

现代操作系统：

由一个或多个处理器，一些主存储器，磁盘，打印机，键盘，鼠标，网络接口和其他输入输出设备组成。

内存：

处理缓存时的问题：

何时把一个新项目放入缓存

把新项目放入哪个缓存

当需要插槽时，从缓存中删除哪一个

在更大的内存中放入一个新项目

进行多线程全局变量问题

多线程就是在同个进程中运行的。因此在进程中的全局变量所有线程都可共享。这就造成了一个问题，因为线程执行过程的顺序是无序的，导致有可能造成数据错误

cpu缓存与系统缓存

CPU缓存（Cache Memory）是位于CPU与内存之间的临时存储器，它的容量比内存小的多但是交换速度却比内存要快得多。缓存的出现主要是为了解决CPU运算速度与内存读写速度不匹配的矛盾，因为CPU运算速度要比内存读写速度快很多，这样会使CPU花费很长时间等待数据到来或把数据写入内存。在缓存中的数据是内存中的一小部分，但这一小部分是短时间内CPU即将访问的，当CPU调用大量数据时，就可避开内存直接从缓存中调用，从而加快读取速度。

操作系统缓存：为提高系统的存取速度，在地址映射机制中增加一个小容量的寄存器，即快表，用来存放当前访问最频繁的少数活动页面的页号，当用户需要时可以通过快表查询，大大提高了查询速度

提高CPU利用率

离散化的分配方式

由覆盖到对换的虚拟存储

操作系统基本服务

项目创建，程序执行，io设备接入，文件访问控制，系统接入，错误检测与响应

什么是操作系统

管理计算机硬件和软件资源并为计算机程序提供公共服务的软件，是计算机系统中系统软件重要组成部分。

应用程序通常需要操作系统才能够运行。

扩展机器
资源管理器
进程管理器
一个可扩展的服务机器

操作系统类型：

批处理操作系统、分时操作系统和实时操作系统。

多处理器操作系统 个人电脑操作系统 手持机操作系统 嵌入式操作系统
传感器节点操作系统 实时操作系统

操作系统历史

手工操作
批处理系统：联机批处理系统 脱机批处理系统
多道程序系统：单处理机系统中多道程序 多道批处理系统
分时系统
实时系统

操作系统的三种基本类型：多道批处理系统、分时系统、实时系统。

演变

资源利用最大化
硬件升级和新型硬件
新服务
修复
用户体验

第一代 真空管 插板（机器语言）
第二代 晶体管 批处理系统（汇编语言）
第三代 集成电路和多道程序设计（分时系统）
第四代 个人电脑
第五代 移动电脑

系统调用：

用户程序和操作系统之间的接口，内核模式执行

陷阱指令：用户模式到内核模式

库程序：封装陷阱指令，在用户模式下执行

系统调用类型：

进程控制：创建进程，终止进程，获取/设置进程属性

文件操作：创建，删除，读，写，设置/获取文件属性

设备管理：请求设备，释放设备，读，写

socket（套接字）：打开连接，关闭连接，读写消息
信息维护

os功能

处理器管理、存储器管理、设备管理、文件管理、作业管理。

os特征：

并发性，虚拟性，异步性，共享性

os结构

单体系统

层次式结构

微内核

c/s模型

虚拟机

外核

假脱机技术：

假脱机技术”，又称“SPOOLing 技术”，用软件的方式模拟脱机技术。SPOOLing系统的组成如下：

“输入井”模拟脱机输入时的磁带，用于收容I/O设备输入的数据

“输出井”模拟脱机输出时的磁带，用于收容用户进程输出的数据

“输入进程”模拟脱机输入时的外围控制机

“输出进程”模拟脱机输出时的外围控制机

用户态和核心态

用户态和内核态

内核态：cpu可以访问内存的所有数据，包括外围设备，例如硬盘，网卡，cpu也可以将自己从一个程序切换到另一个程序。

用户态：只能受限的访问内存，且不允许访问外围设备，占用cpu的能力被剥夺，cpu资源可以被其他程序获取。

为什么要有用户态和内核态？

由于需要限制不同的程序之间的访问能力，防止他们获取别的程序的内存数据，或者获取外围设备的数据，并发送到网络，CPU划分出两个权限等级 -- 用户态和内核态。

用户态与内核态的切换

所有用户程序都是运行在用户态的，但是有时候程序确实需要做一些内核态的事情，例如从硬盘读取数据，或者从键盘获取输入等，而唯一可以做这些事情的就是操作系统，所以此时程序就需要先操作系统请求以程序的名义来执行这些操作。

