操作系统原理「通关指南」

名词解释

Mutual exclusion（互斥）：互斥也叫间接制约关系。当一个进程进入临界区使用临界资源时，另一个进程必须等待，当占用临界资源的进程退出临界区后，另一进程才运行去访问此临界资源，阻止对共享资源同时访问。

Process（进程）：进程是进程实体的运行过程，是程序的一次执行过程，是具有独立功能的程序在一个数据集合上运行的过程，是系统进行资源分配和调度的一个独立单位。

Thread（线程）：线程是”轻量级进程“，是进程中的一个实体，是被系统独立调度和分派的基本单位，是一个基本的 CPU 执行单元，也是程序执行流的最小单元，由线程 ID、程序计数器、寄存器集合和堆栈组成。线程自己不拥有系统资源，只拥有一点在运行中必不可少的资源，但它可与同属于一个进程的其他线程共享进程所拥有的全部资源。

Operating System（操作系统）：是一种运行在内核态的软件，是控制和管理整个计算机系统的硬件与软件资源，合理地组织、调度计算机的工作与资源的分配，进而为用户和其他软件提供方便接口与环境的程序集和。操作系统是计算机系统中最基本的系统软件。

Race Conditions（竞态条件）：由于两个或者多个进程竞争使用不能被同时访问的资源，使得这些进程有可能因为时间上推进的先后原因而出现问题，这叫做竞争条件。分为两类，互斥、同步。竞争条件指有两个或多个进程读写某些共享数据，而最后的结果取决于进程运行的精确时序。

Deadlock（死锁）：多个进程并发执行，出现进程对资源的竞争，每个进程都持有其他进程等待的资源，导致所有进程等待，形成僵局（互相等待），若无外力作用，这些进程都将无法向前推进。

System Calls（系统调用）：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。

Multiprogramming（多道程序设计）：多道程序设计是指在一台处理机上同时并发运行多个程序，即在一台处理机上有多个程序同时进入主存并发运行，宏观上并行，微观上串行交替运行。

Physical Address（物理地址）：物理地址是指出现在 CPU 外部地址总线上的寻址物理内存的地址信号，是地址变换的最终结果地址。

Critical Region（临界区）：访问临界资源的代码称为临界区。

Busy Waiting(忙等待)：在进程互斥访问时，当一个进程位于其临界区内时，其他试图进入临界区的进程都必须在进入区内连续空循环，占用 CPU。在实现 I/O 时，用户程序发出一个系统调用，内核将其翻译成一个对应设备驱动程序的过程调用。然后设备驱动程序启动 I/O 并在一个连续不断的循环中检查该设备，看该设备是否完成了工作。当 I/O 结束后，设备驱动程序把数据送到指定的地方，并返回。然后操作系统将控制返回给调用者，这种方式称为忙等待。

Buffer（缓冲器)：分为输入缓冲器和输出缓冲器，前者将外设送来的数据暂时存放，以便处理器将它取走；后者的作用是用来暂时存放处理器送往外设的数据。用来解决外设和处理器速度不匹配的问题。

I-nodes（i 结点）：UNIX 系统采用文件名和文件描述信息分开的方法，使文件描述信息单独形成一个称为索引节点的数据结构，简称 i 结点。在文件目录中每个目录项仅由文件名和指向该文件所对应的 i 结点的指针组成。

Monitors（管程）：管程可以看做一个软件模块，它定义了一个数据结构，将共享的变量和对于这些共享变量的操作封装起来，形成一个具有一定接口的功能模块，进程可以调用管程来实现进程级别的并发控制。任一时刻管程中只能有一个活跃进程。

Virtual Address（虚拟地址）：虚拟地址并不真实存在于计算机中。每个进程都分配有自己的虚拟空间，而且只能访问自己被分配使用的空间，

进程隔离，更好的保护系统安全运行。操作系统为应用程序提供的一个统一的内存访问接口。所有的应用程序只需要面向虚拟地址进行编写，而不用考虑实际的物理地址的使用情况。当应用程序使用虚拟地址访问内存时，处理器（CPU）会通过 MMU（内存管理单元）将其转化成物理地址，方便编程。

Interrupt（中断）：指处理机处理程序运行中出现的紧急事件的整个过程。程序运行过程中，系统外部、系统内部或者现行程序本身若出现紧急事件，处理机立即中止现行程序的运行，自动转入相应的处理程序(中断服务程序)，待处理完后，再返回原来的程序运行，这整个过程称为程序中断。中断分为外中断和内中断。外中断是指来自 CPU 执行指令外部的事件，通常用于信息输入/输出，比如 I/O 中断，时钟中断；内中断是指来自 CPU 执行指令内部的事件，入程序的非法操作码、地址越界……一旦出现，就应立即处理，不能被屏蔽。

Semaphore（信号量）：信号量（semaphore）是操作系统用来解决并发中的互斥和同步问题的一种方法。是在多线程环境下使用的一种设施，是可以用来保证两个或多个关键代码段不被并发调用。在进入一个关键代码段之前，线程必须获取一个信号量；一旦该关键代码段完成了，那么该线程必须释放信号量。其它想进入该关键代码段的线程必须等待直到第一个线程释放信号量。

Device Driver（驱动程序）：是一种可以使计算机和设备通信的特殊程序，可以说相当于硬件的接口，操作系统只能通过这个接口，才能控制

硬件设备的工作，假如某设备的驱动程序未能正确安装，便不能正常工作。

Relocation（重定位)：重定位就是把程序的逻辑地址空间变换成内存中的实际物理地址空间的过程，也就是说在装入时对目标程序中指令和数据的修改过程。

Atomic Action（原子操作)：是指不可被中断的一个或一系列操作，不会被线程调度机制打断的操作，这种操作一旦开始，就一直运行到结束。要么全部成功，要么全部失败。

Device Independence（设备独立性）：设备独立性是指操作系统把所有外部设备统一当作成文件来看待，只要安装它们的驱动程序，任何用户都可以像使用文件一样，操纵、使用这些设备，而不必知道它们的具体存在形式。

