第一章 绪论

1.计算机系统四个组成部分

计算机硬件、操作系统、系统程序、用户。

2.操作系统概念

操作系统是一组控制和管理计算机硬件和软件资源、合理地对各类作业进行调度，以及方便用户的程序集合。

3. 操作系统的目标

执行用户程序，更容易地解决用户问题。

使计算机系统使用方便。

有效地使用计算机硬件。

4. 两个视角

用户视角、系统视角

5.硬中断（interrupt）

interrupt 是一种在系统内硬件设备产生，又称外部中断（相对于 CPU)。

6.软中断（trap）

Trap 是指令产生的中断，又称内部中断（相对于 CPU)。

7.存储结构

主存：CPU 可以直接访问的大型存储介质。

辅存：非易失性大存储容量的主存储器的扩展。(又称二级存储)

磁盘：最常用的辅存。磁盘表面逻辑上划分为磁道，再细分为扇区。磁盘控制器决定设备和计算机之间的逻辑交互。

8.DMA（直接内存访问）

用于高速 I/O 设备，能够以接近内存速度传送信息。

设备控制器无需CPU干预直接将数据块从缓冲存储器(设备控制器中的存储)直接传输到主存,每个块只产生一个中断,而不是每个字节的一个中断(块设备)。

9.计算机系统

单处理器系统、多处理器系统、集群系统

10.多处理器系统（并行系统、紧耦合系统）

定义：多处理器系统有多个紧密通信的CPU，它们共享计算机总线，有时还有时钟、内存和外设等。

优点：增加吞吐量、规模经济、增加可靠性。

分类：

非对称多处理：每个处理器都有各自特定的任务。一个主处理器控制系统，其他处理器或者向主处理器要任务或做预先定义的任务。

对称多处理:每个处理器都要完成操作系统中的所有任务。所有处理器对等，处理器之间没有主-从关系。

11.集群系统

定义：与多处理器系统一样，集群系统将多个CPU集中起来完成计算任务。然而,集群系统与多处理器系统不同,它是由两个或多个独立的系统耦合起来的。集群计算机共享存储并通过局域网络连接或更快的内部连接。

12.操作系统结构

操作系统最重要的一点是要有多道程序处理能力。多道程序设计通过组织作业（编码或数据）使CPU总有一个作业在执行，从而提高了CPU的利用率。

13.操作系统的三种基本类型

Batch systems（批处理系统）

Time-sharingsystems（分时系统）

Real timesystems（实时系统）

14.批处理系统

工作方式：

用户将作业交给系统操作员，系统操作员将许多用户的作业组成一批作业(jobs)之后输入到计算机中，在系统中形成一个自动转接的连续的作业流，系统自动、依次执行每个作业。最后由操作员将作业结果交给用户。操作系统：自动将控制从一个任务转到下一个任务。

分类：单道批处理系统、多道批处理系统

批处理操作系统优点：作业流程自动化、效率高、吞吐量高。

批处理操纵系统缺点：无交互手段、调试程序困难。

15.分时系统- 交互式计算

分时系统（或多任务）是多道程序设计的延伸。

共享需要一种交互计算机系统，它能提供用户与系统之间的直接通信。响应时间短（通常小于一秒钟）。

由于每个动作或命令都较短，每个用户只需少量CPU时间，用户之间切换时间短，所以用户会感觉整个系统为自己所用。

16.实时系统

定义：实时操作系统是保证在一定时间限制内完成特定功能的操作系统。

分类：

硬实时系统：硬实时要求在规定的时间内必须完成操作，这是在操作系统设计时保证的（不按时间约束完成意味着失败）。（又称强实时系统）

软实时系统：软实时则只要按照任务的优先级，尽可能快地完成操作即可，（不按时完成意味着“不好”）。（又称准实时系统、若实时系统）。

17.操作系统的双重模式操作

为了区分操作系统代码和用户定义代码的执行，至少需要两种独立的操作模式：用户模式（用户态）、监督程序模式（管理模式、系统模式、特权模式、核心态）。操作系统保护能力实现的必要条件。

将能引起损害的机器指令作为特权指令。用户模式下想要执行特权指令，硬件不会执行，会认为是非法指令，并以陷阱的形式通知操作系统。

本章应掌握主要内容：

1. 计算机系统硬件结构

2. 操作系统定义

3. 应用、操作系统、硬件的层次关系

4. 多道批处理、分时系统、实时系统的定义及意义。

5. 用户态与核心态（管态、监督模式、系统态）的理解。

第二章 操作系统结构

1. 操作系统服务

用户界面(一种是命令行界面;另一种是批界面,最为常用的是图形用户面)、程序执行、I/O 操作、文件系统操作、通信、错误检测、资源分配、统计、保护和安全。

2. 操作系统的用户界面：

命令解释程序（CLI)、图形用户界面（GUI）

命令解释程序主要作用：获取并执行用户指定的下一条命令。

系统调用（System Call）

操作系统内核提供一系列预定功能，通过一组称为系统调用的接口呈现给编程人员，系统调用把应用程序的请求传给内核，系统调用相应的内核函数完成所需的处理，将处理结果返回给应用程序。

