内存以后，处于执行状态，则此作业对应在系统建立若干个进程。进程的所有状态对应作业的执行状态，通过这若干个进程的执行，来完成该作业。

☑ • 哪些事件会导致进程被创建？

- ▶ 有四种事件导致进程的创建：
  - （1）系统初始化。系统初始化会创建许多进程，如windows刚开机的时候。
  - （2）执行了正在运行的进程所调用的系统调用。如程序运行了一个fork()调用。
  - （3）用户请求创建一个进程。如命令行中输入./a.out。
  - （4）一个批处理作业的初始化。

☑ • 由核心返回目态程序时,进程的PSW和PC为何必须用一条机器指令同时恢复?

- ▶ 中断向量中程序状态字PSW和指令计数器PC的内容必须由一条指令同时恢复,这样才能保证系统状态由管态转到目态的同时,控制转到上升进程的断点处继续执行。
- ▶ 如果不同时恢复,则只能
  - 1.先恢复PSW再恢复PC,在恢复PSW后已经转到目态,操作系统恢复PC的使命无法完成
  - 2.先恢复PC再恢复PSW,PC改变后转到操作系统另外区域(因为PSW仍在系统状态),PSW无法恢复

- **进程切换时需要保存哪些现场信息？请尽量考虑完全。**
  - 进程切换过程是进程上下文的切换过程,进程上下文是指进程运行的物理环境。
  - 现场信息包括地址映寄存器、通用寄存器、浮点寄存器、SP、PSW(程序状态字)、PC(指令计数器)、以及打开文件表等。

第三章 中断处理与处理器调度

2024年6月19日 1:57

- **处理器资源管理需要解决3个问题?**

1. 按照什么原则分配处理器，即需要确定处理器调度算法
2. 什么时候分配处理器，即需要确定处理器调度时机
3. 如何分配处理器，即需要给出处理器调度过程

- **处理器调度算法的选择指标?**

①CPU利用率：使CPU尽量处于忙碌状态。
②吞吐量：单位时间内所处理计算任务的数量。
③周转时间：从计算任务就绪到处理完毕所用的时间。
④响应时间：从任务就绪到开始处理所用的时间。
⑤系统开销：系统调度进程过程中所付出的时空代价。

- **中断的响应过程主要有哪几个步骤?**

▶ ①识别中断源：当有多个中断源同时存在时，由中断装置选择优先级别最高的中断源。
▶ ②保存现场：将正在运行的进程的程序状态字及指令计数器中的内容压入系统栈。
▶ ③引出中断处理程序：将与中断事件相对应的中断向量由内存指定单元处取出并送入程序状态字及指令计数器中，如此便转入对应的中断处理程序。

- **中断有哪些类型？分别有什么特点?**

▶ 中断可以分为两大类，即强迫性中断和自愿性中断。
▶ 强迫性中断：这类中断事件是正在运行的程序所不期望的。强迫性中断事件是否发生，何时发生，事先无法预知，因而运行程序可能在任意位置被打断。这类中断大致有以下几种：
  ○ ①时钟中断：如硬件实时时钟到时等。
  ○ ②输入输出中断：如设备出错、传输结束等。
  ○ ③控制台中断：如系统操作员通过控制台发出命令等。
  ○ ④硬件故障中断：如掉电、内存校验错误等。
  ○ ⑤程序错误中断：如目态程序执行特权指令、地址越界、虚拟存储中的缺页故障或缺段 故障、溢出、除数为0等。
▶ 自愿性中断：这类中断事件是正在运行的程序有意识安排的。通常由于正在运行的程序执行访管指令而引起自愿性中断事件，其目的是要求系统提供某种服务。这类中断的发生具有必然性，并且发生位置确定。

- **用户栈有哪些用途?系统栈有哪些用途?**

▶ 用户栈的用途包括保存函数之间相互调用的参数、返回断点、局部变量、返回值。

- 系统栈的用途包括保存函数之间相互调用的参数、返回断点、局部变量、返回值;保存中断现场(PSW和PC)。

- **为什么说中断是进程切换的必要条件,但不是充分条件?**
  - 假如在时刻T1与时刻T2之间发生了进程切换,则在时刻T1与时刻T2之间一定执行了处理器调度程序,而处理器调度程序是操作系统低层中的一个模块,运行于管态,说明在T1与T2时刻之间处理器状态曾由目态转换到管态。由于中断是系统由目态转换为管态的必要条件,所以在时刻T1与时刻T2之间一定发生过中断,也就是说,中断是进程切换的必要条件,然而中断不是进程切换的充分条件。
  - 例如,一个进程执行一个系统调用命令将一个消息发给另外一个进程,该命令将通过中断进入操作系统得以执行,操作系统处理完消息的发送工作后可能返回原调用进程,此时中断未导致进程切换;也可能选择一个新的进程,此时中断导致了进程切换。

- **试分析中断与进程状态转换之间的关系。**
  - 进程状态转换是由内核控制的,如果一个进程的状态发生了改变,则在新旧状态之间一定发生了处理器状态由目态到管态的转换。而中断是处理器状态由目态转换到管态的必要条件,所以中断也是进程状态转换的必要条件。

- **中断发生时,旧的PSW和PC为何要压入系统栈?**
  - 因为通常中断处理程序的最后一条指令是中断返回指令,该指令从系统栈顶弹出断点信息,如果未将PSW和PC压入系统栈,则中断返回指令弹出的不是中断前的断点信息,而是不确定的信息,这将导致系统处于不确定的状态,严重时会使系统崩溃。
  - 采用栈结构的原因是中断可能发生嵌套,此时能保证以与发生中断相反的次序返回上层中断处理程序或返回目态。
  - 在某些硬件系统中,没有采用栈结构,中断发生时现场信息被送到系统空间指定单元,每个现场保存单元与中断事件一一对应。这样处理的缺点是中断类型不能增加,相同类型中断不能嵌套发生。

- **何谓中断向量?用户能否修改中断向量的值?**
  - 当中断事件发生时,中断装置根据中断类别自动地将中断处理程序所对应的PSW和PC送入程序状态字和指令计数器,如此便转移到对应的中断处理程序。这个转移类似于向量转移,因而PSW和PC被称为中断向量。
  - 用户不能修改中断向量的值,因为修改中断向量是特权指令,普通用户程序不能执行特权指令。另外,如果允许用户修改中断向量的值,那么用户就可以破坏中断向量与处理程序之间的联系,并可能攻击系统。例如,将中断向量与一段病毒程序联系起来,使中断发生时执行病毒程序,这将导致对计算机系统的破坏。

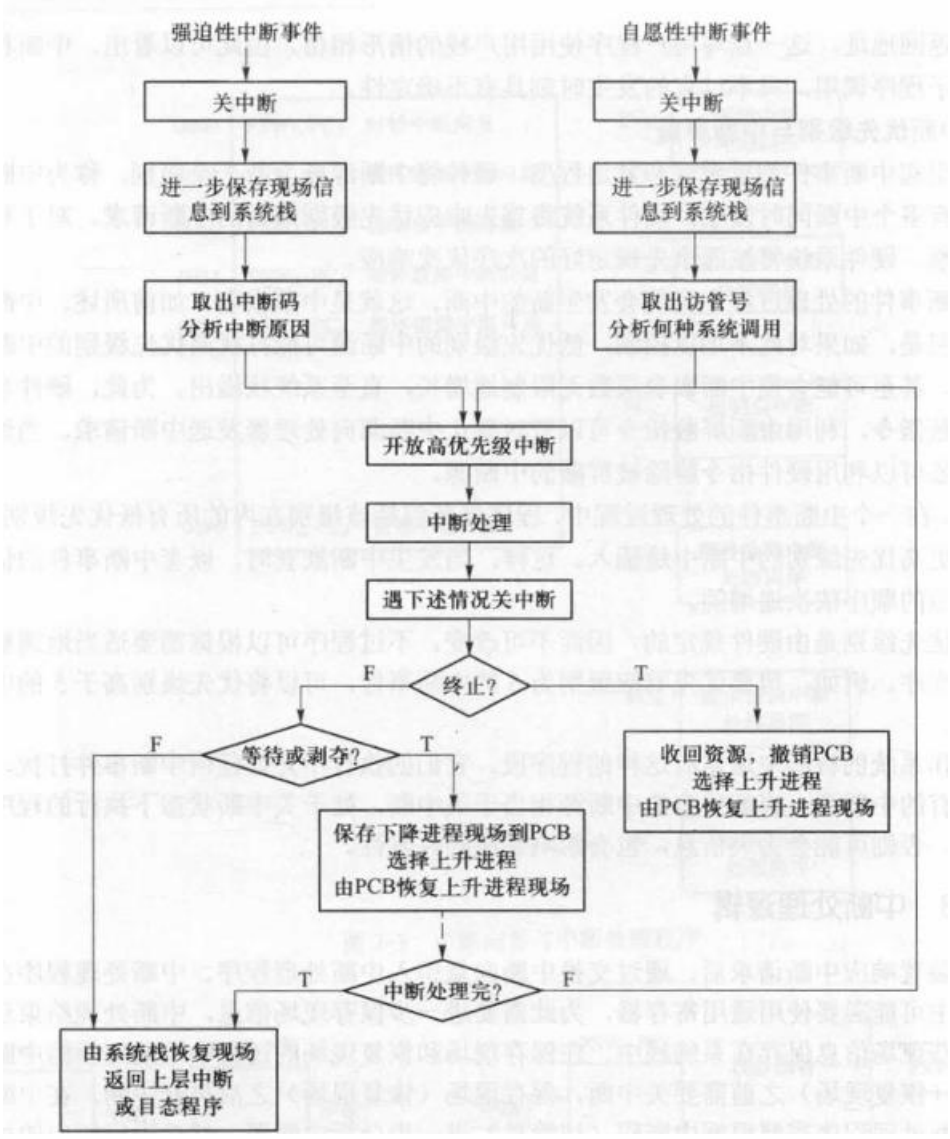
- **为什么需要中断屏蔽?**
  - 如果对中断嵌套不加以控制，低优先级别的中断源可能打扰高优先级别的中断事件的处

理过程，甚至可能会使中断嵌套层数无限制地增长，直至系统栈溢出。为此，硬件系统提供了中断屏蔽指令，利用中断屏蔽指令可以暂时禁止中断源向处理器发送中断请求。

- **描述中断处理逻辑，画出中断处理过程的流程图**

► 中断装置响应中断请求后，通过交换中断向量引入中断处理程序。中断处理程序在处理中断的过程中可能需要使用通用寄存器，为此需要进一步保存现场信息，中断处理结束后再恢复现场。这些现场信息保存在系统栈中，在保存现场和恢复现场的过程中不响应新的中断，为此保存现场（恢复现场）之前需要关中断，保存现场（恢复现场）之后再开中断。在中断处理过程中，中断处理程序需要根据中断码（访管号）进一步分析中断源，然后进行相应的处理，最后根据情况决定是否需要切换进程。

►



- **解释“中断现场”与“进程切换现场”的差异**

► 中断现场保存在系统栈中，进程切换现场保存在进程PCB中。

- **为什么保存在PCB中的现场都是核心级别的现场?**

- ► 等待和剥夺都是由操作系统完成的，因而等待和剥夺均发生在核心态，因而保存在PCB中的现场都是核心级别的现场，而并非目态程序的现场。

- **考虑系统状态与中断，画图解释进程状态转换关系**

- ►

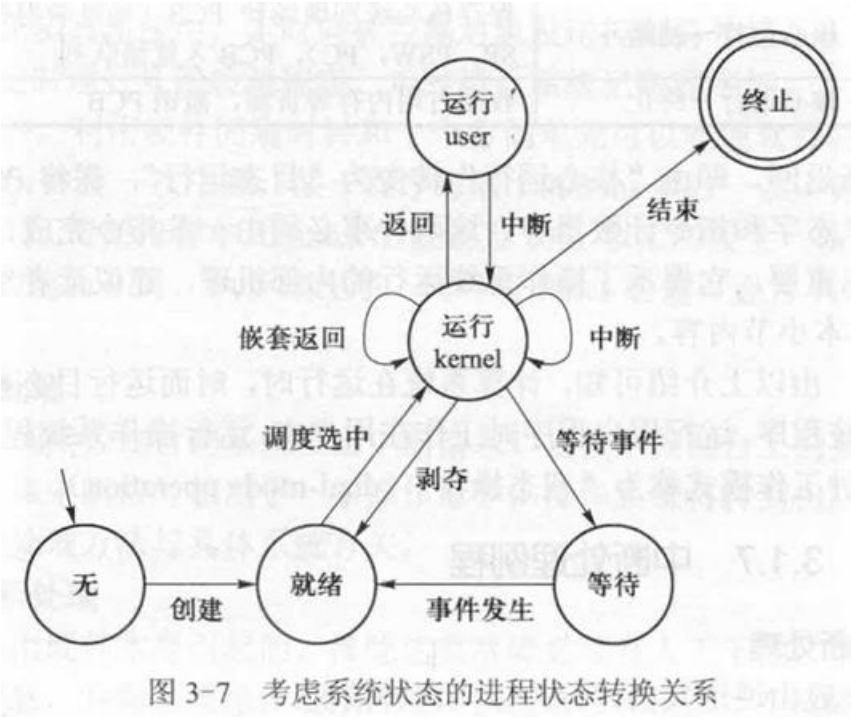


图 3-7　考虑系统状态的进程状态转换关系

- **试举一些中断处理的例子**
  - ► 输入输出中断处理
  - ► 时钟中断处理
  - ► 控制台中断处理
  - ► 硬件故障中断处理
  - ► 程序错误中断处理

- **中断向量的存储位置是否可由程序改变?为什么?中断向量的值是如何确定的?**
  - ► 中断向量的存储位置是由硬件确定的,不能由程序改变。中断发生后,中断装置按照中断类型到内存指定位置取出中断向量。
  - ► 操作系统的设计者根据各中断事件处理程序的存储位置及运行环境确定对应中断向量的值,系统启动时由初始化程序将该值填入指定位置。

- **有人说，“中断发生后,硬件中断装置保证处理器进入管态。”这种说法准确吗?试说明理由。**
  - ► 这种说法不准确。中断发生后,硬件中断装置负责引出中断处理程序,中断处理程序是否运行于管态取决于PSW中的处理器状态位,该位的值是在操作系统初始化时设置的,只有在初始化程序正确设置该状态位的前提下,才能保证中断后系统进入管态。

- **为什么在中断处理过程中通常允许高优先级别的中断事件中途插入,而不响应低优先级别的中断事件?**

▶ 根据中断事件的重要性和紧迫程度,硬件将中断源分为若干个级别,称为中断优先级。

▶ 如果有多个中断同时发生,硬件将首先响应优先级别最高的中断请求。对于优先级别相同的中断,硬件将按照事先规定好的次序依次响应。

▶ 在中断事件的处理过程中可能会发生新的中断,这就是中断嵌套。中断嵌套是必要的。但是,如果不加以控制,低优先级别的中断源可能打扰高优先级别中断事件的处理过程,甚至可能会使中断嵌套层数无限增长,直至系统栈溢出。为此,硬件提供了中断屏蔽指令,利用中断屏蔽指令可以暂时禁止任意一个或多个中断源向处理器发中断请求。当然,在需要的时候还可以利用硬件指令解除对中断源的屏蔽。

▶ 通常,在一个中断事件的处理过程中,程序屏蔽包括该级别在内的所有低优先级别的中断,但允许更高优先级别的中断中途插入。这样,发生中断嵌套时,嵌套中断事件的优先级别是按照响应的顺序依次递增的。这样做主要有两个原因:

  - (1)从逻辑上来说,高优先级别中断源所对应的事件比低优先级别中断源所对应的中断事件急迫;
  - (2)由于硬件中断类型是有限的,这样做实际上也就限制了中断嵌套的深度。

- **为什么说“关中断”会影响系统的并发性?**

▶ 考虑单处理器系统。在单处理器系统中,并发是通过将处理器轮流分配给多个进程而实现的,这个分配由操作系统中的处理器调度程序完成。中断是进程切换的必要条件,如果关中断,则操作系统无法获得处理器的控制权,也就无法使多个进程分时共享处理器,处理器将被一个进程独占。所以说“关中断”会影响系统的并发性。

▶ 关中断是实现临界区管理的简捷方法,而且没有“忙式等待”问题,但临界区应尽量小,执行完后要立即“开中断”,因为在“关中断”期间互斥已经不局限于临界区范围,而被扩展到了整个操作系统空间。

- **假如关中断后操作系统进入死循环,将会产生什么后果?**

▶ 系统不响应任何外部干预事件,系统表现为“死机”。

- **为什么不允许目态程序执行关中断指令及中断屏蔽指令?**

▶ 开/关中断指令和中断屏蔽指令属于特权指令,一般用户无权访问。如果允许用户使用,用户关中断后可能会影响系统对内部或外部事件的响应,也会使操作系统无法获得系统控制权。

- **如果没有中断,是否能够实现多道程序设计?为什么?**

▶ 不能。因为一个程序一旦被调度执行,就将一直执行下去,中间不可能被打断,不可能达到多个进程交替执行的并发目的。

- **下列中断源哪些通常是可以屏蔽的,哪些通常是不可屏蔽的?(1)输入输出中断;(2)访管中断;(3)时钟中断;(4)掉电中断。**
  - (1)输入输出中断可以屏蔽;(2)访管中断不可以屏蔽;(3)时钟中断可以屏蔽;(4)掉电中断不可以屏蔽。
  - 对于访管中断来说,若在管态屏蔽,则没有意义(不会发生访管中断);若在目态屏蔽,则应用程序无法访问操作系统,不能正常运行。

- **下列中断事件哪些可由用户自行处理?哪些只能由操作系统中断处理程序统一处理?为什么?(1)溢出;(2)地址越界;(3)除数为零;(4)非法指令;(5)掉电。**
  - 一般来说,只影响应用程序自身的中断,可以由用户自行处理,包括(1)溢出;(3)除数为零。
  - 可能影响其他用户或操作系统的中断只能由操作系统中断处理程序统一处理,包括(2)地址越界;(4)非法指令;(5)掉电。

- **如果中断由用户程序自行处理,为何需要将被中断程序的断点由系统栈弹出并压入用户栈?**
  - 中断发生时,被中断程序的现场信息已被压入系统栈中。而中断续元运行于目态,它执行完毕后将从用户栈区中恢复现场信息。为此,操作系统在转到中断续元之前应当将系统栈中的现场信息弹出并压入用户栈,否则,用户中断续元执行完毕后将无法恢复现场信息返回断点。

- **对于下述中断与进程状态转换之间的关系,各举两个例子说明之。**
  (1)定会引起进程状态转换的中断事件;
  (2)可能引起进程状态转换的中断事件。
  - 定会引起进程状态转换的中断事件:当前运行进程终止、应用程序启动I/O传输并等待I/O数据、运行程序申请当前被占用的某一资源。
  - 可能引起进程状态转换的中断事件:时钟中断事件可能引起进程状态转换。例如,对于时间片轮转进程调度算法,若时钟中断发生后当前进程的时间片已用完,则会发生进程切换;否则不发生进程切换。

- **若在T1时刻进程P1运行,在T2时刻进程P2运行,且P1≠P2,则在时刻T1~T2期间内一定发生过中断。这种说法对吗?为什么?**
  - 这种说法对。
  - 已知时刻T1进程P1在运行,时刻T2进程P2在运行,且P1≠P2,可知在时刻T1和时刻T2之间发生了进程切换。由于进程切换的必要条件是执行处理器调度程序,而处理器调度程序是操作系统低层中的一个模块,只能在系统态执行,而只有中断才能进入系统态。因此,假如

在时刻T1与时刻T2之间发生了进程切换,则在时刻T1与时刻T2之间一定发生过中断。

- **进程上下文包括哪些内容?请尽量考虑完整。**

► 进程上下文包括(1)通用寄存器;(2)PSW,PC,SP;(3)地址映射寄存器;(4)用户栈和系统栈;(5)进程控制块。

- **进程切换时,上升进程的PSW和PC为何必须由一条指令同时恢复?**

► 中断向量中程序状态字PSW与指令计数器PC的内容必须由一条指令同时恢复,这样才能保证系统状态由管态转到目态的同时,控制转到上升进程的断点处继续执行。如果不同时恢复,则只能(1)先恢复PSW,再恢复PC。由于在恢复PSW后已经转到目态,因此操作系统恢复PC的使命无法完成。(2)先恢复PC,再恢复PSW。由于PC改变后转到操作系统的另外区域(因为PSW仍为系统状态),因此PSW无法恢复。

- **简述先到先服务算法的思想与优缺点**

► 先到先服务（FCFS）算法按照进程申请CPU的次序，即进入就绪态的次序来调度。

► 先到先服务算法具有公平的优点，不会出现饿死的情况。

► 其缺点是短进程（线程）的等待时间长，从而平均等待时间较长。

- **简述最短作业优先算法的思想与优缺点**

► 最短作业优先（shortest job first，SJF）算法按照CPU阵发时间递增的次序调度，易于证明其平均周转（等待）时间最短。

► 最短作业优先算法最大限度地降低了平均等待时间，但是也带来不公平性：一个较长的就绪任务可能由于短任务的不断到达而长期得不到运行机会，发生饥饿，甚至被饿死。

- **简述最短剩余时间优先算法的思想**

► 最短剩余时间优先（shortest remaining time next，SRTN）算法是剥夺式调度算法，当CPU空闲时，选择剩余时间最短的进程或线程。当一个新进程或线程到达时，比较新进程所需时间与当前运行进程的估计剩余时间。如果新进程所需的运行时间短，则切换运行进程。该算法实质上是可剥夺形式的最短作业优先算法。

- **简述最高响应比优先算法的思想与优缺点**

► 最高响应比优先（highest response ratio next，HRN）算法是先到先服务算法和最短作业优先算法的折中，响应比的计算公式如下：

► $$RR = \frac{BT + WT}{BT} = 1 + \frac{WT}{BT}$$

► 其中，RR为响应比，BT为CPU阵发时间，WT为等待时间。显然，对于同时到达的任务，处理时间较短的任务将被优先调度，处理时间较长的任务将随其等待时间的增加而

动态提升其响应比，因而不会出现饥饿现象。

- **处理器的调度时刻有哪些选择方法？分别加以介绍**

► 非剥夺式：所谓非剥夺式（non—preemPtive，也称非抢先），就是一个进程不能将处理器资源强行地从正在运行的进程那里抢占过来。亦即一旦一个进程被进程调度程序所选中，它将一直运行下去， 直至出现如下两种情形：该进程因某种事件而等待，或者该进程运行完毕。这种方法的优点是：系统开销较小，其缺点是不能保证当前正在运行的进程永远是系统内当前可运行进程中优先数 最高的进程。

► 剥夺式：所谓剥夺式（PreemPtive，也称抢先），就是一个进程能够将处理器资源强行地从正在运行的进程那里抢占过来。此时，当发生如下3种情形时可能发生进程切换：正在运行的进程因某种事件而等待；正在运行的进程运行完毕；出现新的就绪进程，该进程的优先级高于正在运行的进程的优先级。注意，此处的“出现”有两种情况：其一是某一进程被唤醒（由等待态转换 为就绪态），其二是动态创建了新的进程。这种方法的优点是能够保证正在运行的进程永远是系统内当前可运行进程中优先数最高的进程；其缺点是处理器在进程之间的切换比较频繁，系统开销较大。