这时需要一个这样的机制：用户态程序切换到内核态，但是不能控制在内核态中执行的指令

这种机制叫系统调用，在CPU中的实现称之为陷阱指令(Trap Instruction)

他们的工作流程如下：

1. 用户态程序将一些数据值放在寄存器中，或者使用参数创建一个堆栈(stack frame)，以此表明需要操作系统提供的服务。
2. 用户态程序执行陷阱指令
3. CPU切换到内核态，并跳到位于内存指定位置的指令，这些指令是操作系统的一部分，他们具有内存保护，不可被用户态程序访问
4. 这些指令称之为陷阱(trap)或者系统调用处理器(system call handler)，他们会读取程序放入内存的数据参数，并执行程序请求的服务
5. 系统调用完成后，操作系统会重置CPU为用户态并返回系统调用的结果

2. 用户态和内核态的转换

1) 用户态切换到内核态的3种方式

a. 系统调用

这是用户态进程主动要求切换到内核态的一种方式，用户态进程通过系统调用申请使用操作系统提供的服务程序完成工作，比如前例中fork()实际上就是执行了一个创建新进程的系统调用。而系统调用的机制其核心还是使用了操作系统为用户特别开放的一个中断来实现，例如Linux的int 80h中断。

b. 异常

当CPU在执行运行在用户态下的程序时，发生了某些事先不可知的异常，这时会触发由当前运行进程切换到处理此异常的内核相关程序中，也就转到了内核态，比如缺页异常。

c. 外围设备的中断

当外围设备完成用户请求的操作后，会向CPU发出相应的中断信号，这时CPU会暂停执行下一条即将要执行的指令转而去执行与中断信号对应的处理程序，如果先前执行的指令是用户态下的程序，那么这个转换的过程自然也就发生了由用户态到内核态的切换。比如硬盘读写操作完成，系统会切换到硬盘读写的中断处理程序中执行后续操作等。