Avoiding Locks（避免锁）：避免锁，指对进程资源申请不加限制，但在分配之前会作安全检查，只有安全才进行分配。

Read-Copy-Update（RCU）：是一种同步机制，它的优点就是可以在更新的过程中，运行多个 reader 进行读操作。支持一个写操作和多个读操作同时进行。对于被 RCU 保护的共享数据结构，读者不需要获得任何锁就可以访问它，但写者在访问它时首先拷贝一个副本，然后对副本进行修改，最后使用一个回调（callback）机制在适当的时机把指向原来数据的指针重新指向新的被修改的数据。

简答题

1. 批处理方式已经出现几十年了，为什么现在还会有操作系统在使用？

系统吞吐量大，CPU 和其他资源保持“忙碌”状态

资源利用率高，多道程序共享计算机资源，从而使各种资源得到充分利用

用户可脱机工作

作业成批处理 ，批量改名，批量删除，脚本自动操作，定时计划任务，循环任务。很多地方都能用到批处理。

2. CPU 利用率越高，说明系统效率越高对不对？

不对，比如程序中出现死循环使得 CPU 利用率解决百分之百，无法处理其他请求，系统效率极低。

3. CPU 的管态和目态的区别，能不能从目态直接进入另一个应用程序的目态

如果程序处于管态，那么程序可以访问计算机的任何资源，它的资源访问权限不受限制，通常，操作系统在管态下运行。

目态又称为常态或者用户态，机器处于目态时，程序只能执行非特权指令，不能直接使用系统资源，也不能改变 CPU 的工作状态，并且只能访

问这个用户程序自己的存储空间。

能。

4. 操作系统为什么提出“作业、进程、线程”的概念？

作业：用户在一次解题或一个事物处理过程中要求计算机系统所做工作的集合。它包括用户程序、所需要的数据及控制指令等。作业是由一系列有序的步骤组成的。

进程：一个程序在一个数据集合上的一次运行过程。一个程序在不同数据集合上运行，乃至一个程序在同样数据集合上的多次运行都是不同的进程。

线程：线程是进程中的一个实体，是被系统独立调度和执行的基本单位。

作业是在早期的多道批处理系统中提出的，在现代操作系统中基本没有概念及应用。

引入进程的目的是更好地使多道程序并发执行，提高资源利用率和系统吞吐量

引入线程地目的使减小程序在并发执行时所付出地时空开销（上下文切换），提高操作系统地并发性能。

5. 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。

6. 理解一下系统调用 fork()的功能

fork 函数的主要作用是在父进程调用的基础上创建一个其的子进程。子进程是一个相对的概念，他会复制父进程的数据代码等部分，是完全复制一份而不是共用相同的变量相当于克隆。

fork 函数有一个特点就是只调用一次却会返回两次，一次是父进程返回的值 一个大于 0 的数 （即他创建的子进程的 PID，PID 是操作系统中进程的唯一标识 ），而另外一次则是子进程返回的值为 0。还有一种情况就是父进程调用该函数的时候如果返回的 <0 的数则说明创建进程失败（失败的原因有很多）。

另外一个值得注意的是调用 fork 函数 的时候子进程是接着该调用后的代码继续执行的，如果其后还存在调用 fork 函数 子进程也可以作为父进程创建它的子进程。

7. 为什么要中断？中断后需要保存什么信息

中断指处理机处理程序运行中出现的紧急事件的整个过程。程序运行过程中，系统外部、系统内部或者现行程序本身若出现紧急事件，处理机

立即中止现行程序的运行，自动转入相应的处理程序(中断服务程序)，待处理完后，再返回原来的程序运行。系统必须保存当前处理机程序状态字 PSW 和程序计数器 PC 等的值以及一些通用寄存器的值。

8. 什么是操作系统的内核

内核是指将操作系统的主要功能模块都作为一个紧密联系的整体运行在核心态，从而为用户程序提供高性能的系统服务。

9. 为什么建立管道文件？普通文件与管道文件的区别

管道文件是建立在内存之上可以同时被两个进程访问的文件。

向管道提供输入的发送进程，以字符流形式将大量的数据送入管道；而接收管道输出的接收进程从管道中接收数据。

实际上，管道是一个固定大小的缓冲区。

从管道读数据是一次性操作，数据一旦被读取，就释放空间以便写更多数据。管道只能采用半双工通信，即某一时刻只能单向传输。要实现父子进程互动通信，需定义两个管道。

10. C/S 与 B/S 的区别

C/S 架构，即 Client/Server(客户端/服务器)架构，是一个典型的两层架构。开发比较容易，操作简便，但应用程序的升级和客户端程序的维护较为困难。C/S 是建立在局域网的基础上的；

1. 优点：

   （1）安全性：需要其特定的客户端，所以面向对象比较确定，将所进行的信息安全处于一个可控的范围

   （2）效率：客户端的服务器直接相连，省却了中间环节，数据的传输比较快

   （3）个性化：有特定的客户端，所以可以在较大程度上满足客户的个性化要求

   （4）稳定性：结构比较稳定，有较强的事务处理能力，可以实现较复杂的业务逻辑

2. 缺点：

   （1）特定的客户端：对 pc 机有一定的要求，如：操作系统，并且它就像订在墙上的石头桌子，不可再利用

   （2）中间环节：因为省却了中间环节，所以当客户端达到一定的量时，同时访问服务器，造成服务器的相应变慢，效率变低

B/S 架构，即 Brower/Server(浏览器/服务器)架构。它由逻辑上相互分离的表示层、业务层和数据层构成。B / S 系统统一了客户端，无需特殊安装，拥有 Web 浏览器即可；它将系统功能实现的核心部分集中到服务器上，简化了系统的开发、维护和使用。B/S 是建立在广域网的基础上的。

1. 优点：

   （1）范围：零安装，拥有一个浏览器，即可访问，面向的范围更

广

（2）维护性：维护简单，更新页面，即可实现面向所有用户的更新

（3）共享性：通过浏览器访问，共享性强，就像买来的餐桌，可以再利用

2. 缺点：

（1）安全性：面向的范围广，所以安全性比较低

（2）个性化：因为面型的范围广，所以它是一种公共审美，无法满足个性化的需求

11. 进程创建时是什么状态

就绪态

12. 进程结束时是什么状态

终止态

13. 子进程与父进程的关系

子进程为由另外一个进程（对应称之为父进程）所创建的进程。子进程继承了父进程的大部分属性，例如文件描述。

子进程得到的是除了代码段是与父进程共享的以外，其他所有的都是得到父进程的一个副本，子进程的所有资源都继承父进程，得到父进程资源的副本，既然为副本，也就是说，二者并不共享地址空间。