- **简述循环轮转算法的思想**

► 采用循环轮转（round—robin，RR）算法，系统为每一个进程规定一个时间片（t ime sliee），所有进程按照其时间片的长短轮流地运行。也就是说，将所有就绪进程排成一个队列，即就绪队列。每当处理器空闲时便选择队列头部的进程投入运行，同时分给它一个时间片。当此时间片用完时，如果此进程既未结束其CPU阵发期也未因某种原因而等待，则剥夺此进程所占有的处理器，将其排在就绪队列的尾部，并选择就绪队列中的队头进程运行。

- **举例说明分类排队算法的思想**

► 分类排队（multilevel queue，MLQ）算法是以多个就绪进程队列为特征的。这些队列将系统中所有可运行的进程按照某种原则加以分类，以实现所期望的调度目标。

► 例如，在通用操作系统中，可以将所有就绪进程排成如下3个队列。

  ○ Q1：实时就绪进程队列
  ○ Q2：分时就绪进程队列
  ○ Q3：批处理就绪进程队列

► 当处理器空闲时，首先选择Q1中的进程。如果该队列为空，则选择Q2中的进程。如果上述两个队列均为空，则选择Q3中的进程。在每个队列内部又可以分别采用最高优先数优先算法 和（或）循环轮转算法，如Q，和Q3采用最高优先数优先算法，必采用循环轮转算法。

- **简述反馈排队算法的思想，指出该算法有什么特点**

► 反馈（feedback）排队算法也是以多个进程就绪 队列为特征的，在每个队列中通常采用循环轮转算法，所不同的是，进程可以在不同的就绪队列之间移动。这些就绪队列的

优先级依次递减，而其时间片的长度则依次递增。 当一个进程被创建或所等待的事件发生时，进入第一级就绪队列。当某队列的一个进程获得处理器并且使用完该队列所对应的时间片后，如果它尚未结束，则进入下一级就绪队列。当最后 一级队列的一个进程获得处理器并月．使用完该队列所对应的时间片后，如果它尚未结束，则进 入同一级就绪队列。当处理器空闲时，先选择第一级就绪队列中的进程，当该级队列为空时， 选择第二级就绪队列的进程，以此类推。

- 这种进程调度算法有以下特点：
  - ①短进程优先处理，因为运行时间短的进程在经过前面几个队列之后便执行完毕，而这些队列中的进程将被优先调度。
  - ②设备资源利用率高，因为与设备打交道的进程可能会由于下面两个原因而进入等待态： 所申请的资源被占用，或者启动了数据传输。当进程得到所等待的资源或数据传输完成时，它 将进入第一级就绪队列，尽快获得处理器并使用资源。
  - ③系统开销较小，因为运行时间长的进程最后进入低优先级的队列，这些队列的时间片较长，因而进程的调度频率较低。

• 处理器调度需要经历哪几个阶段？分别加以介绍

- 1．保存下降进程现场
  - 当前核心态的现场信息（包括地址映射寄存器、通用寄存器以及SP 、 PSW 、 PC）被保存到下降进程的进程控制块中。注意，这里保存的是核心级别的现场，以后再次调度到该进程时将继续核心态尚未执行完的代码。 这里有两种特殊情形：系统初次启动时无下降进程，这一步不执行；下降进程为终止进程， 直接转善后处理。
- 2．选择将要运行的进程
  - 按照处理器调度算法在就绪队列中选择一个进程，准备将其投入运行，被选中的进程称为上升进程。为防止出现就绪队列为空的情况，在系统中通常安排一个特殊的进程，该进程永远也进行不完，且其调度级别最低，称为“闲逛”进程。当系统中无其他进程可运行时，就运行这个“闲逛”进程。 “闲逛”进程是容易构造的，如一个死循环程序、一个计算二值的程序等所对应的进程都可以作为“闲逛”进程。
- 3．恢复上升进程现场
  - 由于进程下降时已经将其现场信息保存在对应的进程控制块中，进程上升时应从其进程控制块中恢复现场。进程现场恢复的步骤是：先恢复地址映射寄存器、通用寄存器及SP等内容， 最后恢复PSW和PC 。注意：这里恢复的也是核心级别的现场。
- 进程调度的3个步骤需要执行一定数量的CPU指令才能完成。这些指令通常是用汇编语言编写的，而且经过认真优化，以尽量节省处理器时间。由于处理器调度是经常性的工作，节省一条指令都会对提高系统效率产生积极的影响。

• 某系统采用可抢占处理器的静态优先数调度算法,何时会发生抢占处理器的现象?

► 当一个新创建进程或一个被唤醒进程的优先数比正在运行进程的优先数高时,可能发生抢占处理器现象。

- **某系统采用可抢占处理器的动态优先数调度算法,何时会发生抢占处理器的现象?**

► 当一个新创建的进程或一个被唤醒进程的优先数比正在运行进程的优先数高时,或当一个就绪进程优先数超过正在运行进程优先数时,可能发生抢占处理器现象。

- **在实时系统中,采用不可抢占处理器的优先数调度算法是否合适?为什么?**

► 不合适。一旦一个低优先数、需要大量CPU时间的进程占用处理器,它就会一直运行,直到运行结束,或者直到因某事件而阻塞。在此之前,即使高优先数的紧急任务到达,也得不到处理。因此,可能延误对重要事件的响应和处理。

- **在分时系统中,进程调度是否只能采用时间片轮转调度算法?为什么?**

► 分时系统的特点是要求响应速度及时,除RR(Round Robin,循环轮转)调度算法之外,还可以采用可剥夺CPU的动态优先数调度算法。例如,经典UNIX的处理器调度算法,由于负反馈性质,算法也可以保证响应速度。

- **有人说,在采用等长时间片轮转处理器调度算法的分时操作系统中,各个终端用户所占处理器的时间总量是相同的。这种说法对吗?为什么?**

► 这种说法不对。因为处理器是分配给进程(线程)的,而不同终端用户可能有不同数量的进程,一个拥有较多数量进程的终端显然比拥有较少数量进程的终端获得CPU的时间要多。

- **举出两个例子说明操作系统访问进程空间的必要性。**

► (1)进程执行输出操作时,通过系统调用进入系统,由操作系统将待输出的数据由进程空间取出送给指定的外部设备,为此操作系统必须访问用户进程空间。

► (2)当发生可由用户自己处理的中断事件时,操作系统在转到中断续元之前应当将系统堆栈中的现场信息弹出并压入用户堆栈,为此操作系统也必须访问进程空间。

- **根据进程和线程的组成,说明进程调度和线程调度各需要完成哪些工作。**

► 进程调度需要完成的工作:(1)地址映射寄存器;(2)用户栈指针;(3)通用寄存器;(4)PSW与PC。

► 线程调度需要完成的工作:(1)用户栈指针;(2)通用寄存器;(3)PC。

- **系统资源利用率与系统效率是否一定成正比?如果不是,试举例说明**

- 系统效率高则资源利用率高,而反之却不尽然。例如,在虚拟页式存储管理系统中,当页面置换算法不合理或分给进程的页框数过少时,可能发生颠簸或抖动(thrashing),此时I/O设备很忙碌,但系统效率可能很低。

- **画出具有中级调度的进程状态转换关系图**

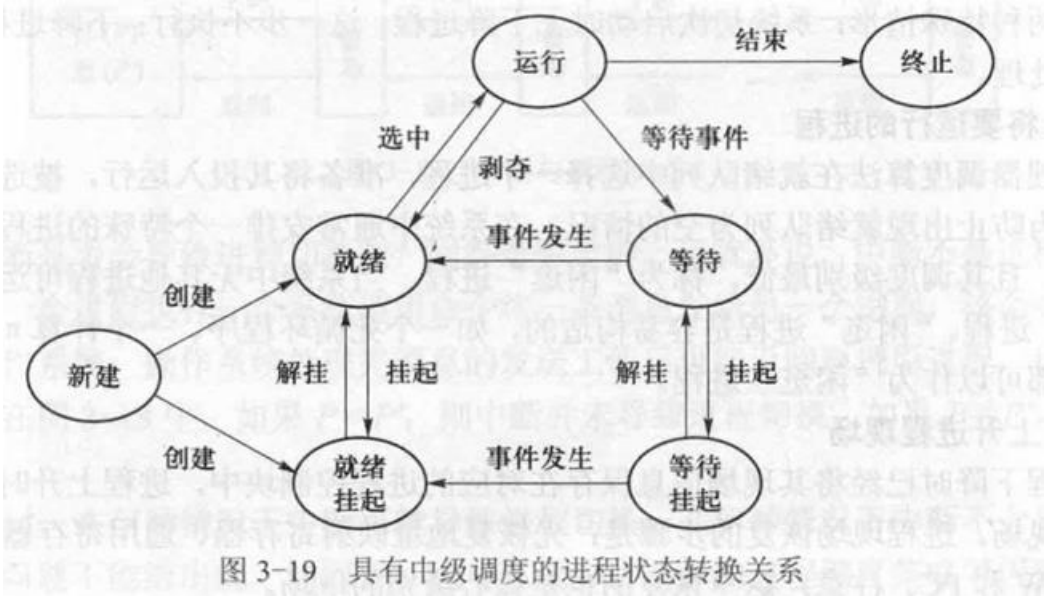


图 3-19 具有中级调度的进程状态转换关系

- **什么是低级调度?低级调度的职能是什么?**
- 负责分派处理器的调度称为低级调度,也称处理器调度。低级调度使被选中的进程真正进入运行状态。

- **什么是中级调度?中级调度的职能是什么?**
- 介于低级调度与高级调度之间的调度称为中级调度。在UNIX系统中,中级调度负责进程在内存和外存之间进行交换,以缓解内存资源紧张的矛盾。

- **什么是高级调度?高级调度的职能是什么?**
- 高级调度又称作业调度,负责将作业由输入井调入内存,并为其创建作业控制进程。

- **一个用C和汇编两种语言书写的程序，作为批作业处理主要包括哪几个步骤?**
- 运行C语言编译程序对C代码部分进行编译，运行汇编程序对汇编代码部分进行汇编，运行连接装配程序对前两步产生的浮动程序进行连接装配，执行上一步产生的目标代码。

- **画图解释作业的状态转换关系**

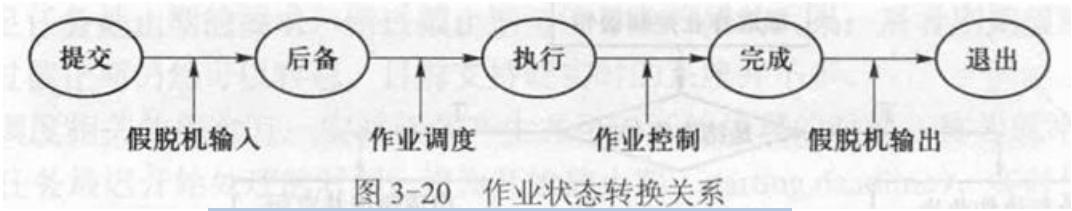
- 

图 3-20 作业状态转换关系
- 作业一般经历“提交” “后备” “执行” “完成”和“退出”这5个状态：
  - 由输入机向输入井传输的作业处于“提交”状态，
  - 进入输入井尚未调入内存的作业处于“后备”状态，
  - 被调度选中进入内存处理的作业处于“执行”状态，
  - 处理结果传送到输出井的作业处于“完成”状态，
  - 由输出井向打印设备传送的作业处于“退出”状态。
  - 作业由“提交”到“后备”的状态转换由假脱机输入完成，
  - 由“后备”到“执行” 、由“执行”到“完成”的状态转换由作业调度完成，
  - 由“完成”到“退出”的状态转换由假脱机输出（SPOOL输出）完成。

• 作业调度和进程调度各自的主要功能是什么？

- 作业调度的主要功能是：
  - ①记录系统中各个作业的情况；
  - ②按照某种调度算法从后备作业队列中挑选作业；
  - ③为选中的作业分配内存和外设等资源；
  - ④为选中的作业建立相应的进程；
  - ⑤作业结束后进行善后处理工作。
- 进程调度的主要功能是：
  - ①保存当前运行进程的现场；
  - ②从就绪队列中挑选一个合适进程（使用一定的调度算法），将其状态改为运行态，准备分配CPU给它；
  - ③为选中的进程恢复现场，分配CPU。

• 说明作业调度与进程调度的区别？

1. 作业调度是宏观调度，它所选择的作业只是具备获得处理机的资格，但尚未占有处理机，不能立即在其上实际运行；而进程调度是微观调度，它动态地把处理机实际地分配给选中进程，使之活动；
2. 进程调度相当频繁，而作业调度的执行次数很少；
3. 有的系统可以不设作业调度，但进程调度必不可少。

• 作业与进程有何不同？它们之间有什么关系？

- （1）不同：
  - 作业：是用户在一次上机活动中，要求计算机系统所做的一系列工作的集合。也称作任务（task）。
  - 进程：是一个具有一定独立功能的程序关于某个数据集合的一次可以并发执行的运

行活动。
    - 作业是一个宏观的执行单位，它主要是从用户的角度来看待的。
    - 作业的运行状态是指把一个作业调入内存，然后产生若干个进程可以去竞争CPU。
    - 进程是微观的执行单位，它主要从系统的角度来看待的，它是抢占CPU和其他资源的基本单位。
    - 进程的执行状态是指一个进程真正占用了CPU。
- ▶ 关系：一个作业调入内存以后，处于执行状态，则此作业对应在系统建立若干个进程。进程的所有状态对应作业的执行状态，通过这若干个进程的执行，来完成该作业。

- **实时任务按照发生规律可以分为哪几类？按照时间约束的强弱程度又可以分为哪几类？**

- ▶ 实时任务按其发生规律可以分为以下两类：
    - ①随机性实时任务：由随机事件触发，其发生时刻不确定。
    - ②周期性实时任务：每隔固定时间发生一次。
- ▶ 按对时间约束的强弱程度又可分为“硬实时” 和“软实时” 。 前者必须满足任务截止期的要求，错过截止期可能导致严重的后果；后者则期望满足截止期要求，但是错过截止期仍然可以容忍。目前支持硬实时的系统并不多。

- **简述最早截止期优先调度的思想**

- ▶ 最早截止期优先（earliest deadiine first , EDF）调度优先选择完成截止期最早的实时任务。对于新到达的实时任务，如果其完成截止期先于正在运行任务的完成截止期，则重新分派处理器，即剥夺。

- **简述单调速率调度的实现思想**

- ▶ 单调速率调度（ RMS）面向周期性实时任务，属于非剥夺式调度的范畴。速率单调调度将任务的周期作为调度参数，其发生频度越高，则调度级别越高。

- **什么是中断向量？什么是多级中断？中断处理的过程一般有哪几步？**

- ▶ 当中断事件发生时,中断装置根据中断类别自动地将中断处理程序所对应的PSW和PC送入程序状态字和指令计数器,如此便转移到对应的中断处理程序。这个转移类似于向量转移,因而PSW和PC被称为中断向量
- ▶ 多级中断：为了便于对同时产生的多个中断按优先次序来处理，所以在设计硬件时，对各种中断规定了高低不同的响应级别。优先权相同的放在一级。
- ▶ 中断处理步骤：响应中断，保存现场；分析中断原因，进入中断处理程序；处理中断；恢复现场，退出中断。

- **列举一定能引起进程切换的中断原因、可能引起进程切换的中断原因**

- ▶ 一定能引起进程切换的中断原因有进程运行终止、进程等待资源、进程等待数据传输的完成等

► 可能引起进程切换的中断原因有时钟中断、接收到设备输入输出中断信号等。

• 在OS中，引起进程调度的因素有哪些?

1. 完成任务；正在运行的进程完成任务，释放CPU
2. 等待资源；等待资源或事件，放弃CPU
3. 运行时刻；规定时间片已用完，时钟中断，让出CPU
4. 发现标志；核心处理完中断或陷入事件后，发现“重新调度标志”被置上，执行进程调度。

第四章 互斥、同步与通信

2024年6月24日 13:37

- **并发程序有哪些特性？分别加以解释**
  - ①间断性。多个程序是交叉执行的，处理器在执行一个程序的过程中有可能被中断，并转而执行另一个程序。
  - ②非封闭性。一个进程的运行环境可能被其他进程所改变，从而相互产生影响。
  - ③不可再现性。由于交叉的随机性，并发程序的多次执行可能对应不同的交叉，因而不能期望重新运行的程序能够再现上次运行时产生的结果。

- **什么是与时间有关的错误？试举例说明**
  - 并发进程的执行实际上是进程活动的某种交叉,某些交叉次序可能得到错误结果。由于具体交叉的形成与进程的推进速度有关,而速度是时间的函数,因而将这种错误称为与时间有关的错误。
  - 例如,两个并发进程的程序如下。

```
int n=0;
main( ){
   创建进程 A;
   创建进程 B;
};
A( ){                          B( ){
   while(1){                      while(1){
      n++;                           睡眠一段时间;
   }                                 printf("%d",n);
};                                   n=0;
                                   }
                                };
```

  - 假设进程 A 被部署在公园入口的终端上,用来记录一段时间内进入公园的人数,进程 B 被部署在公园的控制中心,用来输出一段时间内进入公园的总人数。进程 A 和进程 B 共享全局变量 n,n 表示记录下的人数。如果进程 B 执行完打印语句后被进程 A 打断,进程 A 执行了若干次变量自增语句,之后进程 B 接着执行清 0 语句,那么进程 A 对 n 的累加结果被丢失,相当于在进程 B 被打断的这段时间内进入公园的人没有被记录下来。这就发生了与时间有关的错误。

- **有人说,假设两个进程之间没有共享内存,则两者之间没有公共变量。这种说法准确吗?试说明原因。**
  - 如果只从用户空间考虑,这种说法是正确的。但从操作系统的角度来说并不准确。两个没有公共内存的用户进程可能同时(宏观)进入操作系统,并访问操作系统空间中的公共变量。

- **何谓忙式等待?是否还有其他方式的等待?比较它们之间的联系和差别。**
  - 不进入等待状态的等待称为忙式等待。
  - 另一种等待方式是阻塞式等待,进程得不到共享资源时将进入阻塞状态,让出CPU给其他进程使用。
  - 忙式等待和阻塞式等待的相同之处在于进程都不具备继续向前推进的条件,不同之处在于尽管CPU可能被剥夺,但处于忙式等待的进程不主动放弃CPU,因而是低效的;而处于阻塞状态的进程主动放弃CPU,因而是高效的。

- **与排队等待相比,忙式等待是否一定是低效的?为什么?**
  - 不绝对。如果进程切换所带来的系统开销比忙式等待时间还长,那么忙式等待更具有优势。

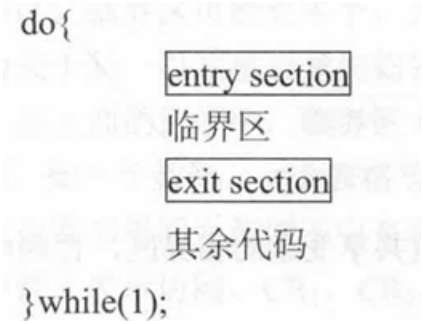
- **下列进程互斥方法中,哪些存在忙式等待问题?**
  (1)软件:面包店算法;(2)硬件:“测试并设置”指令;(3)开关中断指令。
  - (1)、(2)存在忙式等待问题。

- **为何开关中断进程互斥方法仅在单处理器系统中是有效的?**
  - 关中断方法不适用于多处理器系统,因为关中断只能保证处理器不由一个进程切换到另外一个进程,以防止多个进程并发地进入公共临界区域。但即使在关中断后,不同进程也可以在不同处理器上并行执行关于同一组共享变量的临界区代码。

- **在多处理器系统中,软件互斥方法是否有效?为什么?**
  - 依然有效。多处理器并行与单处理并发之间的差别在于程序交叉的粒度,单处理器环境中进程交叉发生在指令之间,多处理器环境中进程交叉发生在指令周期之间。由于纯软件互斥算法并不依赖特殊的硬件指令(如test_and_set),指令之间的交叉与指令周期之间的交叉结果相同。

- **共享变量属于操作系统空间还是用户进程空间?**
  - 共享变量既可能属于操作系统空间，也可能属于用户进程空间。对于前者， 其临界区也属于操作系统空间，而对于后者，其临界区则属于用户进程空间。

- **画出临界区的框架，并介绍管理临界区的3个正确性原则**

- ▶ 

- ▶ 管理应当满足下面3个正确性原则。
  - ①互斥性原则。任意时刻至多只能有一个进程处于关于同一组共享变量的临界区之中。
  - ②进展性原则。当临界区空闲时，只有那些执行entry section和exit seetion的进程参与下一个进入临界区进程的决策，该决策不能被无限期地推迟。
  - ③有限等待性原则。一个请求进入临界区的进程应当在有限的等待时间内获得进入该临界区的机会。

- **进程互斥有哪些常见的软件实现方法？其中哪些方法存在忙式等待的问题?**

- ▶ Dekker互斥算法
- ▶ Peterson互斥算法
- ▶ Lamport面包店算法
- ▶ Eisenberg—Mcguire算法
- ▶ 其中Lamport面包店算法、Eisenberg—Mcguire算法存在忙式等待的问题

- **进程互斥有哪些常见的硬件实现方法？其中哪些方法存在忙式等待的问题?**

- ▶ 硬件提供存储障碍语句
- ▶ 硬件提供原子变量
- ▶ 硬件提供“测试并设置”指令
- ▶ 硬件提供“交换”指令
- ▶ 硬件提供“开关中断”指令

- **列举常见的进程同步机制，简要介绍其思想**

- ▶ 信号量与PV操作
  - 这种同步机制包括一种称为信号量类型的变量以及对于此种变量所能进行的两个操作，即P操作和V操作。信号量变量包括两个部分，即值部分s . value和指针部分s . queue 。P操作中调用asleep函数，执行此操作的进程的进程控制块进入队列s—>queue的尾部，其状态由运行态转为等待态，系统转到处理器调度程序。V操作中调用wakeup函数，将队列s—>queue头部进程的进程控制块由该队列中取出，并将其排入就绪队列，其状态由等待态转为就绪态。
- ▶ 条件临界区

- 条件临界区（CCR）是一种高级的同步机制，其语法格式为“regionl r when b do s”，其中，r为一组相关的共享变量，b为条件表达式，s为语句。进程能够进入条件临界区执行，语句的条件是：没有其他进程处于与r相关的任何条件临界区内，进入条件临界区时b为真。

▶ 管程

- 管程的基本思想是将共享变量以及对于共享变量所能执行的所有操作集中在一 个模块中，一个操作系统或并发程序由若干个这样的模块所构成。由于一个模块通常比较短， 模块之间联系清晰，提高了可读性，便于修改和维护，易于保证正确性。

▶ 会合

- 在Ada语言中，一个任务可以有若干个入口，一个入口对应一段程序，一个任务可以调用另一个任务的入口。当一个任务调用另一个任务的入口，而且被调用者已经准备好接受这个调用时，便发生会合。当调用者发出调用请求，被调用者尚未准备好接受这个调用时，调用者等待：反之，当被调用者准备好接受调用，而当前尚无调用者时，被调用者等待，即先到达者等待后到达者，这就是会合的本质含义。