这3种方式是系统在运行时由用户态转到内核态的最主要方式，其中系统调用可以认为是用户进程主动发起的，异常和外围设备中断则是被动的。

从不同角度看操作系统

资源管理角度

程序控制角度

操作系统控制计算机的角度

人机交互的角度

程序接口的角度

程序接口的角度

进程的创建

系统初始化

由一个正在运行的进程执行创建进程的系统调用

用户创建一个新进程的请求

发起一个批处理作业

进程终止：

正常退出

错误退出

致命错误

被他人杀死

临界区互斥访问条件

空闲让进

忙则等待

有限等待

让权等待（不能进入临界区时，应立刻释放处理机，防止忙等待）

设计os目标：

定义抽象

提供基本操作

确保隔离

管理硬件

类Unix系统

```
debian
Gentoo
Ubuntu
centos
fedora
openSUSE
Redhat
```

类windows

```
winxp
win7
win8
win8.1
win10
```

历史上电脑

1950年

EDVAC

第一台并行计算机，实现了计算机之父“冯·诺伊曼”的两个设想：采用二进制和存储程序。

1954年

TRADIC

IBM公司制造的第一台使用晶体管的计算机，增加了浮点运算，使计算能力有了很大提高

IBM 1401

这是第二代计算机中的代表，用户当时可以租用。

DEC 公司推出了 PDP8 型计算机，标志着小型机时代的到来，接着而来的是一系列的PDP和VAX小型机

IBM S/370

这是IBM的更新换代的重要产品，采用了大规模集成电路代替磁芯存储，小规模集成电路作为逻辑元件，并使用虚拟存储器技术，将硬件和软件分离开来，从而明确了软件的价值。

Altair 8800

MITs制造的，带有1KB存储器。这是世界上第一台微型计算机。

地址空间是一个进程可用于寻址内存的一套 地址集合

中断与陷入的主要区别陷入通常由**处理机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。

操作系统设计

- 实施
 - 机制与政策
 - 静态与动态结构
- 业绩
 - 空间-时间的权衡
 - 缓存
- 对操作系统的评价
 - 业绩
 - MTBF 平均故障时间
 - MTTR 平均故障修复时间
 - 可靠性、可用性、可维护性
 - 方便性
 - 便携性

- 操作系统设计的趋势
 - 多核心
 - 虚拟化
 - 大地址空间，网络
 - 并行和分布式系统
 - 多媒体
 - 电池供电的计算机
 - 嵌入式系统
 - 传感器节点

操作系统期末知识点整理

\1. 操作系统的目标：方便、有效、可扩充、开放

\2. 操作系统的作用：

- (1) OS作为用户与计算机硬件系统之间的接口
- (2) OS作为计算机资源系统的管理者
- (3) OS实现了对计算机资源的抽象

\3. 推动操作系统发展的主要动力

- (1) 不断提高计算机资源的利用率
- (2) 方便用户
- (3) 器件的不断更新换代
- (4) 计算机体系结构的不断发展
- (5) 不断提出新的应用需求

\4. 实时系统与分时系统异同

- (1) 多路性，都表现为系统按分时原则为多个终端用户服务
- (2) 独立性，都表现为每个终端用户与系统交互时，彼此相互独立互不干扰
- (3) 及时性，都表现为对实时性的要求依据人所能接受的等待时间确定
- (4) 交互性，信息查询系统中仅限于访问系统中某些特定的专用服务，分时系统能向终端用户提供数据处理、资源共享服务
- (5) 可靠性，分时系统要求系统可靠，实时系统要求系统高可靠度