一个进程可能下属多个子进程，但最多只能有 1 个父进程，而若某一进程没有父进程，则可知该进程很可能由内核直接生成。

孤儿进程：一个父进程退出，而它的一个或多个子进程还在运行，那么那些子进程将成为孤儿进程。 孤儿进程将被 init 进程(进程号为 1)所收养，并由 init 进程对它们完成状态收集工作。

僵尸进程：一个进程使用 fork 创建子进程，如果子进程退出，而父进程并没有调用 wait 或 waitpid 获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中。 这种进程称之为僵死进程。

14. 进程的状态与中断

处于运行态的进程如果需要等待某一事件完成，如 I/O，则会进入阻塞队列等待，变为阻塞态，释放处理机。如果事件完成，该进程从阻塞队列中移出，进入就绪队列，修改状态为就绪态，重新等待处理机资源。

15. 进程之间的关系

不同的系统中，进程的关系不同，在 Windows 系统中，父子进程是平等的，在 UNIX 系统中，进程和它的子进程组成一个进程组，父进程可以杀死子进程。但进程之间都具有同步与互斥关系，不同进程相互配合可以提高效率，但不同进程也会争夺同一个资源。

16. 进程的现场信息包括什么，可能存放的位置

现场信息包括，通用寄存器的内容、控制寄存器（如 PSW 寄存器）的内容、用户堆栈指针、系统堆栈指针，程序计数器（PC），程序状态字（PSW）

存放位置：进程栈（不是系统栈）

17. 什么是自愿性中断和强迫性中断？

强迫性中断事件：包括硬件故障中断，程序性中断，外部中断和输入输出中断等。

自愿性中断事件：是由正在运行的进程执行一条访管指令用以请求系统调用而引起的中断，这种中断也称为"访管中断"。

自愿中断的断点是确定的，而强迫性中断的断点可能发生在任何位置。

18. 进程创建时是什么状态？进程终止时是什么状态？为什么？

进程创建时是就绪态，进程终止时是终止态

就绪状态：当进程已分配到除处理器(CPU)以外的所有必要资源后，只要再获得处理器就可以执行的状态称为就绪状态。在一个系统里,可以有多个进程同时处于就绪状态，通常把这些就绪进程排成一个或多个队列，称为就绪队列。

终止状态：当一个进程已经正常结束或异常结束，操作系统已将其从系统队列中移出，但是，尚未撤消，这时称为终止状态。

19. 系统栈 用户栈 PCB 在内存什么位置存放？

系统栈在内存的操作系统空间的一块区域

用户栈是用户进程空间的一块区域

pcb 在系统内存的一段连续内存中，存在于所有进程共享的内核空间中，

20. 作业、进程、线程的状态有哪几种？

一个作业从进入系统到它运行结束，一般要经历 4 个状态，分别是进入状态、后备状态、运行状态和完成状态。

在三态模型中，进程状态分为三个基本状态，即运行态，就绪态，阻塞态。

线程包括新建状态，就绪状态，运行状态，阻塞状态，死亡状态。

**21. 作业、进程、线程之间的关系？ **

作业是从用户角度出发的，它是由用户提交，以用户任务为单位，操作系统没有作业概念

进程是从操作系统出来的，它是由系统生成，是操作系统的资源分配和独立运行的基本单位，进程是线程的载体

线程是进程的基本执行单元，是操作系统调度和分派的基本单位，是一个基本的 CPU 执行单元，也是程序执行流的最小单元

通常指一个进程就叫一个作业。

22. 进程现场信息存放的数据结构有哪些？什么情况该数据结构会被使用？

在计算机系统中，对于每个资源和每个进程都设置了一个数据结构，用于表征其实体，称为资源信息表或进程信息表，其中包含了资源或进程的标识，描述，状态等信息以及一批指针。通过这些指针，可以将同类资源或进程的信息表，或者同一进程占有的资源信息表分类链接成不同的队列，以便于操作系统进行查找。分为四类：内存表，设备表，文件表和用于进程管理的进程表(通常又称为进程控制 PCB）。

当发生中断，创建进程，实现进程间的通讯时会被使用

23. 进程 PCB 为什么要分为 P 区和 U 区？

UNIX 系统把进程控制块分成两部分。一部分为进程的基本控制块，简称 proc 结构，它存放着进程最常用的一些信息（进程标识符(PID)，用户标识符(UID)，进程状态…），所以 proc 结构一般常驻内存。将那些只在进程运行时才用到的控制信息存储在 U 区（user 结构），从而使这部分信息不必常驻内存，节省内存空间的占用。

24. 有了进程机制后，为什么提出线程机制?

1.

进程在同一时间只能干一件事

2.
3.

进程在执行的过程中如果阻塞，整个进程就会挂起，即使进程中有些工作不依赖于等待的资源，仍然不会执行。

4.
5.

进程上下文切换开销大

6.

因此，操作系统引入了比进程力度更小的线程，作为并发执行的基本单位，从而减少程序在并发执行时所付出的时空开销，提高并发性

25. 线程机制的优点

1. 调度：线程是独立调度的基本单位，线程切换的代价远低于进程。在同一进程中，线程的切换不会引起进程切换。
2. 并发性：在引入线程的操作系统中，不仅进程之间可以并发执行，而且一个进程中多个线程也可以并发执行，甚至不同进程中的线程也能并发执行，使得操作系统有更好的并发性，提高资源利用率和吞吐量。
3. 拥有资源：线程不拥有系统资源，但可以访问隶属进程的系统资源，线程切换时空开销小。

4. 系统开销：线程创建和撤销不涉及进程控制块，地址空间，内存空间等资源，开销小。进程切换设即进程上下文切换，而线程切换只需要保存和设置少量寄存器的内容，开销小。
5. 同步和通信：隶属于同一进程的线程共享进程地址空间和数据端、代码段，线程之间同步和通信容易实现，甚至不需要操作系统干预。
6. 支持多处理机系统：对于多线程进程，可以将进程中的多个线程分配到多个处理机上执行。