- **试分析临界区域的大小与系统并发性之间的关系**

▶ 关于同一组变量的临界区域是不能并发执行的代码,临界区越大,并发性越差。因此,编写并发程序应尽量缩小临界区域范围。

- **设CR1是关于一组共享变量SV1的临界区,CR2是关于另外一组共享变量SV2的临界区,当进程P1进入CR1时,进程P2是否可以进入CR2?为什么?**

▶ 可以。因为互斥是在变量级别上的,多个进程同时进入关于不同变量的临界区不会引起与时间有关的错误。

- **Lamport面包店互斥算法是否会出现饿死的情况?**

▶ 不会,该算法是公平的。假定系统中共有n个进程,每个想要进入临界区域的进程(线程)在最坏的情况下需要等待其他n-1个进程进入并离开临界区域之后才可获得进入临界区域的机会,因而存在(忙式)等待的上界。

- **由V操作唤醒的进程是否一定能够直接进入运行态?试举例说明**

▶ 否。一般来说,唤醒是将进程状态由等待状态变成就绪状态,而就绪进程何时获得处理器则是由系统的处理器调度策略确定的。

- **如果采用抢占式优先级调度算法,并且被唤醒的进程是当前系统中优先级最高的进程,那么该进程将被调度执行,其状态变成运行态。如果该进程不是系统中优先级最高的进程或系统采用非剥夺式调度算法,那么**

该进程不会被调度执行,其状态将维持在就绪状态。虽然管程是互斥进入的,但管程中定义的外部子程序必须是可载入的,试说明原因。

- 管程互斥是在变量级别的,同一管程类型可以有多个实例,而管程内部子程序只在管程类型定义时生成一套代码。为保障对不同管程变量的操作具有相同语义,管程外部子程序必须是可载入的。

- **管程与会合这两种同步机制之间的主要差别何在?**
  - 管程与会合都属于集中式结构化同步机制,但二者的实现机理完全不同。
  - 管程是被动性语言成分,管程本身不能占有处理器,管程的外部子程序是由调用进程占有处理器来执行的。
  - 会合是主动性语言成分,一个任务调用另一任务的入口时,入口代码是由被调用任务自己占有处理器执行的。

- **进程间因为独占资源的竞争产生的联系是什么关系?如何区分进程间的互斥与同步关系?**
  - 互斥关系
  - 同步关系是进程间有着某种先后的执行顺序关联；互斥关系是不同进程对独占资源的占有产生的关系，当一个进程占用了独占资源，就会进入临界区，其他进程无法占用独占资源，当它退出临界区时，其他进程才能按照队列顺序来占用独占资源

- **如何理解“与时间有关的错误”，错误出现的原因，为什么会随机出现错误?**
  - 多个线程或者进程在读写一个共享数据时结果依赖于它们执行的相对时间的情形。
  - 出现的原因就是因为代码的执行有先后，当两段代码同时对一个全局变量和文件进行修改时，由于代码执行时间的先后，这种错误往往是随机的，因为不确定是先读后写，还是先写后读，还是你改我读还是你读我改。

- **现实中，有身份证号码重复的情况，出现这种情况的原因？用操作系统思想说明**
  - 两个人同时申请身份证，在第一个人申请身份证时给一个身份证号A，访问共享的储存身份证的地方，判断该身份证号未重复，此时该身份证号还没有入库，就中断该进程，处理另一个人申请，由于两人信息非常相似且身份证号A还没有入库，给一个跟上一个人相同的身份证号，查重时没有查出来，因此两个人身份证号相同。出现与时间有关的错误。

- **信号量机制为什么能够避免出现与时间有关的错误?**
  - 对于信号量的操作是原语，不能被调度机制中断，从而避免了修改过程中由于被中断而引起与时间有关的错误。

- **信号量机制中的系统调用为什么要设为原语?**
  - 与时间有关的错误往往是由于对公共变量修改或判断前后被中断而引起的，而信号量本身也是一个公共变量，将对信号量的系统调用整个过程作为一个整体设置为原语，保证信号量的不可分割，才避免发生与时间有关的错误。

- **信号量机制的缺点**
  - P和V的操作散布在同步进程中的各处，操作不当容易造成死锁

- **管程机制中的活动进程指的是什么进程？为什么在管程中只能有一个活动进程?**
  - 活动进程：指的是处于就绪态或者是运行态的进程。
  - 管程每次只允许一个进程进入，从而实现了进程的互斥，避免死锁

- **管程机制相对于信号量机制的优缺点**
  - 优点，只能存在一个活动进程，避免死锁，可靠性高；对于共享数据的操作进行封装，利用编程。
  - 缺点：不能并发，效率降低

- **管程机制能满足临界区使用规则吗？为什么?**
  - 能，因为管程中有且只有一个活动进程，而被阻塞的进程不访问临界区

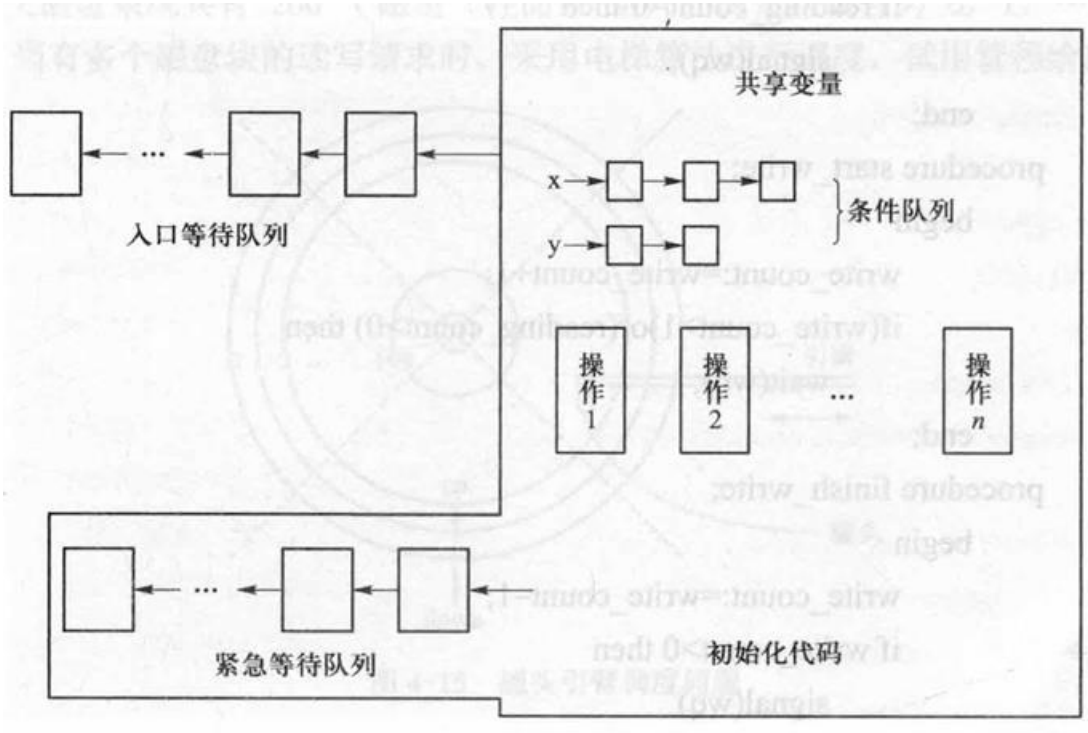
- **画图说明管程的组成**
  - 

图 4-14　包含各种队列的管程

第五章 死锁与饥饿

2024年6月25日 2:14

- [x] **死锁有哪几种类型?**
  - 竞争资源引起的死锁
  - 进程间通信引起的死锁
  - 其他原因引起的死锁

- [x] **简述死锁的条件**
  - ①资源独占。一个资源在同一时刻只能被分配给一个进程。如果某一进程申请某一资源， 而该资源正在被另外的某一进程所占有，则申请者需要等待，直到占有者释放该资源。
  - ②不可剥夺。资源申请者不能强行地从资源占有者手中夺取资源，即资源只能由其占有者在使用完后自愿地释放。
  - ③保持申请。进程在占有部分资源后还可以申请新的资源，而且在申请新资源的时候并不释放它已经占有的资源。
  - ④循环等待。存在一个进程等待序列{p1，p2，…，pn}，其中p1等待p2所占有的某一资源， p2等待p3所占有的某一资源… …pn等待p1所占有的某一资源。

- [x] **不让死锁发生的策略有哪些?**
  - 不让死锁发生的策略可以划分为两种：一种是静态的，称为死锁预防；另一种是动态的，称为死锁避免
  - 所谓静态的，是指对于进程有关资源的活动按某种协议加以限制，如果所有进程都遵守此协议，即可保证不发生死锁；
  - 所谓动态的，是指对于进程有关资源的申请命令加以实时检测，拒绝不安全的资源请求命令， 以保证死锁不会发生。

- **常见的死锁预防策略有哪些策略？各自有什么特点？有什么缺点?**
  - 预先分配策略：采用预先分配策略，进程在运行前一次性地向系统申请它所需要的全部资源。如果系统当前不能满足进程的全部资源请求，则不分配资源，此进程暂不投入运行。如果系统当前能够满足进程的全部资源请求，则一次性地将所申请的资源全部分配给申请进程。这种策略有以下一些缺点：
    - ①资源利用率低。
    - ②进程在运行前可能并不知道它所需要的全部资源。
  - 有序分配策略：采用这种策略，事先将所有资源类完全排序，即对每一个资源类赋予唯一的整数。进程必须按照资源编号由小到大的次序申请资源。也就是说，当进程不占有任何资源时，它可以申请某一资源类（如ri）中的任意多个资源实例。此后，它可以申

请另一个资源类（如rj）中的若干个资源实例的充分必要条件是f(ri)<f(rj)。有序分配法不会发生死锁。有以下一些缺点：

- 给资源类一个合理的编号比较困难；
- 按编号申请资源增加了资源使用者即进程的负担；
- 如果有进程违反了按编号申请的规定，则仍然有可能发生死锁；
- 为了保证按编号申请的次序，暂不需要的资源也可能需要提前申请，增加了进程对于资源的占有时间。

- **简述饿死与死锁的联系与区别**

▶ 联系：二者都是由于竞争资源而引起的

▶ 区别：

- ①从进程状态考虑，死锁进程都处于等待态。忙式等待（处于运行态或者就绪态）的进程并非处于等待态，但是却有可能被饿死。
- ②死锁进程等待永远不会被释放的资源，饿死进程等待会被释放但却不会分配给自己的资源，其等待时限没有上界（排队等待或忙式等待）。
- ③死锁一定是发生了循环等待，而饿死则不然。这也表明通过资源分配图可以检测死锁存在与否，但却不能检测是否有进程饿死。
- ④死锁一定涉及多个进程，而饥饿或被饿死的进程可能只有一个。

- **下面关于死锁问题的叙述,哪些是正确的,哪些是错误的?试说明判断原因。**

  (1)参与死锁的所有进程都占有资源。

  (2)参与死锁的所有进程中至少有两个进程占有资源。

  (3)死锁只发生在无关进程之间。

  (4)死锁可以发生在任意进程之间。

▶ (1)是错误的,应该是参与死锁的所有进程都等待资源。

▶ (2)正确,参与死锁的进程至少有两个

▶ (3)错误,死锁也可能发生在相关进程之间。

▶ (4)正确,死锁既可能发生在相关进程之间,也可能发生在无关进程之间，即死锁可以发生在任意进程之间

- **什么叫饥饿?什么叫饿死?什么叫活锁?试举例说明之。**

▶ 在一个动态系统中,资源请求与释放是经常发生的进程行为。对于每类系统资源,操作系统需要确定一个分配策略,当多个进程同时申请某类资源时,由分配策略确定资源分配给进程的次序。资源分配策略可能是公平的(fair),能保证请求者在有限的时间内获得所需资源;资源分配策略也可能是不公平的(unfair),即不能保证等待时间上界的存在。在后一种情况下,即使系统没有发生死锁,某些进程也可能会长时间等待。如果等待时间给进程推进和响应带来明显影响,则称发生了进程饥饿(starvation),当饥饿到一定程度的进程所赋予的

任务即使完成也不再具有实际意义时，称该进程被**饿死**(starve to death)。

- 考虑一台打印机分配的例子。当有多个进程需要打印文件时,系统按照短文件优先的策略排序,该策略具有平均等待时间短的优点,似乎非常合理,但当短文件打印任务源源不断时,长文件的打印任务将被无限期地推迟,导致饥饿乃至饿死。
- 与饥饿相关的另外一个概念称为活锁(live lock),在忙式等待条件下发生的饥饿,称为**活锁**。例如不公平的互斥算法。

- **何谓银行家算法的保守性?**
  - 银行家算法的保守性是指银行家算法只给出了进程需要资源的最大量,而所需资源的具体申请和释放顺序仍是未知的,因而银行家只能往最坏处设想。

- **能否给出避免死锁的充要性算法?为什么?**
  - 目前关于避免死锁的算法,如银行家算法是充分性算法,即确保系统时刻处于安全状态,这是在系统已知每个进程所需资源最大量的条件下可以给出的最好结果。
  - 如果系统不仅知道每个进程所需资源的最大量,而且知道进程有关资源的活动序列,在这个更强的条件下,则可以给出避免死锁的充要性算法(读者可以证明),但其复杂度是很高(NP完全)的。而且由于程序中分支和循环的存在,事先给出进程有关资源的命令序列一般是不可能的。

- **进程是否会因为竞争处理器资源而发生死锁?为什么?**
  - 不会,因为处理器资源是可剥夺的,参与死锁的进程都处于等待状态,这种等待不包括对处理器的等待,等待处理器的进程处于就绪状态。

- **饥饿进程可能处于什么状态?试举例说明之。**
  - 忙式等待的饥饿进程处于“就绪”或“运行”状态,例如改进之前的test_and_set互斥算法。排队等待的饥饿进程处于“等待”状态,例如利用经典信号量和PV操作实现的具有优先级的PP操作与VV操作。

- **有人对哲学家就餐问题的描述作了如下修改:哲学家中既有右撇子,又有左撇子。右撇子饥饿时先取右边的叉子,后取左边的叉子;左撇子饥饿时先取左边的叉子,后取右边的叉子。试回答:是否有死锁?是否有饥饿?为什么?**
  - (1)没有死锁,因为破坏了循环等待的条件;(2)没有饥饿,把叉子作为资源用初值为1的信号量管理,申请时执行P操作,释放时执行V操作,饥饿的进程终能获取左、右两把叉子,并进食。

- **为什么将所有资源按类型赋予不同的序号，并规定所有的进程按资源**

号递增的顺序申请资源后，系统便不会产生死锁?

- 此时系统不会发生死锁的原因是死锁发生的必要条件之一——循环等待条件不可能成立。因为多个进程之间只可能存在占据较低序号资源的进程等待占据较高序号资源的进程释放资源的情况，但不可能存在反向的等待，因此它们之间不会形成循环等待链。

第六章 主存储器管理

2024年6月21日 13:31

• 存储管理的基本任务是什么?

1. 存储分配
2. 存储共享
3. 存储保护
4. 存储扩充
5. 地址映射

• 地址映射流程图

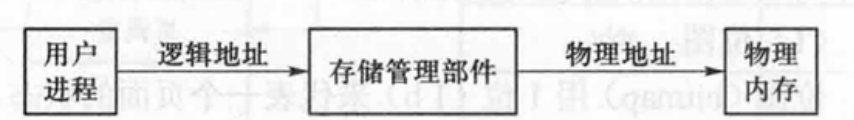


图 6-1 地址映射

• 内存有哪些分区方法?实际系统中一般怎么分区?

- 静态分区与动态分区
  - 静态分区:在系统运行之前就将内存空间划分为若干个区域,通常,分配给进程的内存区域可能比进程实际所需要的区域长。
  - 动态分区:在系统运行的过程中划分内存空间,通常,系统可以按照进程所需存储空间的大小为其分配恰好满足要求的一个或者多个区域。
- 等长分区与异长分区
  - 等长分区:等长分区是指将存储空间划分为若干个长度相同的区域。
  - 异长分区:是指将存储空间划分为若干个长度不同的区域
- 在实际系统中,静态分区与等长分区通常结合在一起,构成静态等长分区的分配形式;动态分区与异长分区通常结合在一起,构成动态异长分区的分配形式。

• 静态等长分区的分配适用于什么管理方式?有哪些分配与去配算法?各自有什么特点?

- 这种存储分配形式常用于页式存储管理方式与段页式存储管理方式中。
- 常用位图、空闲页表、空闲页链
  - 位图(bitmap)用1位(1b)来代表一个页面的状态,规定当其值为0时,表示该页面为空闲;当其值为1时,表示该页面被占用。
  - 空闲页表(free page table)中包括两个栏目:首页号和页数。若干个连续的空闲页作为一组登记在空闲页表中。采用此种分配方法能使一个进程的若干个页面尽量连续。
  - 将所有的空闲页面连成一个空闲页链(idle page link),分配时可以取链头的页,去配时可以将被释放的页面连入链头。此种方法适用于内存页面的分配, 但是对于外存

页面的分配因分配和去配均需执行一次数据传输，速度较慢。

- **为什么空闲页链不适合管理外存?**

  不适合管理外存，比如外存要分配几个空闲块，读取head指向的第一个空闲块，再分配下一个空闲块的时候，需要把当前head指向的空闲块的内容读取到内存，才能找到第二个空闲块的地址，需要多次的访问外存，速度慢

- **动态异长分区的分配适用于什么管理方式？有哪些分配与去配算法？各自有什么特点?**

▶ 这种存储分配形式常用于界地址存储管理方式与段式存储管理方式中

▶ 系统中需要一张表，称为空闲区域表，表中记录着所有当前未被进程占用的空闲区域

▶ 算法见下题

- **描述最先适应算法FF的实现机制与其优缺点**

▶ 最先适应算法是指对于存储申请命令，选取满足申请长度要求且起始地址最小的空闲区域。在实现时，可以将系统中所有的空闲区域按照起始地址由小到大的次序依次记录于空闲区域表中。当进程申请存储空间时，系统由表的头部开始查找，选取满足要求的第一个表目。如果该表目所对应的区域长度恰好与所申请的区域长度相同，则将该区域全部分配给申请者，否则将该区域分割为两部分，一部分的长度与所申请的长度相同，将其分配给申请者；另一部分的长度为原长度与分配长度之差，仍将其记录于空闲区域表中。

▶ 该算法的优点是尽量使用低地址空间，因而在高地址空间可能形成较大的空闲区域。对于一个较大空间的请求，可望在存储高地址空间得到满足。

▶ 该算法的缺点是可能分割较大的空闲区。

- **描述下次适应算法NF的实现机制与其优缺点**

▶ 下次适应算法自下次分配空闲区域的下一个位置开始，选取第一个可满足的空闲区域。在实现时，设置一个循环查找指针，开始时指向表头，每次分配从该指针处开始查找（如果找到表尾，则将指针置成表头）第一个遇到的可满足区域。分配后将指针调整为所分配区域的下一个区域。

▶ 该算法的优点是空闲区域分布和分配更均匀

▶ 缺点是可能分割大空间区域

- **描述最佳适应算法BF的实现机制和其优缺点**

▶ 最佳适应算法在分配时取满足申请要求且长度最小的空闲区域。在实现时，将系统中的所有空闲区域按照长度由小到大的次序依次记录在空闲区域表中，当进程申请存储空间时，系统由表的头部开始查找，选取满足条件的第一个表目

▶ 该算法的优点是尽量不分割大的空闲区域

▶ 缺点是可能会形成很小以至以后无法利用的空闲区域，即碎片

- **描述最坏适应算法WF的实现机制和其优缺点**

► 最坏适应算法在分配时选取满足申请要求且长度最大的空闲区域，在实现时，将系统中的所有空闲区域按照长度由大到小的次序依次记录在空闲区域表中，当进程申请存储空间时，系统由表的头部开始查找，选取满足条件的第一个表目

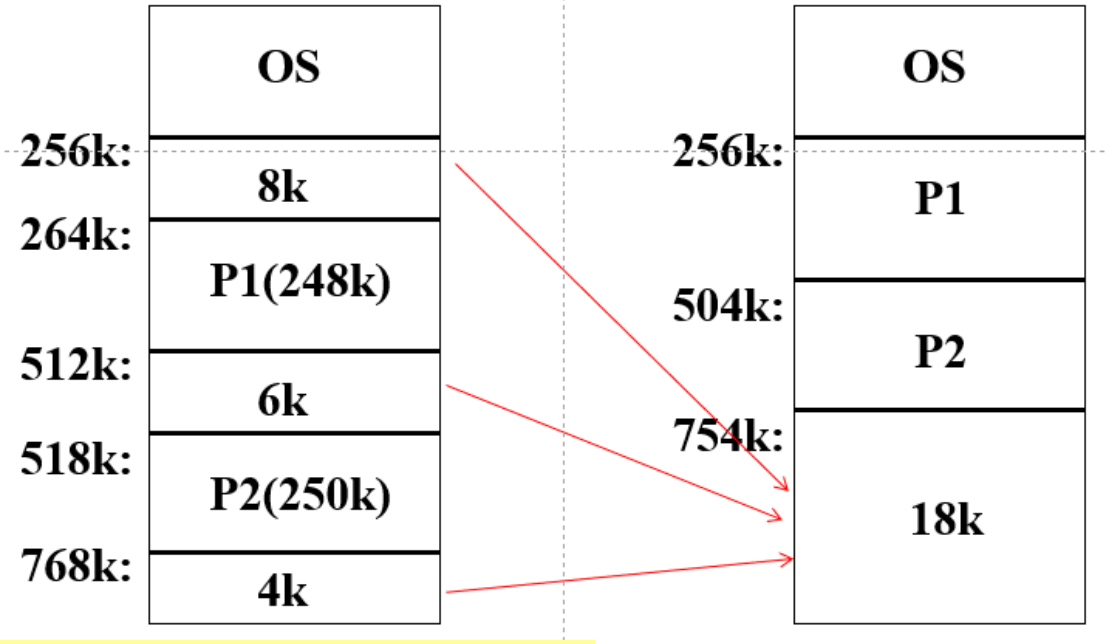
► 该算法的优点是可以避免形成碎片

► 缺点是分割大的空闲区域，当遇到较长存储申请命令时，无法满足要求的可能性较大

- **简述FF、NF、BF、WF算法之间的异同点**

► （根据上面的回答总结吧）

- **画图解释什么是紧凑（如何解决碎片问题）**



► **紧凑**：移动占用区域，使所有空闲区域连成一片

- **介绍一下单一连续区存储管理**

► 单一连续区（single contiguons region）存储管理又称单对界存储管理方式。 一个进程在内存空间的地址由两个参数决定：进程起始地址和长度，称为一个对界。

► 内存空间采用动态异长分区方法

► 一个进程空间由连续的区域构成。假设进程的长度为1，则其逻辑地址为0—l - 1 。

► 一个进程在内存中占有一个连续的区域，假设进程的长度为1，在内存中的起始地址为b，则其物理地址范围为b—b + l- 1 。

► 为实现存储分配和地址映射，需要如下的表目。

- ①内存分配表。用于记录内存中所有已经被分配的区域。该表有时可以省略，此时各个 分配区域分别登记在占有进程的进程控制块中。
- ②空闲区域表。用于记录内存中所有尚未分配的区域。

► 为了加快地址映射的速度，硬件应当提供以下两个寄存器。

- ①首址寄存器。此寄存器整个系统有一个，用于保存正在运行进程的起始地址。
- ②限长寄存器。此寄存器整个系统有一个，用于保存正在运行进程的长度。

- **画图描述单对界存储管理方式中地址映射的步骤**

1. 由程序确定逻辑地址a。
2. 由逻辑地址a与限长寄存器的内容l相比较以判断是否满足$0 \leq a \leqslant l-1$。如果不满足则为越界，发生越界中断。
3. 由逻辑地址a与首址寄存器的内容b相加得到物理地址。