\5. 操作系统的主要功能：

- (1) 处理机管理：进程控制、同步、通信、调度
- (2) 存储器管理：内存分配、保护、地址映射、扩充
- (3) 设备管理：缓冲管理、设备分配、处理

(4) 文件管理：文件存储空间管理、目录管理、文件读写管理和保护

\6. P33前驱图，P37进程状态图

\7. 引起进程创建的事件：用户登录、作业调度、提供服务、应用请求

\8. 进程的创建

(1) 申请空白PCB

(2) 为新进程分配其运行所需的资源

(3) 初始化进程控制块PCB

(4) 如果进程就绪队列能够接纳新进程，便将进程插入就绪队列

\9. 引起进程中止的事件：

(1) 正常结束，进程任务完成，准备退出运行

(2) 异常结束，进程运行时发生某种错误使程序无法继续运行，如越界错，保护错，非法指令，运行超时，等待超时，算术运算错，I/O故障

(3) 外界干预，进程应外界请求而终止运行，如操作员或操作系统干预、父进程请求、因父进程终止

\10. 引起进程挂起的原因：

(1) 终端用户的需要

(2) 父进程的请求

(3) 负荷调节的需要

(4) 操作系统的需要

\11. 进程同步机制遵循：空闲让进，忙则等待，有限等待，让权等待

\12. 进程与管程的区别

(1) 都定义了数据结构，进程定义私有数据结构PCB，管程定义公共数据结构（如消息队列）

(2) 都存在对各自数据结构上的操作，进程顺序程序执行相关操作，管程进行同步操作和初始化操作

(3) 设置进程的目的是实现系统并发性，设置管程的目的是解决共享资源和互斥问题

(4) 进程主动工作方式，管程被动工作方式

(5) 进程之间能并发执行，管程不能与其调用者并发

(6) 进程具有动态性，管程是操作系统的一个资源管理模块，供进程调用

进程间互斥

事件计数器 条件变量 管程 消息通信 管道 共享内存

\13. P60进程同步代码

\14. 进程与线程的区别

(1) 调度的基本单位，在传统OS和引入线程的OS中，进程和线程都是能独立调度和分派的基本单位

(2) 并发性，在引入线程的OS中，进程之间可以并发执行，一个进程中的多个线程可以并发执行

(3) 拥有资源，进程可以用有资源，并作为系统中拥有资源的一个基本单位；线程本身并不用于资源，而是仅有一点必不可少的能保证独立运行的资源

(4) 独立性，同一个进程中的不同线程之间的独立性要比不同进程之间的独立性低得多

(5) 系统开销，在创建或撤销进程时OS付出的开销，明显大于线程的创建和撤销

(6) 支持多核处理机，单线程进程只能运行在一个处理机上，多线程进程能让线程并行执行

\15. 三级处理及调度层次的基本功能

(1) 高级调度，根据某种算法，决定将外存上处于后备队列的那几个作业调入内存，为他们创建进程、分配必要的资源，并将它们放入就绪队列

(2) 低级调度，根据某种算法，决定就绪队列中的那几个进程应获得处理机，并由分派程序将处理机分配给被选中的进程

(3) 中及调度，主要功能是提高内存利用率和吞吐量，将暂时不能运行的进程调至外存等待，把外存上已具备运行条件的进程再重新调入内存

\16. 处理机调度算法的目标

(1) 处理机调度算法的共同目标：资源利用率、公平性、平衡性、策略强制执行

(2) 批处理系统的目标：平均周转时间段、系统吞吐量高、处理及利用率高

(3) 分时系统的目标：响应时间快、均衡性

(4) 实时系统的目标：截止时间的保证、可预测性

\17. P89FCFS先来先服务SJF短作业优先调度算法

\18. 短作业优先SJF缺点

(1) 必须预知作业的运行时间

(2) 对长作业不利，长作业的周转时间会明显增长

(3) 人机无法实现交互

(4) 完全未考虑作业的紧迫度，故不能保证紧迫性作业能及时得到处理

\19. P93轮转调度算法

\20. 实现实时调度的基本条件

(1) 提供必要的信息，就绪时间、开始截止时间和完成截止时间、处理时间、资源要求、优先级

(2) 系统处理能力强，若处理能力不够强可能会导致实时任务不能及时处理

(3) 采用抢占式调度机制

(4) 具有快速切换机制，具有对中断的快速响应能力和快速的任务分派能力

\21. 产生死锁的必要条件

(1) 互斥条件，进程对所分配到的资源进行排他性使用

(2) 请求和保持条件，进程已经保持了至少一个资源，但又提出了新的资源请求，该资源已被其他进程抢占

(3) 不可抢占条件，进程已获得的资源在未使用完之前不能被抢占，只能在进程使用完时释放

(4) 循环等待条件，再发生死锁时，必然存在一个进程—资源的循环链

\22. 处理死锁的方法

- (1) 预防死锁，通过设置某些限制条件去破坏产生死锁四个必要条件中的一个或几个来预防死锁
- (2) 避免死锁，在资源的动态分配过程中，用某种方法防止系统进入不安全状态而产生死锁
- (3) 检测死锁，通过检测机构及时的检测死锁的发生，然后采取适当措施把进程从死锁状态解救出来
- (4) 解除死锁，当检测到系统中已发生死锁时，然后采取适当措施把进程从死锁状态解救出来

\23. 预防死锁

(1) 破坏请求和保持协议

第一种协议，所有进程在开始运行之前，必须一次性申请其在整个运行过程中所需要的全部资源

第二种协议，允许一个进程只获得运行初期的所需资源后，便开始运行，进程运行过程中再逐步释放已分配给自己的、且已用完的全部资源，然后再请求新的所需资源

(2) 破坏不可抢占条件，当一个已保持某些不可抢占资源的进程，提出新的自愿请求而得不到满足时，它必须释放已保持的所有资源

(3) 破坏循环等待条件，对系统所有资源类型进行线性排序并赋予不同编号，每个进程必须按序号递增的顺序请求资源，若一个已经请求道一些高序号资源的进程又想请求低序号的资源，必须先释放所具有相同和更高序号的资源后才能申请