26. 用户级线程与系统级的线程的区别（优缺点）

用户级线程：

- 优点
    - 线程切换不需要转换到内核空间，节省了模式切换的开销。
    - 调度算法可以是进程专用的，不同的进程可以根据需要，对自己的线程选择不同的调度算法。
    - 用户级线程的实现于操作系统平台无关，对线程管理的代码是属于用户程序的一部分。
- 缺点
    - 当线程执行一个系统调用时，不仅该线程被阻塞，而且进程内的所有线程都被阻塞。

- 不能发挥多处理机的优势，内核每次分配给进程仅有一个CPU，因此进程中仅有一个线程能执行。

内核级线程

- 优点
  - 能充分发挥多处理机优势，内核能同时调度同一进程中多个线程并行执行。
  - 如果进程中一个线程被阻塞，内核可以调度该进程中的其他线程占用处理机，也可以运行其他进程中的线程。
  - 内核支持线程具有很小的数据结构和堆栈，线程切换比较快、开销小。
  - 内核本身也可以采用多线程技术，可以提高系统的执行速度和效率。
- 缺点
  - 同一进程中的线程切换，需要从用户态到核心态进行，系统开销大。
  - 线程的创建、撤销、调度在操作系统核心态，占用系统资源，增加系统开销。

27. 说明混合机制线程的实现过程

内核级线程和多少个用户级线程彼此多路复用

彼此多路复用的方法有

多对一，多个用户级线程映射到一个内核级线程；

- 优点：线程管理是在用户空间进行的，效率高。
- 缺点：如果有一个车线程在访问内核时发生阻塞，则整个进程阻塞；在任何时刻只能有一个线程访问内核，多个线程不能同时在多个处理机上运行。

一对一，一个用户级线程映射到一个内核级线程；

- 优点：一个线程阻塞可以调度另一个线程运行，并发能力强。
- 缺点：每创建一个用户线程，相应地就需要创建一个车内核线程，开销大。

多对多，多个用户级线程映射到多个内核级线程;

- 特点：既克服了多对一模型并发度不高地缺点，又克服了一对一模型一个用户进程占据太多内核线程而开销大的缺点。此外还具有上述两种模型各自的优点。

28. 什么是上行调度

当一个线程被阻塞之后，该线程会发送信号通知内核，而当内核了解到该线程被阻塞后，就会通知该进程的运行时系统，并且由该系统去调用用户自己编写的线程调度程序去解决阻塞的线程

29. 为什么早期把线程也叫轻量级进程？

早期没有线程的概念，进程被认为是程序执行的最小单位。而当后来提出的线程概念时，线程拥有许多进程的性质，但资源往往比进程要少，被叫做轻量型进程。

30. 如何理解进程进入等待状态，而该进程的线程还在运行？

进程内部可能包含多个线程，进程与进程之间不能共享数据，而一个进程中的多个线程可以共享进程的内存区域，如堆，方法区，线程的栈是私有的。当进程宏观上进入等待状态时，进程内部的一些轻量级的线程如果不涉及到对外部数据的访问，只是进行内部数据处理，如加减乘除运算，则依旧可以运行，故实际在微观上仍然有内部的线程正在运行。

31. 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

32. 例题中的购票任务是互斥关系 还是同步关系？

互斥

33. 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。

34. 忙式等待与排队等待的区别？

忙式等待是指有一个进程在使用临界资源时，其他进程也想使用临界资源，需要不停的执行循环判断语句（连续空循环）来判断临界资源是否空闲，等待过程需要不断使用 CPU。

排队等待是值当临界资源被占用时，其他想使用的进程被挂起到等待队列中等待唤醒，等待过程不占用 CPU

35. 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。出现时间上错误。

36. 同一公共变量的临界区的数量多少？

取决于访问该公共变量的代码段有多少

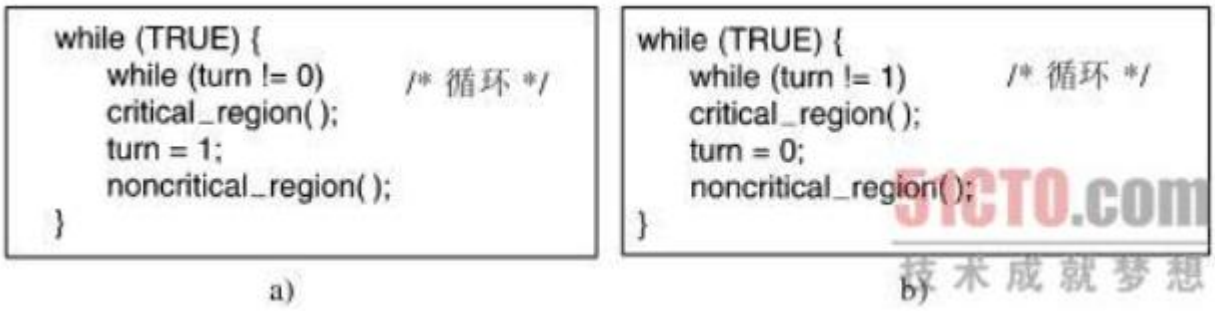
取决于多少个进程想要访问临界资源

37. 购票任务中加上锁变量 S 后，为什么还会出现与时间有关的错误？什么情况下会出现错误？

一个进程判断完锁变量为 0 之后还未设为 1，时间片用尽转去执行另一个进程，这个进程锁变量设为 1，之前的进程再次运行时锁变量也为 1，这个时候有两个进程进入临界区

38. 严格轮转算法中 turn=0 代表着什么 ？

严格轮转法：整型变量 turn，初始值为 0，用于记录轮到哪个进程进入临界区，并检查或更新共享内存。开始时，进程 0 检查 turn，发现其值为 0，于是进入临界区。进程 1 也发现其值为 0，所以在一个等待循环中不停地测试 turn，看其值何时变为 1。连续测试一个变量直到某个值出现为止，称为忙等待（busy waiting）。由于这种方式浪费 CPU 时间，所以通常应该避免。