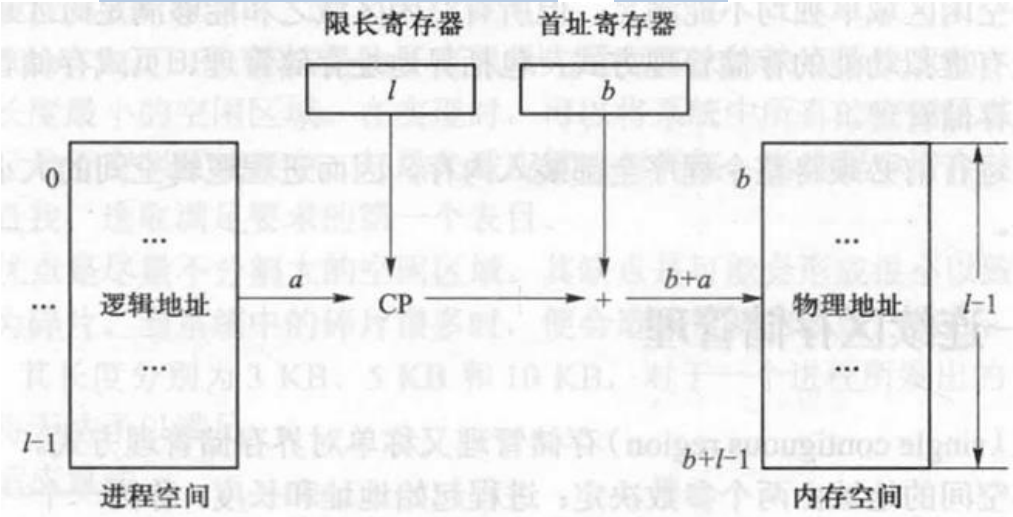


图 6-7　地址映射的关系

- **什么是双对界？和单对界的区别在哪里？如何实现双对界？**

► 在单一连续区存储管理中，一个进程在内存空间的地址由两个参数决定：进程起始地址和长度，称为一个对界，单对界限制一个进程在内存中只占有一个连续区域，双对界则允许一个进程在内存中占有两个连续的区域。这两个区域一个可用于保存代码，另一个可用于保存数据， 其中代码区域可以被多个进程所共享，数据区域则为进程所独享。

► 为实现双对界存储管理，硬件需要提供两对寄存器：代码区域首址寄存器和限长寄存器，数据区域首址寄存器和限长寄存器。取指令时硬件决定使用前一组寄存器，取数据时硬件决定使用后一组寄存器。

- **简述什么是内存的覆盖和交换技术？两者有什么区别？**

► 交换又称换入换出或者滚入滚出，是指进程在内存空间与外存空间之间的动态调度，它是缓解内存空间使用矛盾的一种有效方法。

► 覆盖是将较大程序装入较小进程空间的一种技术。其思想是只将全局代码和数据静态地放在内存中，其他部分则分阶段地动态装入，后装入的对象重复使用以前对象所占用的空间，即覆盖以前的对象，这些动态装入的成分也被称为覆盖。

► 两者的区别主要有：交换技术由操作系统自动完成，不需要用户参与，而覆盖技术需要专业的程序员给出作业各部分之间的覆盖结构，并清楚系统的存储结构；交换技术主要在不同作业之间进行，而覆盖技术主要在同一个作业内进行；另外覆盖技术主要在早期的操作系统中采用，而交换技术在现代操作系统中仍具有较强的生命力。

- **举例解释覆盖技术**

► 如图，一个包括4遍扫描的编译程序，运行时既可以对应4个进程，也可以通过覆盖由一 个进程完成。一

个进程完成4遍扫描，将符号表等公共数据作为共享成分为其静态分配空间， 扫描程序则动态装入，每次扫描结束时转覆盖驱动程序装入下一次扫描代码

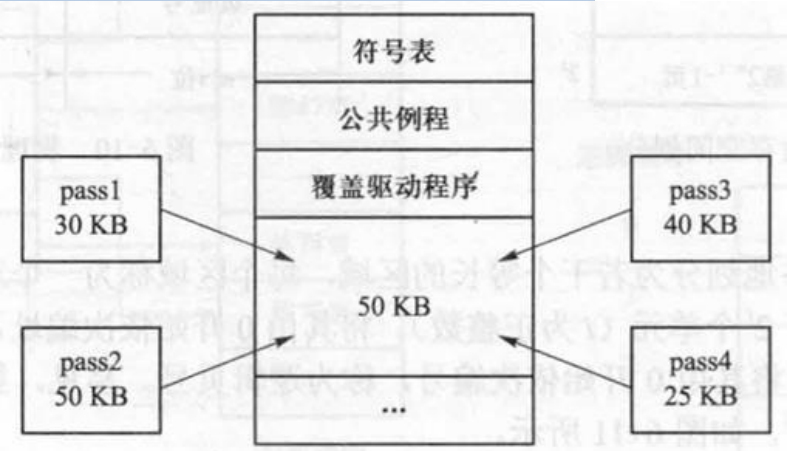
- ▶ 

图 6-8　覆盖的例子

• 介绍页式存储管理

- ▶ 页式（Paging，也称分页）存储管理方式允许一个进程占有内存空间中的多个连续区域，而且这些区域的长度相同，因而采用静态等长存储分配方法，不会产生碎片。
- ▶ 内存空间被静态地划分为若干个等长的区域，每个区域称为一个物理页框
- ▶ 每个页框通常有2^i个单元（i为正整数）。将其由0开始依次编址，称为页内地址
- ▶ 设内存的容量为$2^n B$,则共有个2^{n-i}页框。将所有页框由0开始依次编号,称为页框号。易见,第k个页框的起始地址(即首址)为$k * 2^i$
- ▶ 将页框号作为高位,页内地址作为低位,即可得到物理地址

进程空间也被静态地划分为若干个等长的区域，每个区域称为一个逻辑页面，其长度与页框的长度相同，即共有 2^i 个单元（i 为正整数）。将其由 0 开始依次编址，称为页内地址。设进程共有 l 个逻辑页面，将其由 0 开始依次编号，称为逻辑页号。易见，第 k 个逻辑页面的起始地址（即首址）为 $k\times2^i$，如图 6-11 所示。

一个进程单元的地址称为逻辑地址。显然，逻辑地址可以通过下式计算得到:

逻辑地址=逻辑页首址+页内地址

=逻辑页号×2^i+页内地址

实际上，将逻辑页号作为高位，页内地址作为低位，即可得到逻辑地址，如图 6-12 所示。

- ▶ 

图 6-11　进程空间划分

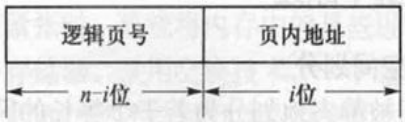


图 6-12　逻辑地址

- ▶ 进程的逻辑页面是连续的,但是这些逻辑页面所对应的页框却不一定连续。
- ▶ 为了实现逻辑地址与物理地址之间的转换,系统中需要以下两个表目。

- 页表。该表用于记录进程的逻辑页面与内存页框之间的对应关系。
- 总页表。该表用于记录页框的使用情况

► 所需寄存器：页表首址寄存器、页表长度寄存器、一组联想寄存器——快表

- **描述界地址存储管理方式与页式存储管理方式的异同点**

► 界地址存储管理方式要求一个进程占有内存空间中的一个连续区域，因而采用动态异长存储分配方法可能产生碎片。相反，页式存储管理方式允许一个进程占有内存空间中的多个连续区域，而且这些区域的长度相同，因而采用静态等长存储分配方法，不会产生碎片。

- **描述页式存储管理方式中地址映射的步骤**

► 地址映射需要将逻辑地址转换成物理地址，即完成如下映射关系：

► $\sigma:(p,d)\rightarrow(f,d)\cup\{\Omega\}$

► 其中，p为逻辑页号，d为页内地址，f为页框号。

► 地址映射的步骤如下。

1. 由指令产生逻辑地址（P，d）。
2. 由逻辑页号p查快表得到页框号f， 如果找不到，由逻辑页号p与页表长度寄存器中的内容l相比较以判断是否满足$0\leq p\leqslant l-1$。如果不满足则为越界，发生越界中断。 由逻辑页号p与页表首址寄存器中的内容b查页表得到页框号f。由页框号f与页内地址d合并得到物理地址（f,d）。 将（p，f）送入快表中。如果此时快表已满，则按置换算法淘汰一个。
3. 如果找到，由页框号f与页内地址d合并得到物理地址（f,d）。

- **在许多情况下，允许进程的逻辑页面不连续会带来更大的方便。例如，在多执行流进程中，每个线程都需要一个目态栈，要求栈在逻辑上连续且具有可伸缩性。该怎么实现?**

► 系统为每个线程分配充足的逻辑页面，但是在物理上只有真正用到的那些逻辑页面才有物理页框与之相对应。在实现时，在页表中增加一个“有效位”，当该位为1时表示有效，即有对应的页框；而当该位为0时表示无效，即尚未分配页框。对有效位为0的逻辑页面的访问将导致物理页框的动态分配。

- **画图举例解释多级页表**

► 对于长度为2^{32}的逻辑地址空间，页面长度定义为2^{12}，则一个进程最多可以拥有2^{20}个页面。也就是说，系统需要提供长度为2^{20}的连续页表存储区，这是很困难的。为此，常将页表分为多级进行存放，例如对于上述例子，将2^{20}分为2^{10}x2^{10}两级，一级称为外页表，另一级称为内页表，如图所示。此时，外页表和内页表的长度都在2^{10}以内。

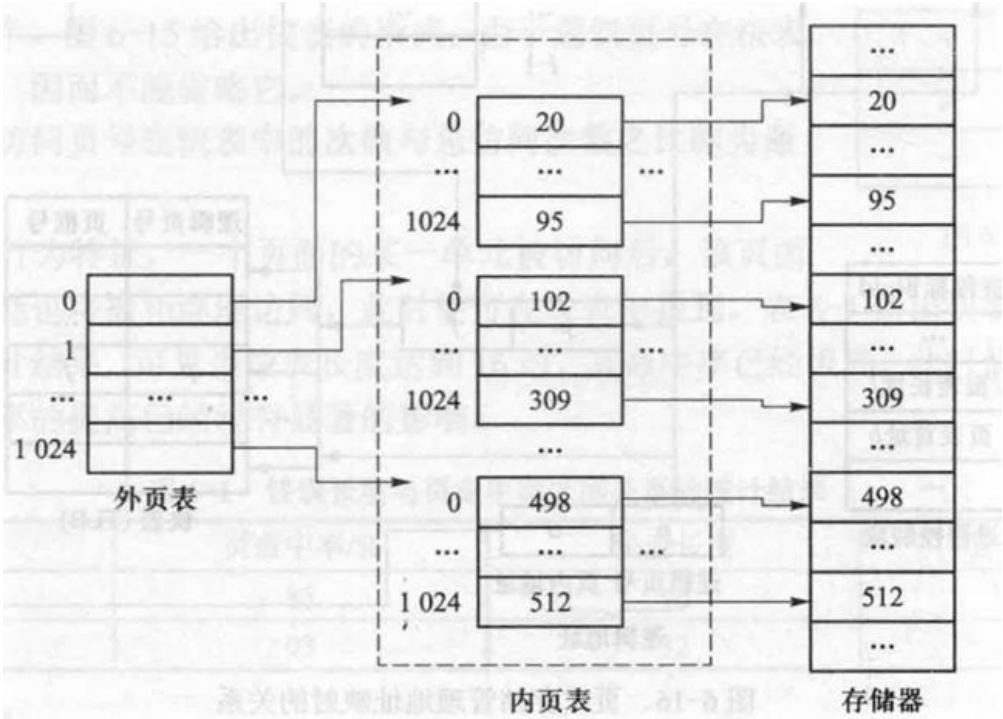


图 6-17　二级页表

- **什么是反置页表？他与传统页表的差别在哪里？有什么明显的问题？如何解决？**

▶ 传统页表是面向进程虚拟空间的，即对应进程的每个逻辑页面设置一个表项，当进程的地址空间很大时，页表需要占用很多存储空间，造成浪费。与经典页表不同，反置页表是面向内存物理页框的，即对应内存的每个物理构架设置一个表项，表项的序号就是物理页框号f，表项的内容则为进程标识Pid与逻辑页号p的有序对。系统只需设置一个反置页表，为映所有进程共用。

▶ 反置页表的一个明显问题是速度：对反置页表的顺序搜索需要多次访问内存，对于不存在的页甚至需要查到表尾。

▶ 为提高访问速度，采用杂凑（hash，也称散列、哈希）技术，在反置页表中增加冲突计数和空闲标志。在进行地址映射时，由hash（pid，p）计算得到反置页表入口地址，从该入口地址开始向下探查找到对应的表项，位移f为对应的页框号

- **简述页式存储管理方式和段式存储管理方式的异同点**

▶ 在页式存储管理中，分页对于用户是透明的，一个进程由若干个页构成，所有页的长度相同；在段式存储管理中，分段对于用户是可见的，一个进程由若干个段构成，各个段的长度可以不同，一个段恰好对应一个程序单位。

▶ 分页是出于系统管理的需要，分段是出于用户应用的需要。一条指令或一个操作数可能会跨越两个页的分界处，而不会跨越两个段的分界处。

▶ 页大小是系统固定的，而段大小则通常不固定。

▶ 逻辑地址表示：分页是一维的，各个模块在链接时必须组织成同一个地址空间；分段是二维的，各个模块在链接时可以每个段组织成一个地址空间。

▶ 通常段比页大，因而段表比页表短，可以缩短查找时间，提高访问速度。

▶ 都是为进程分配内存空间的存储管理方式

- **介绍段式存储管理**

► 内存空间被动态地划分为若干个长度各异的区域，每个区域称为一个物理段。每个物理段在内存中有一个起始地址，称为段首址。将物理段中的所有单元由0开始依次编址，称为段内地址。对于一个长度为m的段来说，其段内地址依次为0，1，…，，m-1 。内存中一个单元的位置被称为物理地址。物理地址可以通过公式计算得到

► 进程空间被静态地划分为若干个长度各异的区域，每个区域称为一个逻辑段。将一个逻辑段中的所有单元由0开始依次编址，称为段内地址。对于一个长度为n的段来说，其段内地址依次为0，1，…，n—1 。每个逻辑段具有一个符号名字，称为段名。将一个进程的所有逻辑段由0开始依次编号，称为段号。对于一个由l个逻辑段所构成的进程来说，其段号依次为0，1，…，l-1

► 通常在源程序中使用段名，而在编译（汇编）、连接后段名被转换为段号，因而操作系统以段号来区分进程的各个逻辑段。

► 源程序中的地址信息由两个部分构成，即段名和段内入口名。这是一个二维地址，称为程序地址。编译（汇编）后的地址信息也由两个部分构成，即段号和段内地址，这也 是一个二维地址，称为逻辑地址

► 进行地址映射后的地址信息只由单一部分构成，即段的起始地址（段首址）与段内地址之和，这是一个一维地址，即物理地址。

► 进程的一个逻辑段与内存的一个物理段相对应。一个进程的多个逻辑段可存放于内存中若干个不相连的物理段中。 在物理上，进程的段内地址是连续的，而段与段之间则可以不连续。

► 为了实现地址映射，系统中需要保存若干个表目，其中包括：

- ①段表。该表用于记录段号与段首址之间的关系。
- ②空闲表。用于记录并管理内存中的空闲区域。

► 为了加快地址映射的速度，硬件通常提供以下3组寄存器。

- ①段表首址寄存器。用于保存正在运行进程的段表的首址。
- ②段表长度寄存器。用于保存正在运行进程的段表的长度。
- ③一组联想寄存器（快表）。用于保存正在运行进程的段表中的部分项目，即当前正在访问的段所对应的项目。

- **描述段式存储管理方式中地址映射的过程**

► 地址映射需要将逻辑地址转换成物理地址，即完成以下映射关系：

► σ:（s,d）→（b'+d）$\bigcup\{\Omega\}$

► 其中，s为段号，d为段内地址，b ’ 为段的起始地址。当逻辑地址中的s或d越界时，地址映射没有意义，结果用{Ω}表示

► 地址映射的步骤如下。

- ①由指令产生逻辑地址（s，d）。
- ②由段号s查找快表得到段首址b'和段长l'
- 如果找不到：
  - 将段号s与段表长度寄存器的内容l相比较以判断是否满足$0 \le s \le l-1$ 。如果不

满足， 则为越界，发生越界中断。

- 由段号s与段表首址寄存器的内容b查找内存段表得到段首址和段长。
- 由段首址b ’ 和段内地址d相加得到物理地址b'+d 。 将（s , b ' , l '）送入快表。如果此时快表已满，则按照置换算法淘汰一个。

- ③将段内地址与段长相比较以判断是否满足$0 \leq d \leq l-1$ 。如果不满足，则为越界，发生越界中断。
- ④由段首址b和段内地址d相加得到物理地址b'+d

• 简述什么是段的共享

► 一个进程的段号是连续的，而段与段之间却不一定连续。如果某一进程的一个段号s_i，与另一进程的一个段号s_j对应同一段首址和段长，即可实现段的共享

• 存储管理的基本任务是什么?

► （1）管理内存空间;
► （2）进行虚拟地址（或：逻辑地址）到物理地址的转换;
► （3）实现内存的逻辑扩充;
► （4）完成内存信息的共享和保护。

• 存储管理的主要功能是什么?

► 存储管理的主要功能是解决多道作业的主存空间的分配问题。主要包括：

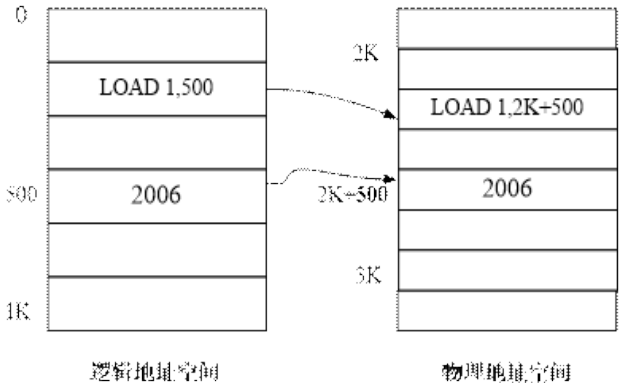
- （1）内存区域的分配和管理：设计内存的分配结构和调入策略，保证分配和回收。
- （2）内存的扩充技术：使用虚拟存储或自动覆盖技术提供比实际内存更大的空间。
- （3）内存的共享和保护技术：除了被允许共享的部分之外，作业之间不能产生干扰和破坏，须对内存中的数据实施保护。

• 为什么分段技术比分页技术更容易实现程序或数据的共享?

► 每一段在逻辑上是相对完整的一组信息，分段技术中共享信息是在段一级出现的。因此，任何共享的信息可以单独作一个段，同样段中所有内容就可以用相同的方式进行使用，从而规定相同的使用权限;
► 而页是信息的物理单位，在一个页面中可能存在逻辑上互相独立的两组或更多组信息，都各有不同的使用方式和存取权限。
► 因此，分段技术较分页技术易于实现程序或数据的共享。

• 举例说明存储管理中地址重定位的概念

► 如图-1所示，作业J的逻辑地址空间是0到1KB，而分配给该作业物理存储空间是2KB到3KB。图中的指令“LOAD 1,500”装入内存时，必须对相应的地址进行变换，实际执行的指令变换为“LOAD 1,500+2K”。



- **考虑在下述存储管理方式中,进程空间和逻辑空间的编址情况。**
  (1)界地址存储管理方式,进程空间的首地址。
  (2)页式存储管理方式,进程空间的首地址。
  (3)段式存储管理方式,进程空间各段的首地址。
  (4)段页式存储管理方式,进程空间各段的起始地址。

► (1)界地址存储管理方式,进程空间的首地址从0开始编址;
► (2)页式存储管理,进程空间的首地址从0开始编址;
► (3)段式存储管理,进程空间各段的首地址从0开始编址;
► (4)段页式存储管理,进程空间各段的起始地址从0开始编址。

- **页式、段式、段页式存储管理方式各有哪些针对越界的检查?**

► (1)页式:逻辑地址的形式为(p,d),其中逻辑页号p可能越界,意味着访问不存在的页面,检查要求为0≤p≤l-1,其中l为页表长度寄存器的内容;
► (2)段式:逻辑地址的形式为(s,d),s和d都可能越界,s越界意味着访问不存在的段,d越界意味着访问段内不存在的单元。检查要求为0≤s≤l-1,0≤d≤l'-1,其中l为段表长度寄存器的内容,l'为段表或快表中段长的内容;
► (3)段页式:逻辑地址的形式为(s,p,d),其中段号s和段内逻辑页号p可能越界,s越界意味着访问不存在的段,p越界意味着访问段内不存在的页,检查要求为0≤s≤l-1,0≤p≤l'-1,其中l为段表长度寄存器的内容,l'为段表中页表长度的内容。

- **对于以下存储管理方式来说,进程的逻辑地址形式如何?其进程地址空间各是多少维的?**
  (1)页式;(2)段式;(3)段页式。

► (1)页式存储管理中,进程地址空间是一维的;
► (2)段式存储管理中,进程地址空间是二维的;
► (3)段页式存储管理中,进程地址空间是二维的。

- **在页式存储管理中,页的划分对用户是否可见?在段式存储管理中,段的划分对用户是否可见?在段页式存储管理中,段的划分对用户是否可见?段内页的划分对用户是否可见?**

- (1)在页式存储管理中,分页对于用户是透明的,一个进程由若干个页构成,所有页的长度相同;
- (2)在段式存储管理中,分段对于用户是可见的,一个进程由若干个段构成,各个段的长度可以不同,一个段恰好对应一个程序单位;
- (3)在段页式存储管理中,段的划分对用户是可见的,段内页的划分对用户是透明的,一个段由若干个页构成,所有页的长度相同。

• 为何引入多级页表?多级页表是否影响指令执行速度?

- 早期的内存空间和进程空间都比较小,一个进程的页表长度也较小,可以在内存中分配一个连续的存储区域保存进程的页表。但随着内存空间和进程空间的快速增长,页表越来越大,单级页表的存放遇到困难。例如,对于长度为232的逻辑地址空间,页面长度定义为212(4KB),则一个进程最多可以拥有220个页面,也就是说,系统需要提供长度为220的连续页表存储区,这是困难的。因此常将页表分为多级进行存放。例如,对于上述例子,将220分为210×21°两级,一级称为外页表,另一级称为内页表,此时外页表和内页表的长度都在210以内。
- 可以很自然地把二级页表扩展为多级页表。显然,多级页表会降低地址映射的速度,但通过快表仍可以将效率保持在合理的范畴内。例如,对于四级页表,假定快表的命中率为98%,快表与内存的访问时间分别为20 ns和100 ns,有效访问时间为EAT=98%x(20+100)+2%x(20+500)=128(ns)，额外的开销是128-100=28 ns。

• 与传统页表相比,倒置页表有什么优势?