3. 向操作系统传递参数的三种方法：

通过寄存器来传递参数。

若参数数量比寄存器多，参数通常存在内存的块和表中，并将块的地址通过寄存器来传递。

参数也可以通过程序放在或压入堆栈中，并通过操作系统弹出。

4. 系统程序分类：

文件管理、状态信息、文件修改、程序语言支持、程序装入和执行、通信。

5. 操作系统设计和实现：

设计目标需求：用户目标和系统目标

机制和策略：机制决定如何做，策略决定做什么

6. 操作系统结构

简单结构、分层方法、微内核、模块、虚拟机

1. 简单结构

MS-DOS、原始的UNIX操作系统

2. 分层方法

定义：操作系统分成若干层（级）。最底层（层0）为硬件，最高层（层N）为用户接口。(理想方法，困难在于如何划分层)

3. 微内核

微内核方法将所有非基本部分从内核中移走，并将它们实现为系统或用户程序，这样得到了更小的内核。

微内核的主要功能是使客户程序和运行在用户空间的各种服务之间进行通信。

4. 模块：

大多数现代操作系统按模块方式实现内核

采用面向对象的方法

每个核心组件是分开的

每部分与已知接口的其他部分通信

可以是动态加载方式，每部分根据需要加载到内核

总之，类似于层，但更灵活。

5. 虚拟机

虚拟机（VirtualMachine）指通过软件模拟的具有完整硬件系统功能的、运行在一个完全隔离环境中的“完整”计算机系统。

本章应掌握主要内容：

1. 操作系统组成部分

2. 系统调用、系统调用参数传递方式、系统调用分类

3. 操作系统内核结构（单核+模块、分层结构、微内核、虚拟机、模块化结构、混合结构）不同结构的基本思想和优缺点。

4. 本章概念总结

第三章 进程

1. 进程概念

进程是执行中的程序。进程还包括当前活动信息：通过程序计数器的值和处理器寄存器的内容等来表示。另外，进程内存影像包括进程堆栈段（包括临时数据，如函数参数、返回地址和局部变量）和数据段（包括全局变量），包括堆，是在进程运行期间动态分配的内存。

进程与程序是截然不同的两个概念

一个程序可以对应多个进程(Oneprogram can be several processes)