表示 0 号进程可以进入临界区，其他进程不能进去临界区并且在临界区外做忙等待。

39. 为什么说严格轮转算法可以实现临界区的互斥使用，但无法保证公平性？

严格轮转法设置了一个 turn 变量，当进程进入临界区时会将 turn 的值改变，从而将后续进程阻塞在 while 循环当中，但是每个进程进入临界区的权限只能由另一个进程赋予，当前一个进程一直处于运行状态时，其余进程就要一直等待前一进程执行结束主动退出临界区后再进入临界区，无法保证公平性。该进程一直在运行，其他进程就得一直等待。

40. 说明 Peterson’s 算法中，为什么不会出现两个进程同时进入临界区的情况？

```
#define FALSE 0
#define TRUE  1
#define N     2                                          /* 进程数量 */

int turn;                                                /* 现在轮到谁？*/
int interested[N];                                       /* 所有值初始化为 0
（FALSE）*/

void enter_region(int process)                           /* 进程是 0 或 1 */
```

```
{
        int other;                                          /* 其他进程号 */
        other = 1 - process;                                /* 另一方进程 */
        interested[process] = TRUE;                         /* 表名所感兴趣
的 */
        turn = process;                                     /* 设置标志 */
        while(turn == process && interested[other] ==TRUE);   /* 空语句 */
}

void leave_region(int process)                          /* 进程：谁离开？*/
{
        interested[process] = FALSE;                        /* 表示离开临
界区 */
}
```

一开始，没有任何进程处于临界区中，现在进程 0 调用 enter_region。它通过设置其数组元素和将 turn 置为 0 来标识它希望进入临界区。由于进程 1 并不想进入临界区，所以 enter_region 很快便返回。如果进程 1 现在调用 enter_region ，进程 1 将在此处挂起直到 interested[0] 变成 FALSE，该事件只有在进程 0 调用 leave_region 退出临界区时才发生。

现在考虑两个进程几乎同时调用 enter_region 的情况。它们都将自己的进程号存入 turn，但只有后被保存进去的进程号才有效，前一个因被重写而丢失。假设进程 1 是后存入的，则 turn = 1.当两个进程都运行到 while 语句时，进程 0 将循环 0 次并进入临界区，而进程 1 则将不停的循环且不能进入临界区，知道进程 0 退出临界区为止。

41. 面包店算法中，什么情况下可能为出现两个或以上的进程抓到相同的号码？

42. 面包店算法中，变量 i 和 j 的含义？

43. 面包店算法中定义的 A[N]作用是什么？

```
VAR A[N]: Boolean;（false)
    number[N]: integer; (0)
Pi 进入：
A[i]=true;  // 进程 i 置标记
number[i]=max{number[0],…,number[n-1]}+1;  // 获取最大号码
A[i]=false; // 撤销标记
For (j=0; j< n; j++) {  // 遍历 n 个进程
    While (A[j]);       // 当有进程正在获取号码，阻塞
    // 如果进程 i 的号码小，则进程 j 进入临界区等待进程 i 结束而阻塞（号码小的优先级高）
    While ((number[j]!=0) && (number[j],j)<(number[i],i));
}
/*(a,b)<(c,d) iff (a<c)or(a=c and b<d)*/
// 先比较号码，如果号码相同再比较进程号（按字典序）
```

Lamport 把这个并发控制算法可以非常直观地类比为顾客去面包店采购：

已知有 n 位顾客要进入面包店采购，安排他们按照次序在前台登记一个签到号码。该签到号码逐次加 1。

根据签到号码的**由小到大**的顺序依次入店购货。

完成购买的顾客在前台把其签到号码归 0.。如果完成购买的顾客要再次进店购买，就必须重新排队。

同时进入面包店采购的两个或两个以上的顾客有可能得到相同的号码。多个顾客如果抓到相同的号码,,则规定按照顾客名字的字典次序进行排序， 这里假定顾客是没有重名的。

在计算机系统中, 顾客就相当于线程，每个进程有一个唯一的标识，而入店购货就是进入临界区独占访问该共享资源。由于计算机实现的特点，存在两个线程获得相同的签到号码的情况，这是因为两个线程几乎同时申请排队的签到号码，这两个线程读到的号码是完全一样的。所以，该算法规定如果两个线程的排队签到号码相等，则线程 id 号**较小的具有优先权**。

1.

当几个进程同时取号时，在 number 修改之前，都调用了 Max，会使抓号相同，但是会在运行时按照字典序执行

2.

i 为要取号进程编号，j 为其他进程编号

3.

记录哪些进程正在抓号或已经抓号，记录想要进入或者已经进入临界区的进程编号。

4.

44. 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。

45. 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。

46. 实现列车发车过程控制的公平效率调度算法

```
semaphore s = 1;  // 控制车道上只能由一个方向的列车，使得轨道上同一时间可以有多个同方向火车运行
semaphore sa = 1; // 用于 A 方向列车控制 xa 相关的临界区
semaphore sb = 1; // 用于 A 方向列车控制 xb 相关的临界区
int xa = 0; // 正在铁轨运行的 A 到 B 方向列车的数量
int xb = 0; // 正在铁轨运行的 B 到 A 方向列车的数量
// A 站发车控制
A() {
    P(sa);
    if (xa == 0)   P(s); // 如果车道上没上有车，来了第一个 A 到 B 方向列车，则车道都归 A 到 B 方向列车所有，获取锁。
    Xa = xa + 1;
    V(sa);
    A 站发车
    火车运行
    到达 B 站
    P(sa);
    Xa=xa-1;
    if (xa == 0)  V(s);  // 如果车道上没有 A 到 B 方向列车了，释放锁。
    V(sa);
}
// B 站发车控制
B() {
    P(sb);
    if(xb == 0)  P(s); // 如果车道上没上有车，来了第一个 B 到 A 方向列车，则车道都归 B 到 A 方向列车所有，获取锁。
    Xb = xb+1;
    V(sb);
    B 站发车
    火车运行
```

```
    到达 A 站
    P(sb);
    Xb=xb-1;
    if (xb == 0)  V(s);  // 如果车道上没有 B 到 A 方向列车了，释放锁。
    V(sb);
}
```