- 传统页表是面向进程虚拟空间的,即对应进程的每个逻辑页面设置一个表项,当进程的虚拟地址空间很大时,页表需占用很多的存储空间,造成浪费。与经典页表不同,倒置页表是面向内存物理页框的,即对应内存的每个物理页框设置一个表项,表项的序号就是物理页框号f,表项的内容则为进程标识pid与逻辑页号p的有序对。系统只需设置一个倒置页表,为所有进程所共用。

• 允许进程空间的逻辑页号不连续所带来的好处是什么?

- 可以给同一进程内的多个线程预留足够的堆栈空间,而又不浪费实际的内存页框。

• 试比较段式存储管理与页式存储管理的优点和缺点。

- 页式存储管理的优缺点是:
  - (1)静态等长存储分配简单,有效地解决了内存碎片问题;
  - (2)共享和保护不够方便。
- 段式存储管理的优缺点是:
  - (1)动态异长存储分配复杂,存在碎片问题;
  - (2)共享与保护方便;
  - (3)可以实现动态连接和动态扩展。

• 试举例说明段长动态增长的实际意义。

- 允许段长动态增长对于那些需要不断增加或改变新数据或子程序的段来说很有好处。
- 例如,分配给进程的栈空间大小通常预先无法准确估计,若分配过少可能不够用,分配过多则会

造成浪费。在栈可以动态增长的情况下,系统开始可以为进程分配一个基本长度的栈空间,这个长度浪费很小。若进程运行时发生栈溢出,通过中断可以进行动态扩展。

- **在段式存储管理中,段的长度可否大于内存的长度?在段页式存储管理中又如何呢?**

► 在段式存储管理中,段的长度不能大于内存的长度,因为一个独立的段占用一段连续的内存空间,内存分配是以段为单位进行的,如果一个段的长度大于内存的长度,那么该段将无法调入内存。在段页式存储管理中,段的长度可以大于内存的长度,因为内存分配的单位是页,一个段内逻辑上连续的页面,可以分配到不连续的内存页面中,不要求一个段的所有逻辑页面都进入内存。

- **共享段表的用途何在?**

► 共享段表的用途主要有如下两个:
  - (1)用来寻找共享段:通过根据进程首次访问某段的名称在共享段表中查找,可以得知该段是否已在内存中;
  - (2)确保一个共享段只有一组描述信息:共享段的地址、长度等信息在共享段表中仅记录一次,防止在多个进程段表中重复登记所带来的维护困难。

- **具有二级页表的页式存储管理与段页式存储管理有何差别?**

► 具有二级页表的页式存储管理的地址空间依然是一维的,二级页的划分对于进程来说都是透明的。而段页式存储管理的地址空间是二维的,段的划分用户能感觉到。

- **何谓请调?何谓预调?为何在预调系统中必须辅以请调?**

► 所谓**请调**是当页故障发生时进行调度,即当被访问页面不在内存时由动态调页系统将其调入内存。显然,采用纯请调策略,被移入内存的页面一定会被用到,即不会发生无意义的页面调度。但是,请调也有一个缺点:
  - 从缺页中断发生到所需页面被调入内存,这期间对应的进程必须等待,这样将会影响进程的推进速度。

► **预调**也称先行调度。是在页故障发生前进行调度,即在一个页面即将被访问之前就由动态调页系统将其调入内存。显而易见,预调可以节省进程因页故障而等待的时间。预调通常根据程序的顺序行为特性而作出:如某进程当前正访问第12页,则接下来很可能会访问第13页、第14页,此时可将第13页甚至第14页预先调入内存。这样当该进程访问第13页以至第14页时它们已经在内存中,不会发生缺页故障,从而提高进程的推进速度。

► 预调不一定是百分之百准确的。由于程序中存在转移语句,第13页用完后可能需要访问第10页,而该页目前可能不在内存中。也就是说,预先调入的页面可能有的未被用到,预先未调入的页面可能有的会被用到。当访问到预先未调入内存的页面时,仍会发生缺页中断。因而,在采用预调的系统中,必须辅以请调的功能。

- **解释下列与存储管理有关的名词:**

► 地址空间与存储空间

- 目标程序所在的空间称为地址空间，即程序员用来访问信息所用的一系列地址单元的集合；存储空间是指主存中一系列存储信息的物理单元的集合。

- 逻辑地址与物理地址
  - 在具有地址变换机构的计算机中，允许程序中编排的地址和信息实际存放在内存中的地址有所不同。逻辑地址是指用户程序经编译后，每个目标模块以0为基地址进行的顺序编址。逻辑地址又称相对地址。物理地址是指内存中各物理存储单元的地址从统一的基地址进行的顺序编址。物理地址又称绝对地址，它是数据在内存中的实际存储地址。

- 虚地址与实地址
  - 虚地址同逻辑地址，实地址同物理地址。

- 地址重定位
  - **重定位**是把逻辑地址转变为内存的物理地址的过程。根据重定位时机的不同，又分为静态重定位（装入内存时重定位）和动态重定位（程序执行时重定位）。

第七章 虚拟存储系统

2024年6月21日 13:24

- **内存空间管理方式存在什么问题？有什么缺点？**

▶ 问题：当一个参与并发执行的进程运行时，其整个程序必须都在内存中

▶ 缺点：假若一个进程的程序比内存可用空间还大，则该程序无法运行；由于程序的局部特性，一个进程在运行的任意一个阶段只需使用所占存储空间的一部分，这样暂未用到的内存区域便会被浪费。

- **简述虚拟存储管理方式能够工作基于的理论**

▶ 虚拟存储管理之所以能够工作，是基于局部性原理的，包括时间上的局部性和空间上的局部性。

▶ 时间上的局部性：一条指令的执行与该指令的下一次执行，以及一个数据的一次访问与该数据的下一次访问，在时间上是相对集中的。产生时间局部性的典型原因是程序中存在大量循环操作。

▶ 空间上的局部性：程序中相邻的指令与相邻的数据在执行时通常会被集中用到。产生空间局部性的典型原因是程序的顺序执行和对数组的访问。

- **在虚拟存储管理系统中，对内存页框的分配有哪些策略？**

▶ 平均分配

▶ 按进程程序长度的比例分配

▶ 按进程优先级别的比例分配

▶ 按进程程序长度和优先级别的比例分配

- **为了实现虚拟页式存储管理，需要分别对页表和快表做什么样的修改？**

▶ 在页表中需要增加以下栏目：外存块号、内外标志、修改标志

▶ 快表增加“修改标志”一栏

- **在虚拟存储管理系统中，有哪些页面调入策略？各自的实现机制和优缺点是什么？**

▶ 请调：所谓请调，是当页故障发生时进行调度，即当访问某一页面而该页面不在内存时由动态调页系统将其调入内存。采用纯请调策略，被移入内存的页而一定会被用到，即不会发生无意义的页面调度。但是，请调方式也有一个缺点，从缺页中断发生到所需页面被调入内存，这期间对应的进程需要等待，如此将会影响进程的推进速度。

▶ 预调：预调又称先行调度，是在页故障发生之前进行调度，即当一个页面即将被访问之前就由动态调页系统将其调入内存。预调可以节省进程因页故障而等待的时间，可以降低页故障率，但是预调不一定是百分之百准确的，并且实现开销比较大。

- **在虚拟存储管理系统中，对外存块的分配有哪些策略？各自如何实现？有什么优缺点？**

- 静态分配：一个进程在运行之前，将其所有页面全部装入外存储器。当某一外存页面被调入内存时， 所占用的外存页面并不释放。这样，当该页面以后被淘汰时，如果它在内存期间未被修改过， 则不必写回外存储器，因为外存储器中有一个与它完全相同的副本。这样可以减少因页面调度而引起的系统开销，其代价是牺牲一定的外存空间。
- 动态分配：一个进程在运行之前，仅将未装入内存的那部分页面装入外存储器。当某一外存页面被调入内存时，将释放所占用的外存页面。这样，当该页面以后被淘汰时，无论它在内存中是否曾经被修改过，都必须重新为其申请外存页面，然后将该页写回外存储器。这样可以节省外存空间，但是会增加由于页面调度而引起的系统开销。

• 什么是写时复制？有什么优点?

- 现今版本fork ()在实现时，并不立即为子进程分配地址空间，而是共用其父进程的地址空间，当子进程或父进程中的任何一个对其地址空间的某个页面执行写操作时，才为子进程分配独立的页框，并进行复制，其后对这个页面来说，父子进程具有不同的页框，该技术被称为写时复制

• 简述虚拟页式存储管理的基本原理，描述具体步骤

- 在虚拟页式存储管理系统中，进程运行之前，部分页面被装入内存，另一部分页面（或者全部页面）被装入外存储器。在进程运行过程中，如果所访问的页面在内存，则与无虚拟的情形相同；如果所访问的页面不在内存，则发生缺页故障，进入操作系统，由操作系统进行页面的动态调度。
- 具体步骤如下：

1. 找到被访问页面在外存储器中的地址。
2. 在内存中寻找一个空闲页框。如果没有，按照淘汰算法选择一个内存页面，将此内存页面写回外存储器，修改页表及页面分配表。
3. 读入所需的页面，修改页表及页框分配表。
4. 启动进程，重新执行被中断的指令。

• 说明局部置换与全局置换的区别

- 局部置换在当前缺页进程的页框集合中选择淘汰页面，全局置换在内存的所有页框集合中选择淘汰页面。

• 有哪些常见的置换算法？简述各自的思想

- 最佳算法OPT：淘汰以后不再需要的或者在最长的时间以后才会用到的页面。
- 先进先出算法FIFO：淘汰最先进入内存的页面
- 最近最少使用LRU算法：淘汰最后一次访问时间距离当前时间间隔最长的页面。
- 最近不用的先淘汰算法NUR：淘汰最近一段时间内未用过的页面。在实现时，为每一个页面增加两个硬件位，引用位和修改位，每隔固定的一段时间将所有引用位都清零。当要淘汰某一页面时，按照下面的次序进行选择。
    - ①引用位＝0，修改位=0：直接淘汰。

- ②引用位=0，修改位=1：淘汰之前写回外存储器。
- ③引用位=1，修改位=0：直接淘汰。
- ④引用位=1，修改位=1：淘汰之前写回外存储器。

► 最不经常使用算法LFU：淘汰访问次数最少的页面。在实现时，为每一个页面设置一个访问次数计数器。当一个页而由外存储器移到内存时，对应的计数器清零。当一个页面被访问时，对应的计数器值加1 。当需要淘汰时，取计数器值最小的页面。

► 最频繁使用算法MRU：与最不经常使用算法完全相反，最频繁使用算法认为计数器值最小的页面很可能刚刚调入内存，正待被使用，因而淘汰时取计数器值最大的。

► 二次机会算法：思想是淘汰装入最久且最近未被访问的页面。在实现时， 可以采用拉链数据结构

► 时钟算法：将页而组织成环状，有一个指针指向当前位置。每次需要淘汰页面时， 从指针所指的页面开始检查。如果当前页面的引用位为0，即从上次检测到目前为止，该页没有被访问过，则将该页替换。如果当前页面的引用位为1，则将其清零，并顺时针一移动指针到下一个位置。重复上述步骤，直至找到一个引用位为0的页面。

- **何谓系统的“抖动”现象？产生的原因是什么？当系统发生“ 抖动”时，你认为应该采取什么措施来加以克服？ （抖动就是颠簸）**

► “抖动”，是指页面在内存与外存储器之间频繁地调度，以致系统用于调度页面的时间比进程实际运行所占用的时间还要长。

► ”抖动“是由于页故障率过高而引起的，导致页故障率高的原因主要有以下3点：

1. 分给进程的页框数过少。
2. 页面置换算法不合理。
3. 程序结构。滥用的转移指令、分散的全局变量都会破坏程序的局部性，从而增高页故障率，甚至引起颠簸。

► 根据产生”抖动“的原因，可以采取以下几种处理方法：

1. 增加分配给进程的页框面数。
2. 改进页面置换算法。
3. 改进程序结构。采用结构化程序设计方法，尽量避免使用goto语句，根据大型矩阵的存放方式设计相应的访问操作。

- **在某些虚拟页式存储管理系统中,内存中永远保持一个空闲页框,这样做有什么好处?**

► 在内存没有空闲页框的情况下,需要按照置换算法淘汰一个内存页框,然后读入所缺页面,缺页进程一般需要等待两次I/O传输。

► 若内存总保持一个空闲页框,当发生页故障时,所缺页面可以被立即调入内存,缺页进程只需等待一次I/O传输时间。读入后立即淘汰一个内存页框,此时可能也需执行一次I/O传输,但对缺页进程来说不需等待,因而提高了响应速度。

- **什么是交换?**

► 当内存中进程较多时，缺页可能频繁发生，引起颠簸。为了提高系统性能，当内存资源紧张时，可以选择将一个进程全部交换到外存（称为换出），所占有的页框分给其他进程使用，待以后适当时候再交换到内存（称为换入）

- **什么是非均匀存储器访问?**

► 一般在多CPU系统中，每个CPU有自己局部存储器，访问局部内存的速度明显比访问非局部内存要快，这就是非均匀存储器访问（NUMA）的含义

- **简述虚拟段式存储管理的基本原理，描述具体步骤**

- 在虚拟段式存储管理系统中，进程运行之前，部分段被装入内存，另一部分段（或者全部段）被装入外存储器。在进程运行过程中，如果所访问的段在内存，则与无虚拟的情形相同；如果所访问的段不在内存，则发生缺段故障，进入操作系统，由操作系统进行段的动态调度。
- 具体步骤如下：

1. 找到被访问段在外存储器中的地址。
2. 在内存中寻找一个空闲区。如果内存没有足够的空间，可以采取两种方法：紧凑，即将内存中的所有空闲区合并； 淘汰，即将内存中的某段移至外存储器，段的淘汰可以采用与页式存储管理相似的算法。
3. 读入所需的段，修改段表。
4. 启动进程，重新执行被中断的指令。

- **解释页故障率反馈模型的思想以及具体实现**

► 颠簸是由于页故障率高而引起的，系统可以利用页故障率的反馈信息来动态地调整页面的分配，这就是页故障率反馈模型的思想。

► 具体地说，可以规定一个页故障率的上界和下界。当运行进程的页故障率高于上界时，表明分给它的内存页面数过少，应当增加；反之，当运行进程的页故障率低于下界时，表明分给它的内存页面数过多，可以减少。这样，根据页故障率动态调整页面的分配，就可以防止颠簸的发生。

- **什么是段的动态连接?**

► 所谓动态连接是指在程序运行过程中需要某一段时才将该段连接上，该连接是由操作系统来完成的。

► 在动态连接中，一个程序共有多少个段是不确定的，因而段名到段号的转换需要由操作系统来完成。由于对于同一个段只应分配一个段号，操作系统需要为每一个进程保持一个用于记录当前已连接段的表目，该表称为段名一段号对照表

► 此外，为了将外段中的一个符号名转换为对应的段内地址，每个段在编译（汇编）时都需要生成一个符号表，它被放在一个段的前面，作为段的组成部分。

- **动态连接有什么问题？如何解决?**

► 动态连接提高了系统的效率，但是与此同时也带来一些问题，主要是对于段共享的影响。代码段共享的必要条件是该段在运行过程中不修改自身，即要求是“纯代码”，而动态连接

需要修改连接字，这与段共享的要求相矛盾。

- 解决这个问题的一种方法是将代码段分为“纯段”和 “杂段”两部分，即将连接字等可修改的内容存放在“杂段”中，而将其他内容放在“纯段”中。“杂段”不共享，“纯段”可以共享。

为实现分页式虚拟存贮，页表中至少应含有哪些内容?

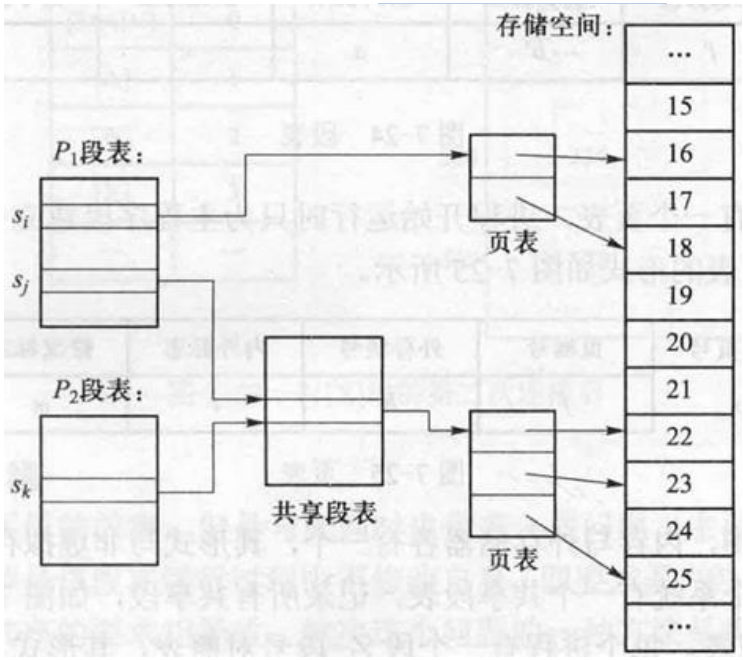
- 页号
- 标志
- 主存块号
- 磁盘上的位置

为什么要引入动态重定位？如何实现?

- 程序放在不连续的实际物理空间中，要进行逻辑地址到物理地址的转换，实现动态重定位
- 一般需要页式存储管理，页式存储管理用的不是寄存器，使用的是称为page talble的数据结构 page table记录了所有逻辑地址到物理地址的转换信息，进城切换的时候需要冲洗硬件上的page table

介绍虚拟段页式存储管理所需表目以及他们之间的联系

- ①段表。每个进程有一个段表，该表的长度动态变化，即每连接一个新的段时，增加一个新的项目。
- ②页表。每个段有一个页表，进程开始运行时只为主程序段建立一个页表，其他段的页表在段连接时建立。
- ③总页表。即位图，内存与外存储器各有一个，其形式与非虚拟存储管理相同。
- ④共享段表。整个系统有一个共享段表，记录所有共享段
- ⑤段名一段号对照表。每个进程有一个段名一段号对照表，其形式与非虚拟存储管理方式完全相同。
- 表间联系：
  - 对于非共享段来说，段表指向页表；对于共享段来说，段表指向共享段表，共享段表指向页表
  - 由于每段由若干页构成，所以对应每个段有一张页表



- **介绍虚拟段页式存储管理所需表目**

► ①段表首址寄存器b 。整个系统有一个，保存正在运行进程的段表首址。

► ②段表长度寄存器l 。整个系统有一个，保存正在运行进程的段表长度。它在进程运行过程中可能动态变化，即每连接一个新的段时其值加1 。

► ③快表。每个进程有一个

- **虚拟段页式存储管理地址映射流程**

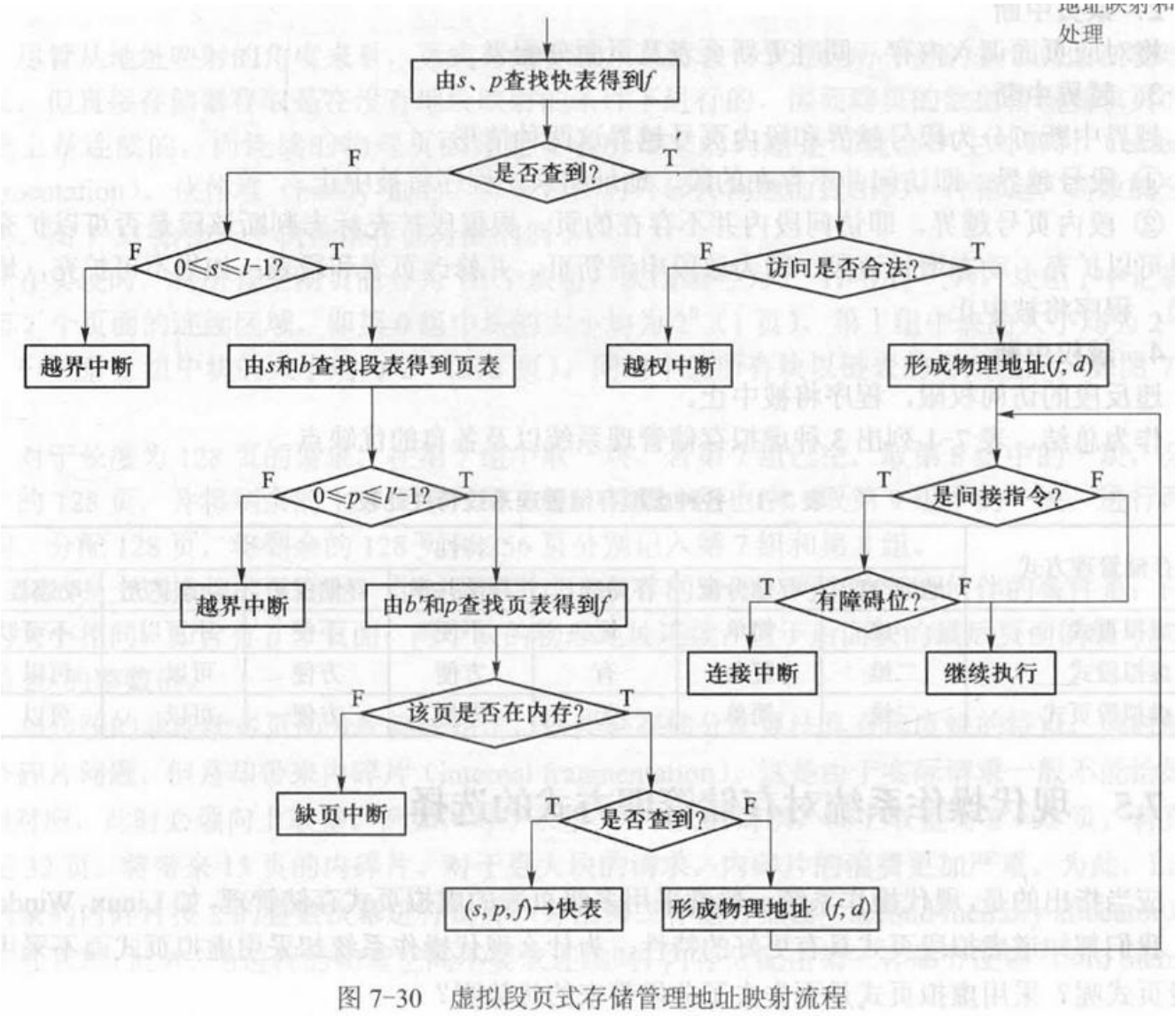


图 7-30　虚拟段页式存储管理地址映射流程

- **存储系统中有哪些中断处理的类型?**

► 连接中断

- 由段名查找本进程的段名一段号对照表及共享段表，经判断可以分为以下3种情形。
  - ①所有进程均未连接过（共享段表、段名一段号对照表均无）。
  - ②其他进程连接过但是本进程尚未连接过（共享段表有，段名一段号对照表无）。
  - ③本进程已经连接过（共享段表有或无，段名一段号对照表有）。

► 缺页中断

- 将对应页面调入内存，同时更新页表及页面分配表。

► 越界中断

- 越界中断可分为段号越界和段内页号越界这两种情形。
- ①段号越界。即访问并不存在的段，地址错误，程序将被中止。
- ②段内页号越界。即访问段内并不存在的页，根据段扩充标志判断该段是否可以扩充， 如果可以扩充，动态增长该段，即为该段申请新页，并修改页表和段表；如果不可扩充，地址错误，程序将被中止。