\24. P113、1**19银行家算法**

\25. P131四种动态分区算法

\26. 四种分区分配算法的基本原理

- (1) 首次适应算法FF，每次从链首开始顺序查找，知道找到一个大小能满足要求的空闲分区位置
- (2) 循环首次适应算法NF，每次从上次找到的空闲分区的下一个空闲分区开始查找，知道找到一个大小能满足要求的空闲分区
- (3) 最佳适应算法BF，每次查找时，总是把能满足要求又是最小的空闲分区分配给作业
- (4) 最坏适应算法WF，每次查找时，总是挑一个最大的空闲区

\27. P140、141、148地址变换机构（分页、分段）

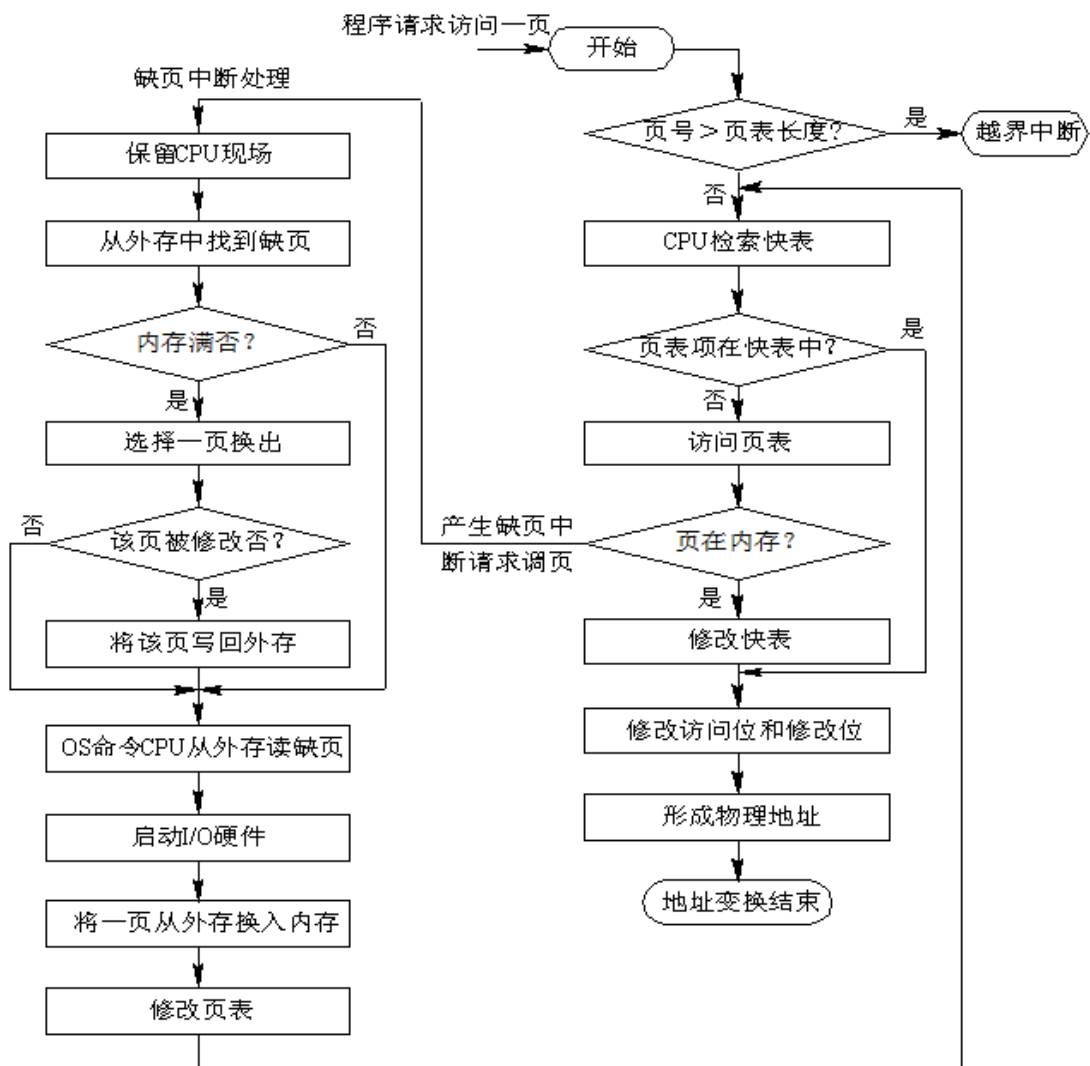
\28. 为什么要引入分段存储管理方式

- (1) 方便变成，
- (2) 信息保护，
- (3) 信息共享，
- (4) 动态增长，
- (5) 动态链接，

\29. 分页与分段的区别

- (1) 页是信息的物理单位，分页仅仅是系统管理上的需要，对用户是不可见的；段是信息的逻辑单位，分段的目的在于能更好的满足用户的需要
- (2) 页的大小是固定且有系统决定的，分页系统中直接由硬件决定；段的长度决定于用户编写的程序
- (3) 分页系统中用户程序地址空间是一维单一线性地址空间；分段系统中用户程序的地址空间是二维的