47. 读者/写者问题中的效率调度为什么是读者优先方式的？

48. 实现读者/写者问题中的写者优先算法

- 读优先

```
semaphore wMutex;         // 控制写操作的互斥信号量，初始值为 1
semaphore rCountMutex;    // 控制对 Rcount 的互斥修改，初始值为 1
int rCount = 0;           // 正在进行读操作的读者个数，初始化为 0

// 写者进程/线程执行的函数
void writer()
{
    while(TRUE)
    {
        P(wMutex);  // 进入临界区
        write();
        V(wMutex);  // 进入临界区
    }
}

// 读者进程/线程执行的函数
void reader()
{
    while(TRUE)
    {
        P(rCountMutex);  // 进入临界区
        if ( rCount == 0 )
        {
            P(wMutex);   // 如果有写者，则阻塞写者
        }
        rCount++;          // 读者计数 + 1
        V(rCountMutex);    // 离开临界区

        read();            // 读数据

        P(rCountMutex);    // 进入临界区
        rCount--;          // 读完数据，准备离开
        if ( rCount == 0 )
        {
            V(wMutex);     // 最后一个读者离开了，则唤醒写者
        }
        V(rCountMutex);    // 离开临界区
    }
}
```

读操作可以并发进行，而写操作之间或写与读之间是互斥的，故读者优先效率更高

- 写优先

```
semaphore rCountMutex;  // 控制对 Rcount 的互斥修改，初始值为 1
semaphore rMutex;       // 控制读者进入的互斥信号量者，初始值为 1

semaphore wCountMutex   // 控制 wCount 互斥修改，初始值为 1
semaphore wDataMutex    // 控制写者写操作的互斥信号量，初始值为 1

int rCount = 0;         // 正在进行读操作的读者个数，初始化为 0
int wCount = 0;         // 正在进行读操作的写者个数，初始化为 0

// 写者进程/线程执行的函数
void writer()
{
    while(TRUE)
    {
        P(wCountMutex);   // 进入临界区
        if ( wCount == 0 )
        {
            P(rMutex);  // 当第一个写者进入，如果有读者则阻塞读者
        }
        wCount++;         // 写者计数 + 1
        V(wCountMutex);  // 离开临界区

        P(wDataMutex); // 写者写操作之间互斥，进入临界区
        write();       // 写数据
        V(wDataMutex); // 离开临界区

        P(wCountMutex);   // 进入临界区
        wCount--;         // 写完数据，准备离开
        if ( wCount == 0 )
        {
            V(rMutex); // 最后一个写者离开了，则唤醒读者
        }
        V(wCountMutex);   // 离开临界区
    }
}

// 读者进程/线程执行的函数
void reader()
{
    while(TRUE)
    {
        P(rMutex);
        P(rCountMutex);   // 进入临界区
        if ( rCount == 0 )
        {
            P(wDataMutex);   // 当第一个读者进入，如果有写者则阻塞写者写操作
        }
        rCount++;
        V(rCountMutex);   // 离开临界区
        V(rMutex);

        read();           // 读数据

        P(rCountMutex);  // 进入临界区
        rCount--;
        if ( rCount == 0 )
```

- 读写公平

```
semaphore rCountMutex;  // 控制对 Rcount 的互斥修改，初始值为 1
semaphore wDataMutex    // 控制写者写操作的互斥信号量，初始值为 1
semaphore flag;         // 用于实现公平竞争，初始值为 1
int rCount = 0;         // 正在进行读操作的读者个数，初始化为 0



// 写者进程/线程执行的函数
void writer()
{
    while(TRUE)
    {
        P(flag);
        P(wDataMutex); // 写者写操作之间互斥，进入临界区
        write();       // 写数据
        V(wDataMutex); // 离开临界区
        V(flag);
    }
}

// 读者进程/线程执行的函数
void reader()
{
    while(TRUE)
    {
        P(flag);
        P(rCountMutex);   // 进入临界区
        if ( rCount == 0 )
        {
            P(wDataMutex);   // 当第一个读者进入，如果有写者则阻塞写者写操作
        }
        rCount++;
        V(rCountMutex);   // 离开临界区
        V(flag);

        read();           // 读数据

        P(rCountMutex);  // 进入临界区
        rCount--;
        if ( rCount == 0 )
        {
            V(wDataMutex);  // 当没有读者了，则唤醒阻塞中写者的写操作
        }
        V(rCountMutex); // 离开临界区
    }
}
```

49. 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

50. 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

51. 用管程机制实现火车站的发车调度问题

建立两个锁分别去控制火车的进和出。当火车进时获得进入锁，到达后再释放。

不符合条件的火车进入等待车道上的其他火车离开。

其他火车离开后发送信号，解开等待的状态，等待中的车进入车道，并释放进入锁。

52. 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

53. 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

54. 进程(或线程）的优先级为什么要动态变化？

在不同的调度策略下，（比如短作业优先），可能会使长作业出现“饥饿”现象，而作业优先级的动态变化可以使得作业优先级比较低的作业也能够获得处理机，保证了作业处理的公平。

55. CPU 调度时机？

从就绪态 -> 运行态：当进程被创建时，会进入到就绪队列，操作系统会从就绪队列选择一个进程运行；

从运行态 -> 阻塞态：当进程发生 I/O 事件而阻塞时，操作系统必须选择另外一个进程运行；

从运行态 -> 结束态：当进程退出结束后，操作系统得从就绪队列选择另外一个进程运行；

从阻塞态 -> 就绪态：当进程等待的事件完成后发出中断，进程移出等待队列，进入就绪队列，状态变为就绪态，重新参与调度。

56. 什么是周转时间？

从一个批处理作业**提交**时刻开始直到该作业**完成**时刻为止的统计平均时间

57. 影响执行优先级的因素有哪些？

CPU 利用率，吞吐量，周转时间，等待时间，响应时间，均衡性，公平性

58. 什么是抢占式调度？抢占式调度的优缺点

抢占式调度算法挑选一个进程，然后让该进程只运行某段时间，如果在该时段结束时，该进程仍然在运行时；或如果有更高优先级的进程出现，则会把它挂起，接着调度程序从就绪队列挑选另外一个进程或更高优先级进程。这种抢占式调度处理，需要在时间间隔的末端发生时钟中断，以便把 CPU 控制返回给调度程序进行调度，也就是常说的时间片机制。

优点：兼顾长短作业，公平性好，提高系统吞吐量

缺点：进程上下文切换耗时大，降低了 CPU 效率，中断次数多，系统开销大

59. 与时间有关的错误、死锁、饥饿产生的原因？

与时间有关的错误：并发进程同时使用共享资源

死锁：进程集合中的每一个进程都在等待另一个死锁的进程占有的资源

饥饿：系统不能保证某个进程的等待时间上界，从而使该进程长时间等待，当等待时间给进程推进和响应带来明显影响时，称发生了进程饥饿。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。

60. 死锁的四个条件是什么？什么情况下是必要条件？什么情况下是充分必要条件？

1.