► 越权中断

- 违反段的访问权限，程序将被中止。

- **在虚拟段页式存储管理中,考虑段的共享与段长度的动态变化,如何处理连接中断?**

► 由段名查本进程的段名一段号对照表及共享段表,经判断可分为如下3种情形。

► (1)所有进程都未连接过(共享段表、段名一段号对照表均无):

- 查文件目录找到该段;
- 为该段建立页表,将该段从文件中全部读入swap空间,部分读入内存,填写页表;
- 为该段分配段号,填写段名一段号对照表;
- 如该段可共享,填写共享段表,共享计数置1;
- 填写段表;
- 根据段号及段内地址形成无障碍指示位的一般间接地址。

► (2)其他进程连接过但本进程未连接过(共享段表有,段名一段号对照表无):

- 为该段分配段号;
- 填写段名一段号对照表,填写段表(指向共享段表),共享段表中共享计数加1;
- 根据段号及段内地址形成无障碍指示位的一般间接地址。

► (3)本进程已连接过(共享段表无,段名一段号对照表有):

- 根据段号及段内地址形成无障碍指示位的一般间接地址。

► 这里,段内地址由两部分构成,即逻辑页号和页内地址。

- **试说明二次机会算法和时钟算法是必定终止的。**

► (1)二次机会算法:

- 淘汰调入内存最久且最近未使用的页面。若在线性链表中有标记为0的页面,则扫描一次链表即可找到一个淘汰页面。若所有标记均为1,扫描到页面时将标记改为0并移到链表末尾,下次扫描到该页面时发现标记为0即可淘汰,因而算法必定终止。

► (2)时钟算法:

- 改进之前的时钟算法与二次机会算法效果相同,只是所采用的数据结构不同。考虑改进后的时钟算法:若存在r=0且m=0的页面,按时钟扫描次序选择第一个满足要求的页面作为淘汰页面,算法终止于第一次步骤1。若1失败,但存在r=0且m=1的页面,按时钟扫描次序选择第一个满足要求的页面作为淘汰页面,算法终止于步骤2。若1和2都失败,在2中已将所有r都清0,若存在m=0,则在第二次执行步骤1时定能找到r=0且m=1的页面;若所有m=1,则算法在第二次执行步骤2时必定终止,因为所有r=0且m=1成立。综上所述,算法必定终止。

- **Denning工作集模型的理论依据是什么?**

► 程序的局部性。程序的执行是由一个局部迁移至另外一个局部的过程。**工作集**(working set)是进程在一段时间之内活跃访问页面的集合。Denning认为,为使程序有效地运行,它的工作集页面必须能够存放到内存中。

第八章 文件系统

2024年6月25日 4:00

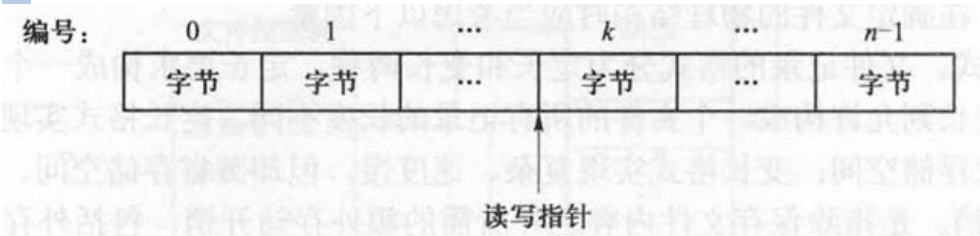
- **文件的逻辑组织形式主要有哪两种？各自有什么特点？**
  - 流式
    - 非结构式的
    - 操作系统对于文件的外部结构没有解释
    - 构成流式文件的基本单位是字节
    - 流式文件是具有符号名且在逻辑上有完整意义的字节序列
    - 用户对于流式文件的访问是以字节作为基本单位的
    - 每个文件的内部都有一个读写指针， 通过系统调用命令可以将该读写指针固定到文件的某一位置，以后的读写系统调用命令将从该指针所确定的位置处开始执行。
    - 虽然流式文件没有结构，但是用户在使用流式文件时可以自定义文件的结构。例如，假设一个学生的档案需要占用50B，则可将50B作为一条记录。
    - 

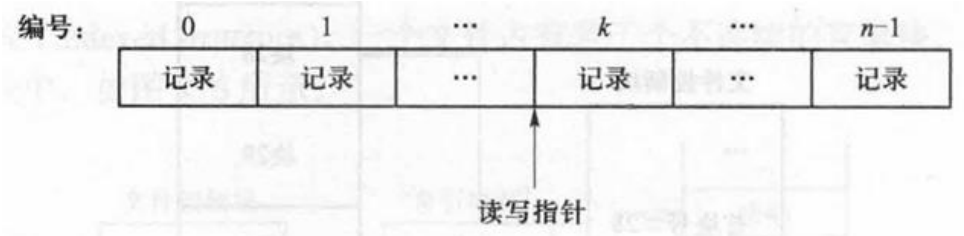
      图 8-2 流式文件
  - 记录式
    - 结构式的
    - 操作系统对于文件的外部结构有解释
    - 构成记录式文件的基本单位是记录
    - 记录式文件是具有符号名且在逻辑上有完整意义的记录序列
    - 一条记录由一组在逻辑上相关的信息项所构成。
    - 用户对于记录式文件的访问是以记录作为基本单位的
    - 每个文件的内部都有一个读写指针，通过系统调用命令可以将该读写指针移动到文件的某一位置，以后的读写系统调用命令将从该指针所确定的位置处开始执行。
    - 流式文件是记录式文件的特例。
    - 

      图 8-3 记录式文件
- **文件的物理组织有哪些形式？各自有什么特点？**

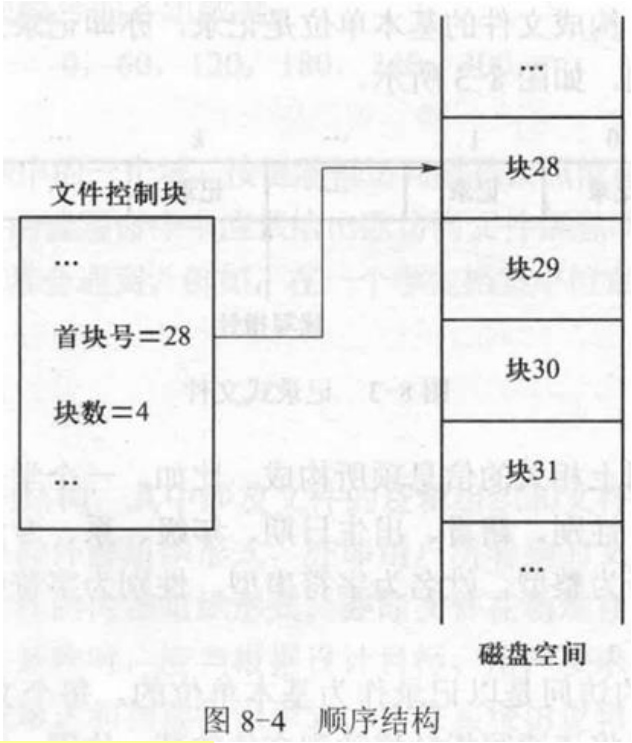
- 用于保存文件的物理设备是被划分为块的
- 文件的物理结构就是要确定如何将记录或者字节保存在存储型设备的物理块中。
- **在确定文件的物理结构时应当考虑以下因素**：
  - 记录格式：定长和变长
  - 空间开销：指除保存文件内容之外所需的额外存储开销
  - 存取速度：顺序存取的速度、按号随机存取的速度以及按键随机存取的速度
  - 长度变化：文件长度的动态增加和动态减少
- 顺序结构（连续结构）
  - 一个文件占有若干个连续的物理块
  - 其首块号和块数被记录在文件控制块中
  - 优点：访问速度快
  - 缺点：文件长度增加起来比较困难
  - 

    图 8-4　顺序结构
- 链接结构（串联结构）
  - 一个文件占有若干个不连续的存储块，各块之间以指针相连
  - 首块号和块数被记录于该文件的文件控制块中
  - 优点：长度易于动态变化
  - 缺点：随机访问的速度很慢
    - 尤其是当文件比较长时，因为若欲随机地访问某一块，就必须将此块之前的所有块都逐个地读入内存，这需要进行大量的输入输出操作。
- 索引结构
  - 一个文件占有若干个不连续的存储块，这些块的块号被记录于一个索引块中
  - 优点：访问速度快，长度变化容易
  - 缺点：系统存储开销大，因为每个文件都有一个索引块，而索引块也由存储块所构成，故需要占用额外的外存空间。更主要的是，当文件被打开时，索引块需要读入内存，否则访问速度会降低一半，故 需占用额外的内存空间。此外，由于构成索引块的存储块的长度固定，从而限制了文件的最大长度
- 散列结构（杂凑结构）

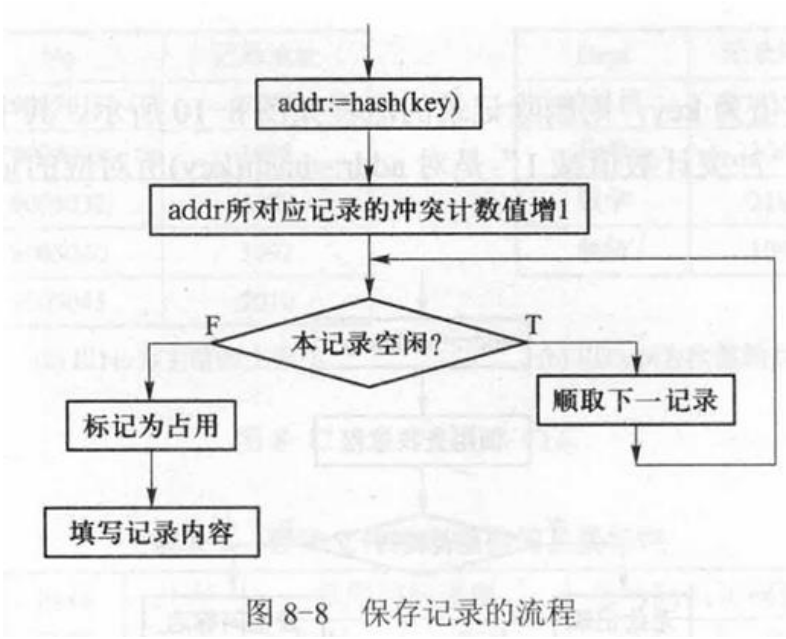
- 只适用于定长记录和按键随机查找的访问方式，常用于构造文件目录
- 保存记录
  - 

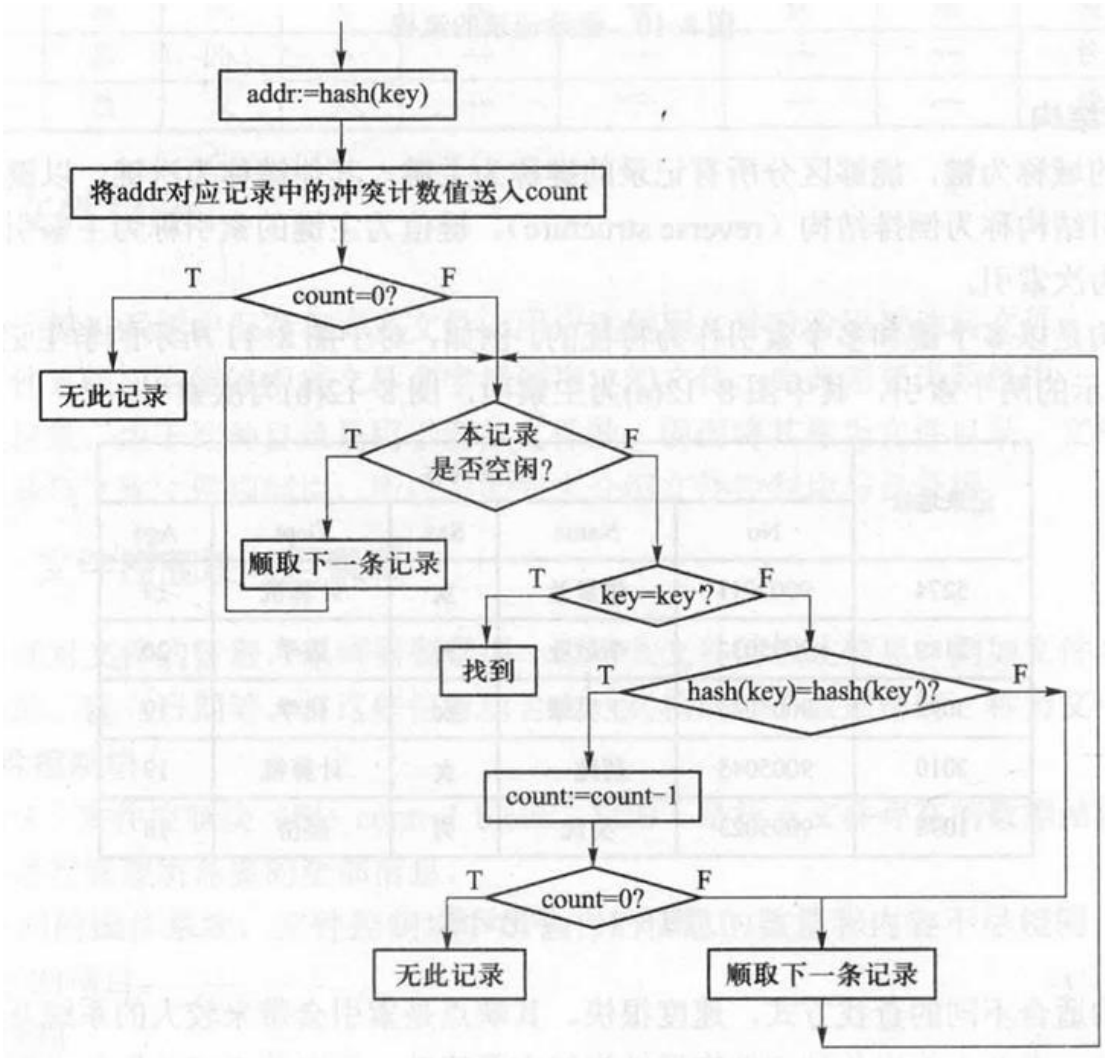
图 8-8　保存记录的流程
- 查找记录
  - 

图 8-9　查找记录的流程
- 删除记录

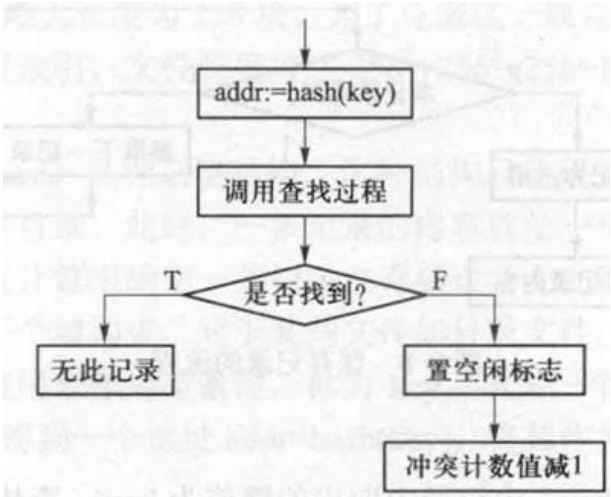
- 

图 8-10　删除记录的流程

- 倒排结构
  - 记录中的域称为键，能够区分所有记录的键称为主键，其他键称为次键
  - 以键值和记录地址构成的索引结构称为倒排结构
  - 键值为主键的索引称为主索引，键值为次键的索引称为次索引。

表 8-1　各种文件物理结构的主要特性

物理结构	长度变化	内存开销	外存开销	顺序访问速度		按号随机访问速度		按键随机访问速度	
				定长	变长	定长	变长	定长	变长
顺序结构	难	小	小	快	快	快	慢	慢	慢
链接结构	易	小	小	快	快	慢	慢	慢	慢
索引结构	易	大	大	快	快	快	慢	慢	慢
散列结构	易	小	小	—	—	—	—	快	—
倒排结构	易	大	大	—	—	—	—	快	快

• 文件存储空间的管理有哪些形式？各自有什么特点？

► 空闲块表

- 将所有的空闲块记录在同一个表中，该表称为空闲块表
- 表中包括两个栏目：首空闲块号和空闲块数
- 表中空闲块数为0的表目被标记为表尾
- 特别适合于文件的物理组织形式为连续结构的文件系统

首空闲块号	空闲块数
12	7
38	27
96	4
—	0
…	…

图 8-22　空闲块表

  - 图8—22所示的空闲块表记录当前外存储器中从第12块到第18块是空闲的，从第38块到第64块是空闲的，从第96块到第99块是空闲的。所有其他块都是被占用的。

空闲块链

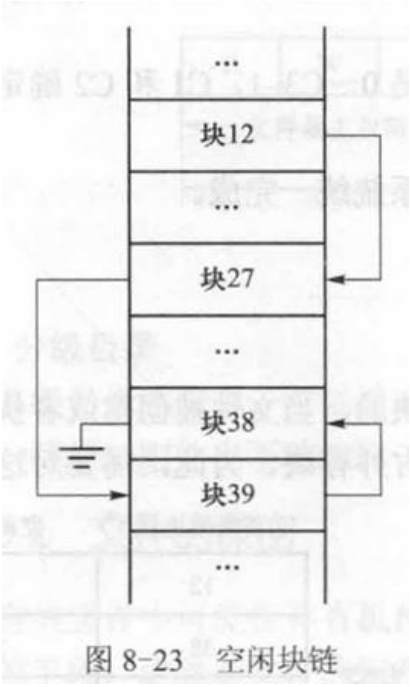
- 将所有的空闲块连成一条链
- 此时，外存空间的申请和释放是以块为单位的。申请时由链头取出一块，释放时将其连入链头。
- 这种外存空闲块的管理方法比较节省外存空间，但是其速度较慢，每申请一块或者释放一块都需要执行一次输入输出操作。
- 

图 8-23　空闲块链
- 8—23所示的空闲块链表示块12 、块27 、块38 、 块39是空闲的，所有其他块都是被占用的。

位图

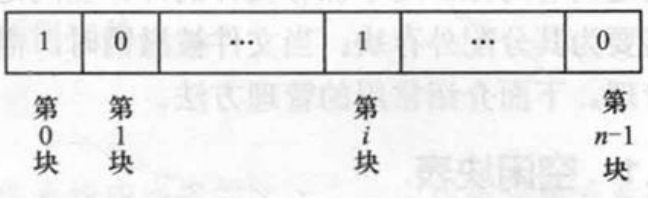
- 这种方法用1位（1b）来表示外存储器中一块的状态
- 可以规定当某一位的值为0时，该位所对应的外存块空闲，而当某一位的值为1时，该位所对应的外存块被 占用。
- 申请外存块时，可以在位图中从头查找为0的字位，将其改为1，返回对应的块号；归还外存块时，在位图中将该块所对应的字位改为0 。
- 

图 8-24　位图

- **介绍文件打开时所需占用的内存表目，以及它们之间的联系**

系统打开文件表

- 在内存中设立一个表目，用来保存已打开文件的文件控制块，该表被称为系统打开文件表
- 表中的内容除了FCB主部外，还包含以下3项信息：
  - 文件号：用于得到该文件FCB主部在外存储器中的位置。
  - 共享计数：用于记录当前有多少进程正在使用该文件，当其值为0时，表明是

一个空 表目。

- 修改标志：用于标识某文件的文件控制块在内存期间是否曾经被修改过，当最后一个 正在使用该文件的进程关闭该文件时，如果该文件控制块在内存期间曾经被修改过，需要将内存中修改后的文件控制块写回外存储器，否则不需要刷新外存储器。

- 系统打开文件表保存于操作系统空间中，用户程序不能访问它。
- 该表的长度决定了系统中可以同时打开的文件的最大数目。
-

FCB主部	文件号	共享计数	修改标志
…	…	…	…

图 8-25　系统打开文件表

用户打开文件表

- 由于文件是可以共享的，多个进程可能会同时打开同一文件，而其打开方式可能是不同的， 当前的读写位置通常也是不一样的。这些信息被记录在另一个表中，该表称为用户打开文件表
- 每个进程都有一个用户打开文件表
- 表中的系统打开文件表入口为一个指针，指向该文件控制块在系统打开文件表中的入口地址。
- 当多个进程共用同一文件时，不同进程的用户打开文件表中会有相同的系统打开文件表入口。
- 表中的文件描述符fd（file descriptor）是一个正整数，其值由其在表中的位置隐含确定，故实际不记录在表中，当文件被打开后返回给进程，其后进程便可使用描述符存取该文件，而不再使用文件名称。
- 用户打开文件表的位置应当记录在各进程的进程控制块中，用户打开文件表的长度决定了一个进程可以同时打开文件的最大数量。
-

文件描述符	打开方式	读写指针	系统打开文件表入口
…	…	…	…

图 8-26　用户打开文件表

表间联系

- 用户打开文件表指向系统打开文件表，如果多个进程共享同一文件，则多个用户打开文件表目对应系统的文件表中的同一入口

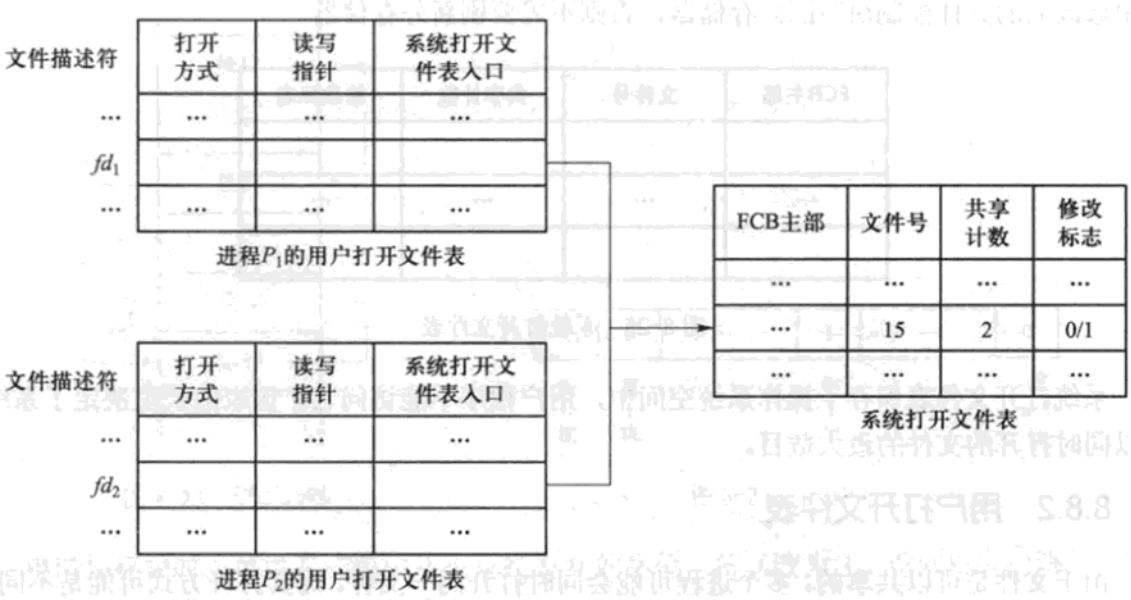


图 8-27　用户打开文件表与系统打开文件表之间的联系

- **描述创建文件的步骤**

► ①为此文件分配一个FCB主部，并对其初始化。

► ②将文件名称和文件号作为FCB次部，即将目录项填入文件路径名末级目录中。

► ③以写方式打开。

- **描述打开文件的步骤**

► ①根据文件路径名查目录找到FCB主部。

► ②根据打开方式、共享说明和用户身份检查访问的合法性。

► ③根据文件号查找系统打开文件表看该文件是否已经被打开，如是则共享计数值加1；否则取一个空闲的系统打开文件表项并将外存储器中FCB主部等信息填入此表项，共享计数值置为1 。