\30. P158请求分页的地址变换流程图



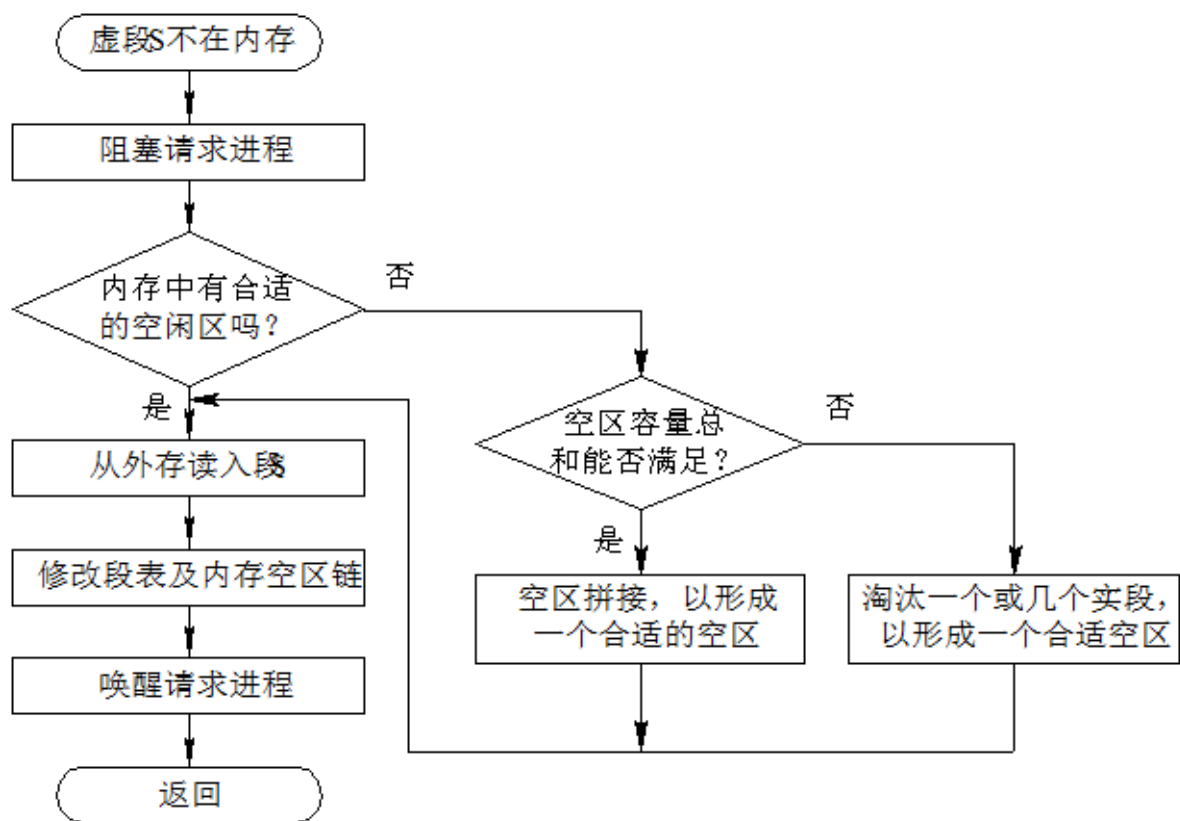
\31. P163页面置换算法

\32. P180 I/O系统层次结构

I/O应答

用户层程序	产生I/O请求、格式化I/O、Spolling
设备独立性软件	映射、保护、分块、缓冲、分配
设备驱动程序	设置设备寄存器，检查状态
中断处理程序	
硬件	执行I/O操作

\33. P174请求分段系统中的中断处理过程



\34. I/O软件的层次结构

- (1) 用户层I/O软件
- (2) 设备独立性软件
- (3) 设备驱动程序
- (4) 中断处理程序

\35. 设备控制器的基本功能

- (1) 接受和识别命令
- (2) 数据交换
- (3) 标识和报告设备状态
- (4) 地址识别
- (5) 数据缓冲区
- (6) 差错控制

\36. 中断处理过程

- (1) 测定是否有未响应的中断信号，程序每执行完当前指令后，处理机都要测试是否有未响应的中断信号，若没有则执行下一条指令，若有则准备转去执行中断处理程序
- (2) 保护被中断进程的CPU环境，把控制权交给中断处理程序之前，需要先保护被中断进程的CPU环境，以便以后能恢复运行
- (3) 转入相应的设备处理程序，由处理机确定引起本次中断的I/O设备后并进行地址装入操作
- (4) 中断处理，对不同的设备，有不同的中断处理程序
- (5) 恢复CPU的现场并推出中断

\37. 为什么要引入缓冲区

- (1) 缓和CPU与I/O设备间速度不匹配的矛盾
- (2) 减少CPU的中断频率，放宽对CPU中断响应时间的限制
- (3) 解决数据粒度不匹配的问题
- (4) 提高CPU和I/O设备之间的并行性

\38. P217磁盘调度算法

\39. 对文件目录的管理要求