互斥条件：在一段时间内某资源仅为一个进程所占有。

2.
3.

占有和等待（请求并保持）条件：进程已经保持了至少一个资源，但又提出了新的资源请求，而该资源已被其他进程所占有，此时请求进程被阻塞，但对自己已获得的资源保持不放。

4.
5.

不可抢占（剥夺）条件：进程所获得的资源在未使用完之前，不能被其他进程强行夺走，即只能由获得该资源的进程自己来释放。

6.
7.

环路等待条件：存在一种进程资源的循环等待链，链中每个进程已获得的资源同时被链中下一个进程所请求。

8.

在任何情况下这四个条件都是必要条件

注意：4 只是死锁的必要条件

只有当系统中每类资源只有一个，资源分配图含圈（环路等待）就变成了死锁的充分必要条件。

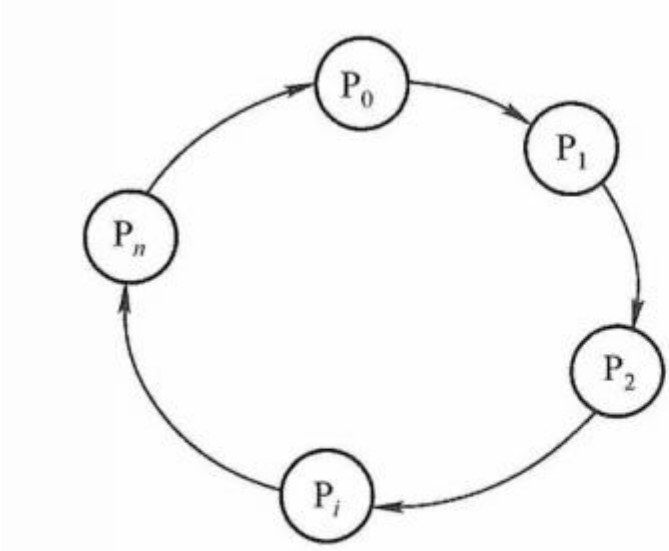


图 2.11　循环等待

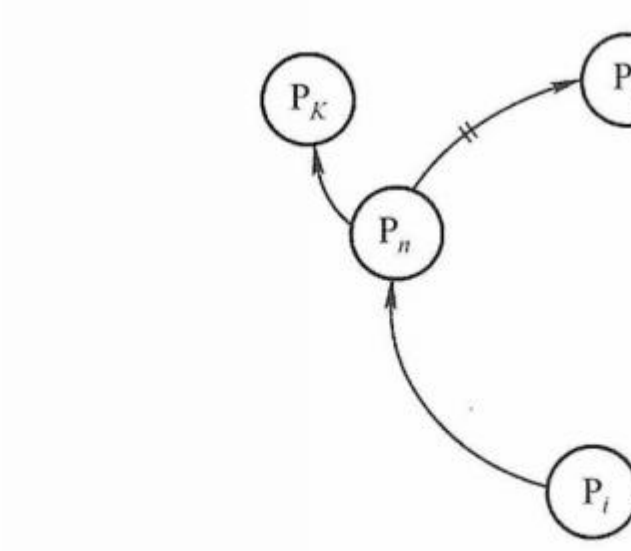


图 2.12　满足条

61. 怎样能避免死锁？

避免死锁的方法中，运行进程动态地申请资源，但系统在进行资源分配之前，应先计算此次分配的安全性。若此次分配不会导致系统进入不安全状态，则允许分配，否则让进程等待。

62. 举例说明如何避免死锁？（如何破坏四个死锁条件）

破坏互斥条件：使用假脱机技术。使资源不被一个进程独占，避免分配不是绝对必需的资源，尽量做到尽可能少的进程真正请求资源。

破坏不剥夺条件：当一个已经保持某些不可剥夺资源的进程请求新的资源而得不到满足时，它必须释放已经保持的所有资源，等待以后需要的时候再重新申请。

破坏占有和等待条件：采用预先静态分配方法，即进程在运行前一次申请完它所需要的全部资源，在它的资源为满足前，不把它投入运行。一旦投入运行，这些资源就一直归它所有，不在提出其他资源请求。

破坏环路等待条件：顺序资源分配法，资源编号，每个进程必须按编号递增的顺序请求资源，同类资源一次申请完。

63. 操作系统的存储管理应该实现的功能是什么？

-

内存空间的分配与回收

-
-

地址转换

-
-

内存空间的扩充

-
-

内存共享

-

-

存储保护

-

64. 系统空闲空间管理时，空闲块应该按照什么排序合理？为什么？

不同的管理算法有着不同的排序规则。

最先适应算法要求按地址递增的顺序排列

最优适应算法要求把空闲区按长度递增次序排

最坏适应分配算法要求把空闲区按长度递减的次序排列

65. 什么是重定位？逻辑地址与物理地址的关系？

重定位就是把程序的逻辑地址空间变换成内存中的实际物理地址空间的过程。

物理地址是指出现在 CPU 外部地址总线上的寻址物理内存的地址信号，是地址变换的最终结果地址。

逻辑地址是程序产生的与段相关的偏移地址。

物理地址可通过逻辑地址加段内偏移量得出。

66. 什么是局部性原理？

- 时间局部性：程序中每条指令一旦执行，不久后该指令可能再次执行；某数据被访问过，不久后该数据可能再次被访问。产生原因是程序中存在着大量的循环操作。
- 空间局部性：一旦程序访问了某个存储单元，在不久后，其附近的存储单元也将被访问，即程序在一段时间内可能再次被访问，可能集中在一定的范围之内，因为指令通常是顺序存放、顺序执行的，数据也一般是以向量、数组、表等形式簇聚存储的。