► ④在用户打开文件表中取一空表项，填写打开方式等，并指向系统打开文件表的对应表项。

返回文件描述符fd，它是一个非负整数，以后读写文件时就使用这个文件描述符，而不使 用文件名称。

- **描述关闭文件的步骤**

► ①由fd查找用户打开文件表，找到系统打开文件表的入口。

► ②系统打开文件表中的共享计数值减1，如减1后的值为0目修改标志为1，则将此文件控制块由内存写回外存FCB主部。

► ③将fd所对应的用户打开文件表项置为空闲。

- **描述指针定位的步骤**

► ①由fd查找用户打开文件表，找到对应的入口。

► ②将用户打开文件表中的文件读写指针位置设定为offset，后续读写命令由该指针处存取文件内容。

- **描述读文件的步骤**

执行步骤：① 由 fd 查找用户打开文件表，找到对应的入口。
② 根据用户打开文件表中所记录的打开方式和存取方式核查访问的合法性。
③ 查找系统打开文件表，找到文件的地址。
④ While (nrd>0)&&(未到文件末尾 offset<size) Do
根据 offset 和 nrd 计算下一个待访问块的逻辑块号和块内起止位置;
根据文件物理结构得到逻辑块号对应的物理块号;
If 该块在内存缓冲区 Then
If 其他进程在用该缓冲 Then 等待 Endif
所需部分复制到 buf，offset+复制量，nrd–复制量，buf+复制量，释放缓冲区
Else
分配缓冲区，填写头部，链入设备 I/O 队列，如设备空闲启动设备，等待 I/O 传输结束（切换进程），所需部分复制到 buf，offset+复制量，nrd–复制量，buf+复制量，释放缓冲区
Endif
Endwhile
⑤ 返回实际传输字节数。

- **描述写文件的步骤**

执行步骤：① 由 fd 查找用户打开文件表，找到对应的入口。
•
② 根据用户打开文件表中所记录的打开方式和存取方式核查访问的合法性。
③ 查找系统打开文件表，找到文件的地址。
④ While nwt>0 Do
根据 offset 和 nwt 计算下一个待访问块的逻辑块号和块内起止位置;
If 物理块存在(offset<长度)Then
If 本块已有缓冲 Then
If 其他进程在用 Then 等待 Endif
Else If(整块写)Then
分配缓冲区，填写头部(出);
Else 分配缓冲区，填写头部(入),缓冲区连入设备 I/O 队列，若设备非忙碌启动设备，等待 I/O 结束
Endif
Endif
buf 内容复制到缓冲区起止位置，调整头部(出)，缓冲区连到设备 I/O 队列，若设备空闲启动设备，offset+复制量，buf+复制量 nwt–复制量;
Else
分配磁盘块，分配缓冲区，填写缓冲区头部，buf 内容复制到缓冲区起止位置，缓冲区链到 I/O 队列，若设备空闲启动设备，offset+复制量，buf+复制量，nwt–复制量;

冲区链到 I/O 队列，若设备空闲启动设备，offset+复制量，buf+复制量，nwt-复制量；

Endif

Endwhile

- **描述建立连接的步骤**

► ①查目录找到文件old_name的FCB主部，由此得到文件号。

► ②查目录找到文件new name的末级目录。

► ③将文件号与new name中的末级名字合起来构成一个新的目录项，将其填入new_name 的末级目录文件中。

► ④将FCB主部中的连接计数值加1 。

- **描述断开连接的步骤**

► ①查目录找到文件Path _ name的FCB主部。

► ②将连接计数值减1 。

► ③如果减1后的值为0，则归还该文件所占用的全部存储块，该文件将被撤销。

► ④将FCB次部从末级目录中清除。

- **什么是文件的逻辑结构和物理结构？他们各自有哪几种形式？**

► 文件的逻辑结构是从用户的观点看到的文件组织形式。它与存储设备的特性无关。分为两种形式：无结构的流式文件和有结构的记录式文件。

► 文件的物理结构是指文件在外存上的存储组织形式。文件的物理结构与存储设备的特性有很大关系。通常有4种形式：顺序结构、链接结构、散列结构、索引结构。

- **何谓文件系统？为何要引入文件系统？文件系统所要解决的问题(功能)主要有哪些？**

► 文件系统是指负责存取和管理文件信息的机构，也就是负责文件的建立、撤销、组织、读写、修改、复制及对文件管理所需要的资源（如目录表、存储介质）实施管理的软件部分。

► 引入文件系统的目的: 实现文件的“按名存取”，力求查找简单；使用户能借助文件存储器灵活地存取信息，并实现共享和保密。

► 文件系统所要解决的问题(功能)主要有：

  - 1）、有效地分配文件存贮器的存贮空间(物理介质)。
  - 2）、提供一种组织数据的方法(按名存取、逻辑结构、组织数据)
  - 3）、提供合适的存取方法(顺序存取、随机存取等)。
  - 4）、方便用户的服务和操作。
  - 5）、可靠的保护、保密手段。

- **试说明文件系统中对文件操作的系统调用处理功能。**

► 系统调用是操作系统提供给编程人员的唯一接口。利用系统调用，编程人员在源程序中

动态请求和释放系统资源，调用系统中已有的功能来完成那些与机器硬件部分相关的工作以及控制程序的执行速度等。系统调用如同一个黑匣子，对使用者屏蔽了具体操作动作，只是提供了有关功能。

► 有关文件系统的系统调用是用户经常使用的，包括文件的创建（create）、打开(open)、读(read)、写(write)、关闭(close)等。

- **为什么将文件定义为信息项的序列,而不是信息项的集合?**

► 因为构成文件的信息项之间是有次序关系的,这与数据库中的记录不同。数据库中同一关系的记录之间没有次序关系,因而关系是记录的集合。

- **试举例说明何种文件长度是固定不变的,何种文件长度是动态变化的。**

► 某些系统可执行程序(如shell、vi)的长度通常是固定不变的;而用户正在编辑的文本文件或源代码文件的长度通常是动态变化的。

- **试比较文件名、文件号、文件描述符之间的关系。**

► 文件名是文件的外部名字,通常是一个符号名(字符串),同一文件可以有多个文件名(如通过link命令建立连接)。文件号是文件的内部名字,通常是一个整数,文件号与文件具有一对一的关系。文件描述符是文件打开时返回的整数(入口地址),对应用户打开文件表中的一个入口。

► 同一个文件可以被多个用户同时打开,此时返回的文件描述符一般不同。同一个文件也可以被同一个用户多次打开,每次打开时返回的文件描述符一般也不同。

- **将文件控制块分为两个部分有何好处?此时目录项中包含哪些成分?**

► 将文件控制块(FCB)划分为主部和次部两部分具有如下两个主要优点。

► (1)提高查找速度。查找文件时,需用欲查找的文件名与文件目录中的文件名相比较。由于文件目录保存于外存,比较时需要将其以块为单位读入内存。由于一个FCB包括许多信息,一个外存块中所能保存的FCB个数较少,这样查找速度较慢。而将FCB分为两部分之后,文件目录中仅保存FCB的次部,这样一个外存块中可容纳较多的FCB,从而大大地提高了文件的检索速度，同时也减少了文件目录所占空间。

► (2)实现文件连接。所谓连接就是给文件起多个名字,这些名字都是路径名,可为不同的用户所使用。次部仅包括一个文件名和一个标识文件主部的文件号,主部则包括除文件名之外的所有信息和一个标识该主部与多少个次部相对应的连接计数。当连接计数的值为0时,表示一个空闲未用的FCB主部。

5. 为何文件在使用之前需要打开？多个进程共享同一文件时，其文件控制块为何在内存中只能保持一个副本？

答：当一个文件被使用时，其 FCB 中的信息需要被经常地访问。如果每次访问 FCB 都去读写外存，则速度会大大地降低。为了解决这一问题，在内存中设立系统打开文件表，将文件对应的 FCB 读入内存并保存在该表中，以备读写时使用，因而文件在使用前必须打开。

由于文件是可共享的，多个进程可能会同时打开同一个文件，而其打开方式可能是不同的，当前的读写位置通常也是不一样的。为了防止信息冗余，将这些个性化信息记录在另外一个表中，该表称为用户打开文件表，每个进程有一个。表中包含以下内容：

文件描述符	打开方式	读写指针	系统打开文件表入口

表中的系统打开文件表入口为一指针，指向该文件 FCB 在系统打开文件表中的入口地址。当多个进程共用同一个文件时，不同进程的用户打开文件表项可能指向相同的系统打开文件表入口。

这样共享文件的 FCB 在内存中只有一个副本，其好处在于：(1) 节省了存储空间；(2) 避免了对于多副本一致性维护的困难；(3) 减少了 I/O 交换次数，提高系统效率。

6. 使用文件描述符存取打开文件与直接使用文件名称相比，有何优点？

答：首先，文件名是一个字符串，操作速度慢且所占空间大，而文件描述符为一整数，其处理效率明显高于字符串。其次，文件被打开后其控制信息(FCB)被缓冲到内存系统空间，文件描述符作为用户打开文件表中的入口地址直接与内存 FCB 建立起联系，而文件名则无法做到这一点。

7. 用户打开文件表中包含哪些内容？为何不能将其合并到系统打开文件表中？

答：用户打开文件表中包含以下内容：

文件描述符	打开方式	读写指针	系统打开文件表入口

由于文件是可共享的，多个进程可能会同时打开同一文件，而其打开方式可能是不同的，当前的读写位置通常也不一样。如果将这些信息合并到系统打开文件表中，就会导致一个共享文件占用多个系统打开文件表表目，这些表目的大部分内容是重复的。若一个进程对文件的操作导致 FCB 内容变化，该进程关闭文件时就要将 FCB 回写到外存中。这样增加了内外存传输的次数，也容易导致 FCB 内容的不一致。因此，通常将打开方式和读写指针记录在另外一个表，即用户打开文件表中。

8. 试说明对于以下文件操作命令，文件管理系统如何进行访问合法性检查。

(1) 打开文件；(2) 读写文件；(3) 删除文件。

答：(1) 打开文件：根据打开方式、共享说明和用户身份检查访问合法性；(2) 读写文件：根据用户打开文件表中所记录的打开方式和存取方式检查访问的合法性；(3) 删除文件：根据共享说明和用户身份检查访问合法性。

9. 采用文件连接技术后，文件名称与文件是否一一对应？文件号与文件是否一一对应？文件描述符与文件是否一一对应？

答：采用文件连接技术后，文件名称与文件为多对一；文件号与文件为一对一；文件描述符与文件为多对一。

10. 对于散列文件结构，回答下述顺序探查法解决冲突方面的问题。

（1）对于一个非空闲记录来说，其键值 key 的散列值 hash(key)是否一定与该记录的地址 addr 相同？

（2）当一条记录的冲突计数值为 0 时，该记录是否一定空闲？

（3）当一条记录空闲时，该记录的冲突计数值是否一定为 0？

答：（1）不一定，当前面记录发生冲突时，可能在顺序探查时占用本记录；（2）不一定，可能存放冲突的记录；（3）不一定，本记录被删除后，仍可能有其他记录的 hash(key)为本记录入口地址。

11. 何谓文件连接？如何实现文件连接？

答：所谓连接就是给文件起多个名字，这些名字都是路径名，不同用户可以使用不同路径名访问同一文件，从而实现文件的共享。

通过将文件 FCB 分为主部和次部，多个次部可以对应同一个主部，从而实现文件连接。例如，在 UNIX 系统中文件连接的命令形式是 link(old_name,new_name)，其中，old_name 表示已存在的文件路径名；new_name 表示欲连接的文件路径名。该命令的执行步骤如下：（1）查目录找到文件 old_name 的 FCB 主部，由此得到文件号；（2）查目录找到文件 new_name 的末级目录；（3）将文件号与 new_name 中末级名字合起来构成一个新的目录项，将其填入 new_name 的末级目录文件中；（4）将 FCB 主部中的连接计数加 1。

第九章 设备与输入输出管理

2024年6月25日 16:30

- **设备管理的功能是什么?**

► 设备分配：按照设备类型和相应分配算法决定将设备分配到哪一个要求该类设备的进程
► 设备处理：即设备驱动，实现I / O进程与设备控制器之间的通信。
► 缓冲管理：为了提高CPU和I/O设备之间操作的并行程度，并且能够尽量减少中断的 次数，大多数的I/O操作都要使用缓冲区

- **写出设备管理的4个目标**

► 方便性
► 并行性
► 均衡性
► 独立性

- **写出几种设备分类的方式与对应的类型**

► **输入输出型设备与存储型设备**
  - 输入输出型设备是向CPU传输信息、或输出经CPU处理过的信息的设备。
  - 从用途上来说， 输入输出型设备包括以下两类：
    - 人机输入输出设备
      - 如键盘、扫描仪、打印机、绘图仪、数码相机等。
    - 机输入输出设备（通信设备）
      - 如网卡、调制解调器等。
  - 存储型设备既可用于实现虚拟存储系统，又可用于建立文件系统，还可用于构造作业输入井和输出井。存储于这类设备上的信息可以长期保存。
► **块型设备与字符型设备**
  - 块型设备
    - 通常块型设备就是存储型设备。
    - 这类设备由若干个长度相同的块所构成， 一块的长度通常为$2^i B$
    - 对于这类设备来说，块是存储和分配的基本单位，也是数据传输的基本单位。
  - 字符型设备
    - 通常字符型设备就是输入输出型设备（包括通信设备）。
    - 这类设备进行数据传输的基本单位是字节。
  - 时钟与显示器
    - 这两种设备既不属于块型设备，也不属于字符型设备。
    - 硬件时钟有两个
      - 一个是绝对时钟：记载当前日期和时间，当其不准时系统用户可以调

整，所用用户均可读取其值
        - 另一个是闹钟： 它定时产生中断，一般以毫秒为单位。
    - 显示器在内存系统空间有一段保留的存储区，通过送数指令（MOV）向该区域写信息即可在显示器上显示出来，这片保留的区域属于系统空间，一般用户不可访问，因而操作系统不会将任何进程的地址空间映射到显示内存。

► **独占型设备与共享型设备**

- 独占型设备包括所有的字符型设备及磁带机
    - 对于这类设备来说，在任意一段时间之内最多只能有一个进程占有并使用它。
- 共享型设备包括除磁带机以外的所有块型设备
    - 对于这类设备来说，多个进程的数据传输以块为单位是可以交叉的。

• 有哪些数据传输方式？各自有什么特点？

► 程序控制查询方式

- **探询**：探询（Polling）又称程序控制输入输出，是最早的输入输出控制方式。处理器代表进程向相应的设备模块发出输入输出请求，然后处理器反复查询设备状态，直至输入输出完成。
- 缺点是忙式等待，处理器与设备完全串行工作， 由于设备速度远远低于处理器速度，忙式等待过程将消耗大量处理器时间。
- 探询方式可以采用硬件提供的专用输入输出指令，也可以采取内存操作指令完成。后者将设备地址映射为内存地址空间的一部分，是硬件通常提供的输入输出指令形式，称为内存映射输入输出。

► 中断驱动方式

- 引入中断之后，设备具有中断CPU（中央处理器）的能力，设备与CPU可以并行。
- 当设备较多时，对CPU的中断打扰很多。
- 另外，中断伴随处理器的状态切换， 增加了系统开销。
- 与探询方式一样，中断驱动输入输出方式既可以采用硬件提供的专用输入输出指令，也可以采用内存映射输入输出指令完成。

► 内存映射方式

- 具有即时完成的特点，即指令执行完1 / 0操作也即 执行完，因而没有中断需要处理。

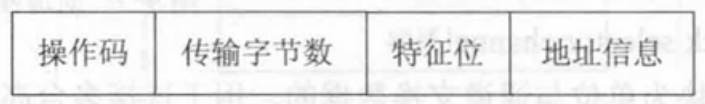
► DMA方式

- 硬件提供DMA控制器
- DMA控制器通过总线与设备、内存相连。
- DMA方式的传输步骤如下。
    - ① CPU将操作量送到DMA控制器的操作数寄存器operands中，包括内存起始地址、传输数量等。
    - ② CPU将操作码送到DMA控制器的opcode寄存器以启动DMA控制器。DMA控制器将其忙碌寄存器（busy）置位，表明在此期间不再接受新的操作命令。此后，CPU可以执行与该控制器无关的其他操作。

- ③ DMA控制器与设备交往，将数据由缓冲区传送到设备或者由设备传送到缓冲区。
- ④ DMA控制器将缓冲区内容复制到内存空间，或由内存空间复制到缓冲区。
- ⑤计数器值减1 。若结果非0（未传送完）则转步骤③继续传输；否则转步骤⑥ 。
- ⑥传输结束时，DMA控制器复位其忙碌寄存器，并向CPU发送中断请求。
- ⑦ CPU读入并检测其DMA状态寄存器，以确认操作是否成功。

► 通道方式

- 具有通道结构的计算机系统，主机、通道、控制器和设备之间采用四级连接，实施三级控制。一个CPU通常可以连接若干通道，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。 CPU通过执行I/O指令对通道实施控制，通道通过执行通道命令对控制器实施控制，控制器发出动作序列对设备实施控制，设备执行相应I/O操作。
- 通道连接方式主要有两种：
  - 单通路的连接方式：对于单通路方式，每个设备与一个控制器相连，每个控制器与一个通道相连
  - 多通路的连接方式：允许多对多相连
- 通道涉及以下内容：
  - 1．通道指令系统
    - (1) 基本操作
      - ①空操作：不执行任何操作，顺序取出下一条指令。
      - ②读操作：从指定设备中输入一批数据。
      - ③写操作：向指定设备输出一批数据。
      - ④控制操作：控制外围设备机械性动作，如磁带反绕、磁盘引臂移动、打印纸换页等。
      - ⑤转移操作：实现指令的非顺序执行。
      - ⑥结束操作：通道程序执行完毕，向主机发送通道中断信号。
    - (2) 指令格式



  - 2．通道运控部件
    - ①通道地址字（channel eddress word , CAW ）
    - ②通道命令字（charmel command word , CCW ）
    - ③通道状态字（channel status word , CSW ）
    - ④通道数据字（channel data word , CDW ）
  - 3．通道存储区域
    - 通道并没有独立的存储空间，它需要与主机共享同一个内存空间。访问内存采用“周期窃用”方式。
    - 通道使用内存具有以下两种用途：

        - ①保存通道程序。
        - ②保存交换数据。
    - 4. 通道程序执行过程
        - 通道有3种类型：
            - 字节多路通道：多个非分配型子通道，连接低速外围设备
            - 数组选择通道：一个分配型子通道，连接多台高速设备
            - 数组多路通道：多个非分配型子通道，连接多台高速设备

- **设计输入输出调度算法应当考虑哪两个基本因素?**

► ①公平性：一个输入输出清求应当在有限的时间之内得到满足。
► ②高效性：减少设备机械运动所带来的时间开销。

- **读写一个磁盘块需要多长时间一般由哪几个因素确定?**

► 寻道时间（seek time ），这是指将磁盘引臂移动到指定柱面所需要的时间
► 旋转延迟（rotational delay ），这是指指定扇区旋转到磁头下的时间；
► 传输时间（transfer time ），这是指读写一个扇区的时间。

- **列举常见的磁盘引臂调度算法并简单介绍**

► 先到先服务算法：先到先服务(FCFS)算法按照输入输出请求的次序为各个进程服务,这是最公平且最简单的算法,但是执行效率不高。
► 最短查找时间优先算法：（SSTF）其思想是优先为距离磁头当前所在位置最近柱面的请求服务。该算法缺乏公平性，存在饥饿和饿死问题。
► 扫描算法：扫描算法（SCAN）又称电梯算法，因其基本思想与电梯的工作原理相似，故得此名。无访问请求时，磁头引臂停止不动；当有访问请求时，引臂按照电梯移动规律运动，并为路经柱而的访问请求服务。起始时磁头由最外柱面向内柱面移动，并为路经的请求服务。一旦内柱面没有访问请求，则改变移动方向（如果外柱面有请求）或者停止移动（外柱面也无请求）
► 循环扫描算法：循环扫描（c-scan）算法是为消除边缘柱面与中部柱面等待时间差异而进行的改进。磁头只在单方向移动过程中才为路经的请求服务，一旦该方向没有请求，则立即快速回扫到另一端提出请求的第一个柱面。在此回扫过程中并不处理访问请求，然后重新开始新一轮扫描。
► N步扫描算法：N步扫描把磁盘请求队列分成若干长度为n的子队列，并按到达次序处理这些子队列。使用扫描策略服务请求队列中的前n个请求，本次扫描后，接着服务下一组n个请求。到达的请求放在请求队列的队尾。 N步扫描可以通过改变N的值进行调节，当N =1时，N步扫描退化为先到先服务；当N很大时，N步扫描接近扫描算法。
► 冻结扫描算法：冻结扫描算法是指，只用扫描策略服务那些在一次特定扫描开始时已经到达的请求，即请求队列被“冻结”，在扫描期间新到达的请求被组合在一起，并在回程扫描时按扫描算法进行处理。在实现时，对每个柱面可以设置两个队列，按扫描方向交替用作服务队列和等待队列。

- **什么叫缓冲(buffering)?什么叫缓存(caching)?缓冲与缓存有何差别?**

► 利用存储区缓解数据到达速度与离去速度不一致而采用的技术称为**缓冲**,此时同一数据只包含一个副本。例如,操作系统以缓冲方式实现设备的输入和输出操作主要是为了缓解处理器与设备之间速度不匹配的矛盾,以提高资源利用率和系统效率。

► 缓存是为提高数据访问速度而将部分数据由慢速设备预取到快速设备上的技术,此时同一数据存在多个副本。例如,远程文件的一部分被取到本地。

► 当然,在有些情况下,缓冲同时具有缓存的作用。例如,UNIX系统对于块型设备的缓冲区,在使用时可保持与磁盘块之间的对应关系,既有缓冲的作用也有缓存的作用,通过预先读与延迟写技术,进一步提高了输入输出效率。

- **考虑由文件读入若干信息到进程空间中,信息首先由磁盘读入操作系统缓冲区,再由缓冲区复制到进程空间中。为什么不能将信息直接由磁盘读到进程空间中?**

► 磁盘输入输出以块为基本单位,而进程读取的文件内容可能对应块中的部分内容,若直接由磁盘块复制到进程空间,则可能会读进不需要的内容,覆盖了进程中有用的单元。

- **与为每个设备配置一个(或者若干个)缓冲区相比,采用可为多个设备共用的缓冲池有何优点?**

► 将一个缓冲区与一个固定的设备相联系,从而不同设备使用不同的缓冲区,这种缓冲区管理模式称为私用缓冲。私用缓冲利用率低,例如,某一执行I/O传输的设备,其私用缓冲区可能不够,而其他未执行I/O操作的设备,其私用缓冲区则可能被闲置而导致浪费。

► 为了提高缓冲区的利用率,通常不将缓冲区与某一个具体设备固定地联系在一起,而是将所有缓冲区集中起来加以管理,按需要动态分派给正在进行I/O传输的设备。系统中的共用缓冲区集合被称为缓冲池(buffer pool)。