- (1) 实现“按名存取”，用户只需要向系统提供所需访问文件的名字，便能快速准确的找到指定文件在外存上的存储位置
- (2) 提高对目录的检索速度，通过合理地组织目录结构加快对目录的检索速度，从而提高对文件的存取速度
- (3) 文件共享，在多用户系统中，应允许多个用户共享一个文件
- (4) 允许文件重名，系统应允许不同用户对不同文件采用相同的名字，以便于用户按照自己的习惯给文件命名和使用文件

\1. 分时系统：在一台主机上连接多个配有显示器和键盘的终端并由此组成的系统，该系统允许多个用户同时通过自己的终端，以交互方式使用计算机，共享主机资源

\2. 进程：进程是进程实体的运行过程，是系统进行资源分配和调度的一个独立单位

\3. 死锁：如果一组进程中的每一个进程都在等待仅由该组进程中的其他进程才能引发的时间，那么该组进程是死锁的

\4. 对换：把内存中暂时不能运行进程或者暂时不用的程序和数据换出到外存上，以便腾出足够的内存空间，再把已具备运行条件的进程或进程所需要的程序和数据换入内存

\5. 虚拟存储器：具有请求调入功能和置换功能，能从逻辑上对内存容量加以扩充的一种存储器系统

\6. 中断：CPU对I/O设备发来的中断信号的一种响应。CPU暂停正在执行的程序，保留CPU环境，自动转去执行该I/O设备的中断处理程序，执行完后，再回到断点，继续执行原来的程序

\7. 中断向量表：

\8. 假脱机：将一台物理I/O设备虚拟为多台逻辑I/O设备，允许多个用户共享一台物理I/O设备

\9. 设备驱动程序：与硬件直接相关，用于具体实现系统对设备发出的操作指令，驱动I/O设备工作的驱动程序

\10. 阻塞状态：正在执行的进程由于突发事件（如I/O请求、申请缓冲区失败等）暂时无法继续执行时的状态，就是进程执行受到阻塞

\11. 快表：具有并行查询能力的特殊告诉缓冲寄存器，又称为“联想寄存器”