67. 使用 TLB（快表）的目的是什么？快表项越多越好吗？

如果页表全部放在内存中，则存取一个数据或一个条指令至少要访问两次内存：第一次是访问页表，确定所存取的数据或指令的物理地址；第二次是根据该地址存取数据或指令。显然，这种方法比通常执行指令的速度慢了一半。为此，在地址变换机制中增设一个具有并行查找能力的高速缓冲存储器——快表，用来存放当前访问的若干页表项，以加速地址变换的过程。使用快表将页表项加入缓存中可以在命中情况下只访问一次内存空间。

68. 倒置页表与进程页表的优缺点

反置页表虽然减少了内存的浪费，但是增加了查询时间并且增加了共享内存的难度。

进程页表消耗空间更大，但是从虚拟地址到物理地址的访问较为便捷，只需要从当前虚拟索引查找物理地址

69. 动态分区与静态分区的特点

- 静态分区（固定分区）将用户内存空间划分为若干固定大小的区域，每个分区只装入一道作业。当有空闲分区时，再从外存的后备作业队列中选择适当大小的作业装入该分区。存在两个问题，一是程序太大可能无法放入任何一个分区，需要使用覆盖技术使用内存空间；二是程序小于固定分区大小时，也要占用一个完整的内存空间，这样分区内存就存在空间浪费，即内部碎片。
- 动态分区根据进程实际需要，动态为之分配内存，并使分区的大小正好适合进程的需要。不会产生内部碎片，可能产生外部碎片。

70. 页式管理中 CPU 执行时需要访问内存多少次？为什么

如果快表命中，则只需要访问一次，否则需要访问两次，第一次查询内存中的页表，第二次访问相应内存空间 。n 层页表需要访问 n+1 次内存

71. 文件控制信息 FCB 为什么要分为主部和次部？

(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字相比较。 由于文件目录是存于外存的，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，这样读入内存作比较的样本也少，就不容易查找成功，就需要不断从外存更新信息到内存，导致查找速度较慢。

而将 FCB 分为两部分之后，文件目录中仅保存 FCB 的次部的话相同大小的外存块就可容纳较多的 FCB，从而大大地提高了文件的检索速度。

(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字和一个标识文件主部的文件号

主部则包括除文件名字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。 相当于就不用来一个新用户建立访问就保存一遍链接内容了，实现了文件目录公共部分的最大化利用。当连接计数的值为 0 时，表示一个空闲未用的 FCB 主部。

72. 文件共享的目的

文件共享使多个用户共享同一个文件，系统中只需保留该文件的一个副本。若系统不能提供共享功能，则每个需要该文件的用户都要有各自的副本，会造成对存储空间的极大浪费。

节约空间，减少额外存储空间的消耗，避免两个用户访问不同的文件信息

73. 目录文件的系统调用与普通文件的系统调用不同点

目录文件的删除操作会删除该目录文件下的所有文件，目录文件系统调用额外含有 opendir，closedir，readdir，link 和 unlink 等目录特有的系统

调用，而普通文件则含有 open，read，close，write，append，seek 等专门针对文件内部基本内容的操作

74. 实现文件系统所需要的基本数据结构有哪些？

iNode，目录文件表，空闲块表，系统打开文件表，用户打开文件表，每个进程的打开文件表，超级块（superblock），主引导记录（MBR），引导块（boot block），

75. 系统打开文件表可以不使用吗？为什么？

不可以。当有多个进程共享文件时，如果其中一个进程需要关闭文件，此时需要通过系统打开文件表判断该文件是否共享来决定是否真正的关闭文件，即需要对打开文件的进程进行计数。

75. 文件系统中如何提高读写效率？提高后会出现什么问题？

进行块大小的优化，块越大花费时间越少，但太大容易浪费空间

增大文件块大小，会使得磁盘空间利用率降低

76. 陷阱与中断的区别

陷阱（Trap）是一种事先安排的“异常”事件，由处理机正在执行的现行指令引起，用于在用户态下调用操作系统内核程序，如条件陷阱指令。

中断也称为外中断，是指来自 CPU 执行指令外部的事件，通常用于信息输入/输出。

陷阱通常由处理器正在执行的现行指令引起，而中断则是由与现行指令无关的中断源引起的。陷阱处理程序提供的服务为当前进程所用，而中断处理程序提供的服务则不是为了当前进程的。

77. 函数调用与中断的关系

中断：

1）何时发生可能是不可预料的； CPU 正在处理某件事情的时候，外部发生的某一事件（如一个电平的变化，一个脉冲沿的发生或定时器计数溢出等）请求 CPU 迅速去处理，于是 CPU 暂时中止当前的工作，转去处理所发生的事件。中断服务处理完该事件以后，再回到原来被中止的地方继续原来的工作

2） 对于可剥夺型内核，完成中断调度后，不是立刻返回原来的执行点执行，而是 回到就绪态优先级最高的任务开始运行。

3） 没有返回值；

4）不能传递参数。

函数：

1）何时发生是可预料的；

2）完成函数调度后会返回到原来的代码片段继续执行；

3）有返回值；

4）能传递参数。

相同点：

1）在执行调度之前，都保存现场；

2）都是调用一个 subcode。

78. I/O 软件的功能

实现设备独立性、统一命名、错误处理、同步与异步传输、缓冲 、驱动程序实现进程与设备控制器之间的通信

79. 中断处理的过程

1. 中断请求，在中断请求被响应之前会一直发送中断请求。
2. 中断源识别，当系统同时有多个中断源发出的中断请求时，系统往往只能相应并处理一个中断，这就要求 CPU 对来到的中断请求进行判优，选择出同一时间优先级最高的给予响应和处理
3. 中断响应，中断响应时，CPU 要向中断源发出中断响应信号
4. 中断处理，保护软件现场（把中断服务子程序中要用到的寄存器的内容压入堆栈）、开中断（为了可以嵌套）、执行中断处理程序、关中断、恢复现场。
5. 中断返回