- **在系统缓冲区空间总长度固定的前提下,一个缓冲区过大或者过小各有何优点和缺点?**

► 缓冲区过大会造成资源浪费(平均浪费半个缓冲区容量),但是有利于减少I/O传输次数;

► 缓冲区过小则会因I/O传输次数增多而加大系统开销。另外,缓冲区过小会引起缓冲链指针过多而浪费缓冲空间。

- **假设要修改某一磁盘块上的一部分,而其他部分保持原内容不变,应当如何做?**

► 首先将该磁盘块内容读入内存缓冲区,然后在内存中修改相关内容,最后将修改后的缓冲区内容完整地回写到磁盘对应块中。

- **操作系统中引入缓冲技术的目的是什么？举例解释缓冲的必要性**

► 在操作系统中，以缓冲方式实现设备的输入输出操作主要是为了缓解处理器与设备之间速度不匹配的矛盾，从而提高资源利用率和系统效率。

► 处理器的速度是很快的，而设备的速度则比较慢。例如有一个进程，它时而进行长时间的计算，时而产生阵发性的输出传送到一台打印机上。若无缓冲机制，在阵发性输出时，由于打印机的速度跟不上处理器的速度，处理器不得不经常等待：而在计算阶段，打印机又被闲置。 若在打印机与处理器之间设置一个缓冲区，情况便可大为改观：当进程产生阵发性输出时，将输出信息暂存在缓冲区中，由打印机取出慢慢打印，处理器在将数据传送到缓冲区之后便可继续执行其计算仟务，此时处理器可与设备并行操作。

- **根据系统设置的缓冲区个数，可将缓冲技术分为哪几类?**

► 单缓冲：在设备与进程之间设置一个缓冲区，设备与进程交换数据（如输入）时信息由设备传到缓冲区，然后再由缓冲区传给进程。然后重复上述过程，直至传输完成。

► 双缓冲：在设备与进程之间设置两个缓冲区，设备与进程交换数据（如输入）时信息由设备传到一个缓冲区。第一个缓冲区填满后，信息由设备传到第二个缓冲区，同时处理器将第一个缓冲区中的内容复制到进程空间。然后，设备再将输入信息传到第一个缓冲区，同时处理器将第二个缓冲区的内容复制到进程空间，如此交替，可以提高处理器与设备之间的并行性。

► 循环缓冲：在设备与进程之间设置多个缓冲区，这些缓冲区链成环状，有两个指针in和out，in指向当前输入的位置，out指向当前取出的位置。

► 缓冲池：为了实现缓冲输入输出，操作系统通常需要提供两组缓冲区：一组用于块型设备，另一组用于字符型设备。用于块型设备的缓冲区一般较大，其长度通常与外围设备物理块的长度相同； 用于字符型设备的缓冲区一般较小

► 

图 9-20　缓冲池

  - 缓冲池是属于操作系统空间的，用户程序不能直接对其进行操作，只能通过与输入输出操作有关的系统调用进入操作系统来间接地使用它们。

- **每个设备都有一个缓冲队列，所需要的缓冲区是动态向系统申请的，用完后立即归还给系统。请分别介绍输入型设备缓冲输入的实现、输出型设备缓冲输出的实现，以及输入输出型设备缓冲输入和缓冲输出的实现。**

► 1. 输入型设备
  - 对于这类设备，信息的流向是这样的：输入设备——缓冲区——进程空间
  - 信息由输入设备到缓冲区的传输由通道程序完成，由缓冲区到进程空间的传输由操作系统代替进程完成。

- 设备被启动之前为其申请一个缓冲区，输入的信息被传送到缓冲区中，缓冲区满时将其连到设备的输入队列上，并申请下一个缓冲区。当进程需要由设备读取信息时，顺取输入队列L上的缓冲区，并将缓冲区中的内容复制到进程空间，然后将缓冲区释放。

► 2. 输出型设备

- 对于这类设备，信息的流向是这样的：进程空间——缓冲区——输出设备
- 信息由进程空间到缓冲区的传输由操作系统代替进程完成，由缓冲区到输出设备的传输由通道程序完成。
- 当进程需要将信息传送到输出设备时，系统代为申请一个缓冲区，信息由进程空间传送到缓冲区中，一个缓冲区装满后连到设备的输出队列上并启动设备输出，同时申请下一个缓冲区。 一个缓冲区中的内容输出完后，缓冲区被释放。

► 3．输入输出型设备

- 这类设备多属于块型设备，如磁盘、磁带等。对于这类设备，信息的流向是这样的：进程空间——缓冲区——输入输出设备
- 对于这类设备来说，输入输出操作可以交叉进行。为此，通道需要知道每一个缓冲区所对应的操作究竟是读还是写。此外，还需要给出设备的地址，如为哪一个磁盘块。为此，输入输出队列上的每一个缓冲区都应当带有一组说明信息，这组说明信息称为缓冲区头
- 进程执行输入操作时，系统代为申请缓冲区，并填写好缓冲区头，然后将此缓冲区连到设备的输入输出队列上。设备驱动程序根据缓冲区头的说明信息将设备上的指定块读入缓冲区体。 数据传输结束后产生中断，由中断处理程序将缓冲区体中的内容复制到进程空间，然后释放缓冲区。注意：这里并未真正起到缓冲作用，因为每次输入输出都需要启动设备将所需要的块传送到缓冲区中，而不能从缓冲区中直接得到。
- 进程执行输出操作时，系统代为申请缓冲区，并将信息由进程空间复制到缓冲区体中，然后填写好缓冲区头，再将缓冲区连到设备的输入输出队列上。设备驱动程序根据缓冲区头的说明信息将缓冲区体中的内容传送到设备的指定块中。数据传输结束后发生中断，由中断处理程序释放缓冲区。

- **简述输入输出操作的演变过程:查询方式→中断方式→通道方式,并分析这种演变对于多道程序设计所带来的影响。**

► I/O操作最早为查询方式,将待传输的数据放入I/O寄存器并启动设备,然后反复测试设备状态寄存器直至完成。采用这种方式,处理器与设备之间是完全串行的。

► 随着中断I/O方式的产生,处理器在启动设备后,可进行其他计算工作,设备与处理器并行。

► 当设备I/O操作完成时,向处理器发送中断信号,处理器转去进行相应的处理,然后可能再次启动设备传输。中断使多道程序设计成为可能:一方面中断使操作系统能够获得处理器控制权,另一方面通过I/O中断可以实现进程状态的转换。

► 中断使处理器与设备之间的并行成为可能,但I/O操作通常以字节为单位,当设备很多时会对处理器打扰很多,为此人们设计了专门处理I/O传输的处理器 -- 通道。通道具有自己的指令系统,可以编写通道程序,一个通道程序可以控制完成许多I/O传输,只在通道程序结束

时,才向处理器发生一次中断。

- **通道与DMA方式之间有何共同点?有何差别?**

► 通道与DMA(Direct Memory Access,直接存储器访问)都属于多数据I/O方式

► 二者差别在于:通道控制器具有自己的指令系统,一个通道程序可以控制完成任意复杂的I/O传输,而DMA并没有指令系统,一次只能完成一个数据块的传输。

- **用户申请独占型设备时为什么不指定具体设备而仅指定设备类别?**

► 进程申请独占型设备资源时,应当指定所需设备的类别,而不是指定某一具体的设备编号,系统根据当前请求以及资源分配情况在相应类别的设备中选择一个空闲设备并将其分配给申请者,这称为设备无关性。这种分配方案具有如下两个优点:

  - (1)提高设备资源利用率。假设申请者指定具体设备,被指定的设备可能正被占用,因而无法得到,而其他同类设备可能空闲,造成资源浪费和进程不必要的等待;
  - (2)程序与设备无关。假设申请者指定具体设备,而如果被指定设备已坏或不联机,则需要修改程序。

- **为何不允许用户程序直接执行设备驱动指令?**

► (1)系统中的设备可能被多个进程所共享,例如,磁盘就是这样的设备。如果允许用户程序直接执行设备驱动指令,那么就有可能损坏设备;

► (2)设备操作涉及很复杂的驱动过程,一般用户编写驱动程序会是很大的负担,操作系统的目标是方便用户使用计算机系统,因而提供标准驱动程序。

- **何谓“磁道歧视”?假设每个磁道各有一个磁头,是否还存在磁道歧视问题?**

► 在最短查找时间优先(Shortest Search Time First,SSTF)等磁盘引臂调度算法中,磁头引臂可能长时间停留在磁盘局部的某些磁道,而不光顾另外一些磁道。例如,如果某一时刻外磁道请求不断,则内磁道请求可能长时间得不到满足,这种现象称为“**磁道歧视**”(track discrimina-tion)。若每个磁道各有一个磁头,则不存在磁道歧视问题。

- **处理器与通道之间是如何通信的?通道与处理器之间呢?**

► 通道与处理器之间相对独立,通道程序的执行可与处理器的操作并行。因为一个系统中可能有多个通道,这些通道也可并行地执行相应的通道程序。

► 通常,通道程序形成之后,处理器将通道程序的起始地址放到内存指定单元处,然后执行通道启动指令使通道开始工作。通道被启动之后从指定单元中取出通道程序的起始地址,并将其放入通道地址字(Channel Address Word,CAW)中,据此依次地执行各条通道指令。当通道程序执行完毕或执行到通道结束指令时,产生通道中断信号,并将该信号发给处理器,处理器响应中断后取出中断字,分析中断原因,进行相应的中断处理。

- **说明下列术语之间的对应关系。**
  (1)I/O设备;(2)I/O驱动程序;(3)I/O进程。

► 一般来说,一个I/O驱动程序与多个同类设备相对应,一个I/O设备与一个I/O进程相对应。

- **UNIX系统中将设备分为块设备和字符设备，它们各有什么特点?**

► 字符设备是以“字符”为单位进行输入、输出的设备，即这类设备每输入或输出一个字符就要中断一次主机CPU请求进行处理，故称为慢速设备。

► 块设备是以“字符块”为单位进行输入输出的设备，在不同的系统或系统的不同版本中，块的大小定义不同。但在一个具体的系统中，所有的块一旦选定都是一样大小，便于管理和控制，传送效率较高。

- **什么叫通道技术？通道的作用是什么？**

► **通道**是一个独立于CPU的专管输入/输出控制的处理机，它控制设备与内存直接进行数据交换。它有自己的通道指令，这些通道指令受CPU启动，并在操作结束时向CPU发中断信号。

► 通道方式进一步减轻了CPU的工作负担，增加了计算机系统的并行工作程度。

- **在设备管理中设置缓冲区的作用是什么？根据系统设置缓冲区的个数，缓冲区可以分为哪几种?**

► 在设备管理中设置缓冲区的作用：
  - （1）缓和CPU和I/O设备之间速度不匹配的矛盾。
  - （2）减少中断CPU的次数。
  - （3）提高CPU和I/O设备之间的并行性。

► 根据系统设置缓冲区的个数，可以分为单缓冲、双缓冲、多缓冲以及缓冲池等四种。

- **按资源分配管理技术，输入输出设备类型可分为哪三类？简述其区别**

► 按资源分配管理的特点，输入输出设备可分为独占型设备、共享型设备和虚拟设备三类。
  - 独占型设备：即不能共享的设备，一段时间只能由一个作业独占。如打印机、读卡机、磁带机等。所有字符型输入输出设备原则上都应是独享设备。
  - 共享型设备：可由若干作业同时共享的设备，如磁盘机等。共享分配技术保证多个进程可以同时方便地直接存取一台共享设备。共享提高了设备的利用率。块设备都是共享设备。
  - 虚拟设备：利用某种技术把独享设备改造成多台同类型独享设备或共享设备。虚拟分配技术就是利用独享设备去模拟共享设备，从而使独占设备成为可共享的、快速I/O的设备。实现虚拟分配的最有名的技术是SPOOLing技术，即假脱机技术。

- **在磁盘调度算法中，SSTF和C_SCAN算法分别是如何实现的？并比**

较它们的性能。

- SSTF方法：根据磁头的当前位置，首先选择请求队列中距磁头距离最短的请求为之服务。
- C_SCAN方法：磁头从盘面上的一端（逐柱面地）向另一端移动，遇到请求立即服务；回返时直接快速移至起始端而不服务于任何请求。如此往返单向地扫描并平均地为各种请求服务。
- 性能比较：SSTF方法可以获得较短的寻道时间，但可能有饿死现象。适合于负载不大的系统。C_SCAN方法在负载较大的系统中，可以获得较好的性能，并且不存在饿死现象。

• 简述设备驱动程序的作用?

- 设备驱动程序是驱动物理设备和DMA控制器或I/O控制器等直接进行I/O操作的子程序的集合。负责设置相应设备有关寄存器的值，启动设备进行I/O操作，指定操作的类型和数据流向等。

• 举例说明面向块的设备与面向流的设备之间的区别?

- 一般来说，面向块的设备以固定大小的块来存储数据，数据的传送是方式是每次一个数据块，对数据的引用通过数据块号来进行，比如磁带、磁盘等就是典型的块设备；而面向流的设备是以字节流的方式进行数据的传送，不存在块结构，如打印机、终端、键盘等都是典型的面向流的设备。

• 什么是设备无关性？如何实现设备独立性?

- 设备无关性是指用户编写程序时所使用的设备与实际使用的设备无关。
- 为实现设备无关性，要求用户程序对设备的请求采用逻辑设备名，而程序执行时使用武力设备名。因此，操作系统需要提供逻辑设备名与物理设备名的转换机制。一般采用系统设备表实现该转换。

• 什么是物理设备？什么是逻辑设备？两者之间有什么区别和联系?

- 进行实际输入输出操作的硬件设施是物理设备.
- 操作系统中规定用户程序中不要直接使用设备的物理名称，而用一另外的名称代之来操作，这就是逻辑设备.
- 逻辑设备是物理设备属性的表示，它并不特指某个具体的物理设备，而是对应于一批设备，具体的对应则在操作系统启动初始化时确定，或在运行过程中根据设备的使用情况由系统或用户再次确定.

选择题知识点总结

2024年6月26日 0:32

- 设计批处理多道系统时，首先要考虑的是系统效率和吞吐量
- 若当前进程因时间片用完而让出处理机，则进程应转变为就绪状态
- 进程所请求的一次打印输出结束后，进程的状态由等待到就绪
- 操作系统提供给应用程序的接口是系统调用
- 作业调度程序是从处于后备状态的作业中选取一个作业并把它装入主存
- 每类中断事件有一个中断向量（不是每个）
- 操作系统提供给应用程序的接口是系统调用
- 操作系统提供给编程人员的接口是程序接口，也就是系统调用命令
- 进入中断处理的程序只能是OS程序
- 进程从运行态到等待态可能是由于现运行进程执行了P操作
- 进程调度程序只能使由就绪态转化为运行态
- 在进程运行时发生的如下事件中：
  ①时钟中断
  ②调用访管输入
  ③执行非法指令
  ④I/O中断
  一定进行进程切换的事件是（②③）
- 造成死锁的必要条件之一是因为资源具有不可剥夺的性质
  - 会引起死锁的资源有：打印机、磁带机、扫描仪
  - 不会引起死锁的资源有：CPU、磁盘
- 不发生死锁的最大进程个数是$\lfloor (n-1)/(m-1) \rfloor$，其中n是资源总个数，m是每个进程需要的资源个数
- 一个进程被唤醒意味着进程状态变为就绪
- 一个进程被唤醒意味着该进程可能重新占用CPU
- 对于两个并发进程，设互斥信号量为mutex，若mutex=0，则表示有一个进程进入临界区
- 不管系统中是否有线程，进程都是拥有资源的独立单位
- 不会产生内部碎片的存储管理是分段式存储管理
- 在Hoare管程中，假设进程p正在使用管程，当P执行唤醒Signal操作时，其含义是Signal and urgent wait
- 在段页式分配中，CPU每次从内存中取一次数据需要3次访问内存
- 操作系统处理缺页中断时，选择一种好的调度算法对主存和辅存中信息进行高效调度，尽可能地避免抖动
- 系统“抖动”现象的发生是由置换算法选择不当引起的
- 在请求分页存储管理中，若采用FIFO页面淘汰算法，则当分配的页面数增加时，缺页中断的次数可能增加也可能减少
- 操作系统采用分页存储管理方式，要求每个进程拥有一张页表，且进程的页表驻留在内存中

- SPOOLing系统是在主机控制下，通过通道把I/O工作脱机处理，SPOOLing不包括的程序是连接程序，包括的程序是预输入程序、作业调度程序、缓输出程序
- 计算机系统的下述机制中，

  Ⅰ. 库函数　　Ⅱ. 终端命令

  Ⅲ. GUI界面　　Ⅳ. 系统调用

  属于操作系统提供给用户的接口是终端命令、GUI界面、系统调用
- 会引起进程的饥饿问题的进程调度算法有优先级、短作业优先（SJF）
- 外部中断是可以屏蔽的中断，内部中断是不能屏蔽的。
  - 程序性中断、访管指令都属于内部中断。
  - 时钟中断和控制台中断是可以被屏蔽的，属于外部中断
- 操作系统的文件管理中，文件控制块（FCB）的建立是在调用creat()时
- 在多道程序设计中，道数限制要考虑的因素是内存容量 、设备数量
- 必然引起进程切换的中断
  - 进程自愿结束, exit()
  - 进程被强行终止;
  - 非法指令，越界，kill
- 可能引起进程切换的中断
  - 时钟
  - 系统调用
  - 输入输出中断
- 不属于强迫性中断的是访管中断、系统调用（属于自愿性中断）
- 进程切换伴随着系统栈的切换，发生进程切换时，下降进程的现场信息从系统栈中弹出，保存到下降进程的PCB中。上升进程的现场信息从上升进程的PCB中恢复
- 下列选项中，降低进程优先级的合理时机是进程的时间片用完
- 在多级中断系统中，多层嵌套中断的最内层中断处理结束后，无论该中断是强迫性中断还是自愿性中断，都不需要进程切换。
  - 当发生中断嵌套时，系统栈中保存的是中断处理程序的现场信息，所以最内层中断处理完毕后，恢复的是上一层中断的现场信息，而不需要进程切换。
- Hoare管程的处理方式是指从条件队列中被唤醒的进程继续执行，执行唤醒操作的进程进入到紧急等待队列。当它从紧急队列被唤醒后，继续执行管程内的其它代码
- 操作系统为实现多道程序并发，对内存管理可以采用多种方式，其中代价最小的是分区管理
- 在页式存储管理中，每个页表的表项实际上是用于实现访问内存单元
- 某系统用位示图管理内存，位示图定义为 char bitmap[400]。页框号为380对应bitmap的位置是 bitmap[47] 的第3位　400*8=3200　380/8=47……4（0，1，2，3）从0开始
- 在动态异长分区的存储分配算法中，能保证空闲区按地址均匀分布的分配算法Next Fit算法
- 采用段式存储管理的系统中，若地址用24 位表示，其中8 位表示段号，则允许程序每个逻辑段的最大相对地址是 2^16-1
- 假设虚拟页式存储管理采用工作集模型。如果在△周期内确定某进程的工作集大小为n，则n的含义是该进程在△周期内访问页面的个数
- 文件系统中，把FCB分为次部和主部的好处是提高文件的查找速度、可以实现文件连接

- 创建进程所必须的步骤是
  - 申请空白PCB（进程控制块）；
  - 为新进程分派内存资源；（不是调度程序）
  - 初始化PCB；
  - 将新进程插入就绪队列；
- 将文件控制块(FCB)划分为主部和次部的好处有：提高查找速度、减少文件目录所占空间、实现文件连接
- 实现虚拟设备采用的技术是SPOOLing
- 假设操作系统利用缓冲技术在进程与打印机之间通过软缓冲区实现向打印机的输出，则该缓冲区的结构是内存链式队列
- 在有n个进程共享一个互斥段，如果最多允许m个进程（m<n）同时进入互斥段，则信号量的变化范围是m-n~m
- UNIX中文件的物理结构是索引结构
- 文件共享的实现方式有共享说明、连接、公共目录
- 使用系统打开文件表的主要目的是提高对文件的检索速度
- 若多个进程共享同一个文件 F，在系统打开文件表中仅有一个表项包含 F 的属性
- 多个进程共享同一个文件 F，各进程有关F的表项中的读写指针可能不同
- 读写指针的值是文件的记录或字节的逻辑顺序号
- 在操作系统的I/O管理中，缓冲池管理中着重考虑的是实现进程访问缓冲区的同步
- 若系统中有n个并发进程涉及某个相同的变量A，则变量A的相关临界区是由n临界区构成
- 文件系统的主要目的是实现对文件的按名存取
- 银行家算法是死锁避免的方法之一（不是死锁预防）
- 若无进程处于运行状态，就绪队列和等待队列不一定均为空（就绪队列为空，可能等待队列中有进程在等待事件）
- 下列选项中,(中断装置)不属于操作系统提供给用户的可使用资源
- 设计实时操作系统必须首先考虑系统的实时性、可靠性
- 若程序正在试图读取某个磁盘的第200个逻辑块, 使用操作系统提供的系统调用接口
- 当操作系统完成用户请求的“系统调用”功能后,应使CPU从内核态转到用户态工作
- UNIX系统在用户态执行的是命令解释程序，缺页处理程序、进程调度程序、时钟中断处理程序都必须在核心态执行
- 下列选项中，不可能在用户态发生的是进程切换，系统调用、外部中断、缺页可以在用户态发生（进程切换完全由内核来控制）
- 通常, 用户进程被建立后，随着进程运行正常或不正常结束而撤销
- “最高响应比优先”、“最短作业优先”适用于作业调度，而不适于进程调度
- 常用的进程调度算法有先来先服务、优先数、时间片轮转及多级调度等
- 先来先服务捌度算法是“非抢占式”的：“优先数调度算法”可以是“非抢占式”的，也可以是“抢占式”的；“时间片轮转调度算法”是一种“抢占式”的
- 导致一个进程创建另一个进程的典型操作有4种：用户登录、作业调度、提供服务、应用请求
- 以下关于管程的叙述错误的是( )。

  A．管程是进程同步工具，解决信号量机制大量同步操作分散的问题

B．管程每次只允许一个进程进入管程

C．管程中的signal操作的作用和信号量机制中的V操作相同

D．管程是被进程调用的，管程是语法范围，无法创建和撤销

（管程的signal操作与信号量机制中的V操作不同，前者必须在wait操作之后）

- 在采用 spooling 技术的系统中，用户的打印数据首先被送到内存固定区域
- 为保证磁盘文件安全，需要对磁盘文件进行转储。假设系统对磁盘文件进行了3次转储之后，发生了磁盘数据丢失。下述对磁盘数据丢失进行恢复的正确论述是差分转储策略只需要2份转储磁带数据恢复磁盘数据
- 系统调用中断异常都是用户态转向内核态，而进程切换只能发生在内核态
- 父进程和子进程可以并发执行
- 进程间的线程共享的资源有堆、静态变量、全局变量、文件等公共资源，独占的资源有栈、寄存器
- 作业在执行中发生了缺页中断，经操作系统处理后，应让其执行被中断的后一条指令

  若一个用户进程通过read系统调用读取一个磁盘文件中的数据，则下列关于此过程的叙述中，正确的是：若该文件的数据不在内存，则该进程进入睡眠等待状态、请求read系统调用会导致CPU从用户态切换到核心态 （read只需使用open返回的文件描述符，并不使用文件名作为参数）