\12. 文件控制块：为了能对一个文件进行正确的存取，必须为文件设置用于描述和控制文件的数据结构，称为“文件控制块”

\13. 进程控制块：作为进程实体的一部分，记录了操作系统所需的，用于描述进程的当前情况及管理进程运行的全部信息，是操作系统中最重要的记录性数据结构

\14. 线程控制块：记录所有用于控制和管理线程的信息

\15. 作业控制块：保存了系统对作业管理和调度所需的全部信息

\16. 作业：不仅包含通常的程序和数据，还配有一份作业说明书，系统根据该书名数来对程序的运行进行控制

\17. 抢占式调度：调度程序根据某种原则，去暂停某个正在执行的程序，将已分配给该程序的处理机重新分配给另一进程

\18. 非抢占式调度：一旦处理机分配给进程后，就一直让他运行下去，直到该进程完成

\19. 文件目录：为向用户提供文件的存取控制及保护功能，而按一定规则对系统中的文件名进行组织所形成的表

抖动的原因

产生抖动的主要原因是进程频繁访问的页面数目高于可用的物理块数（分配给进程的物理块不够）。

为进程分配的物理块太少，会使进程发生抖动现象。为进程分配的物理块太多，又会降低系统整体的并发度，降低某些资源的利用率

-内存颠簸的解决策略是：

- 1-如果是因为页面替换策略失误，可以修改替换算法来解决这个问题；
- 2-如果是因为运行的程序太多，造成程序无法同时将所有频繁访问的页面调入内存，则要降低多道程序的数量。
- 3-否则，还剩下两个办法：1终止该进程；2增加物理内存容量

io控制四种方法

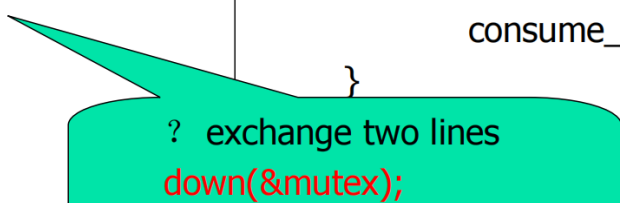
	完成一次读/写的过程	CPU干预频率	每次I/O的数据传输单位	数据流向	优缺点
程序直接控制方式	CPU发出I/O命令后需要不断轮询	极高	字	设备→CPU→内存 内存→CPU→设备	每一个阶段的优点都是解决了上一阶段的最大缺点。总体来说，整个发展过程就是要尽量减少CPU对I/O过程的干预，把CPU从繁杂的I/O控制事务中解脱出来，以便更多地去完成数据处理任务。
中断驱动方式	CPU发出I/O命令后可以去做其他事，本次I/O完成后设备控制器发出中断信号	高	字	设备→CPU→内存 内存→CPU→设备	
DMA方式	CPU发出I/O命令后可以去做其他事，本次I/O完成后DMA控制器发出中断信号	中	块	设备→内存 内存→设备	
通道控制方式	CPU发出I/O命令后可以去做其他事。通道会执行程序以完成I/O，完成后通道向CPU发出中断信号	低	一组块	设备→内存 内存→设备	

###

信号量

```
#define N 100 /*number of slots in the buffer*/
typedef int semaphore;
semaphore mutex=1;
semaphore empty=N;
semaphore full=0;
void producer(void){
    int item;
    while(TRUE){
        produce_item(&item);
        down(&empty);
        down(&mutex);
        enter_item(item);
        up(&mutex);
        up(&full);
    }
}

void consumer(void){
    int item;
    while(TRUE){
        down(&full);
        down(&mutex);
        remove_item(&item);
        up(&mutex);
        up(&empty);
        consume_item(item);
    }
}
```



? exchange two lines
down(&mutex);