一个进程也可以由多个程序段共同完成一项任务（接力）

进程五个特征（而程序不具备）

动态性：是进程的最基本的特征，表现在进程由创建而产生，由调度而执行，因得不到资源而暂停执行，由撤销而消亡。

并发性：多个进程实体同存于内存中，能在一段时间内同时执行。

独立性：进程实体是一个能独立运行的基本单位，同时也是系统中独立获得资源和独立调度的基本单位。

异步性：指进程按各自独立的、不可预知的速度向前推进；或者说，进程按异步方式运行。

结构特征：从结构上看，进程实体由程序段、数据段以及进程控制块组成。三部分也称为进程影像。

进程状态

新建（new）：进程正在被创建。

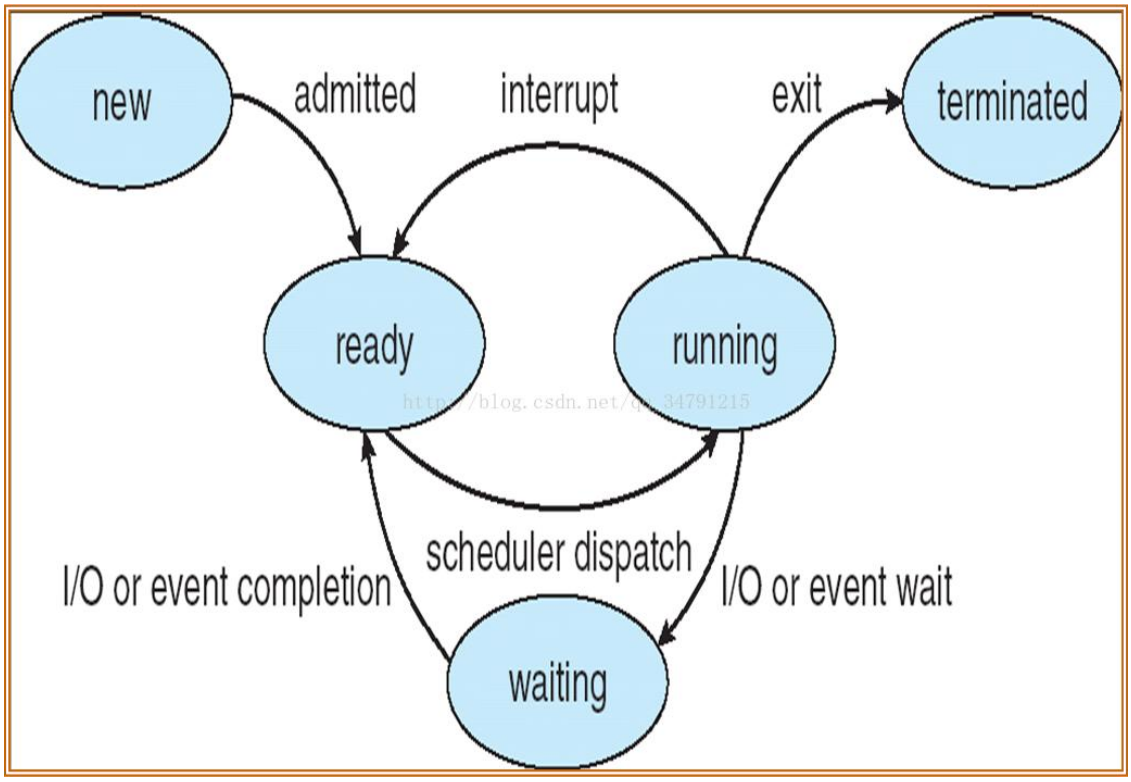
运行（running）：指令正在被执行。

等待（waiting）：进程等待某个时间的发生（如 I/O 完成或收到信号）。

就绪（ready）：进程等待分配处理器。

终止（terminated）：进程完成执行。

进程状态图“”



进程控制块

每个进程在操作系统内对应一进程控制块（PCB，process control block）。

概念：

系统为了管理进程设置的一个专门的数据结构，用它来记录进程的外部特征，描述进程的运动变化过程。

要点：

系统利用 PCB 来控制和管理进程，所以 PCB 是系统感知进程存在的唯一标志

进程与 PCB 是一一对应的

通常进程队列是进程所对的 PCB 队列

操作系统通过 PCB 来感知进程的存在

PCB 表：

系统把所有 PCB 组织在一起，并把它们放在内存的固定区域，就构成了 PCB 表。PCB 表的大小决定了系统中最多可同时存在的进程个数，称为系统的并发度。

结构：

链接结构：同一状态进程的 PCB 组成一个链表，不同状态的进程对应多个不同的链表。（就绪链表、阻塞链表……）

索引结构：对具有相同状态的进程，分别设置各自的 PCB 索引表，表明 PCB 在 PCB 表中的地址。

...

2. 进程调度

多道程序设计的目的是无论何时都有进程在运行，从而使 CPU 利用率达到最大化。

分时系统的目的是在进程之间快速切换 CPU 以便用户在程序运行时能与其进行交互。

为达到此目的，进程调度选择一个可用的进程到 CPU 上执行。单处理器系统从不会有超过一个进程在运行。如果有多个进程，那么余下的则需要等待 CPU 空闲并重新调度。

3. 调度程序

长期调度程序（或作业调度程序）

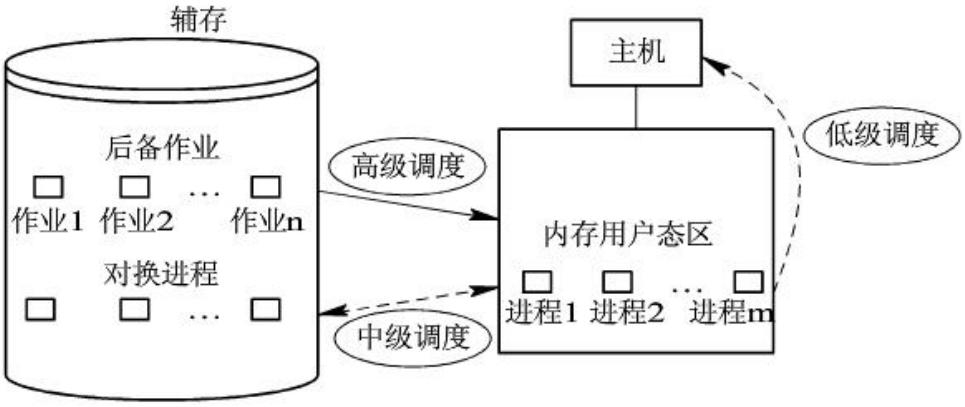
短期调度层序（或 CPU 调度程序）

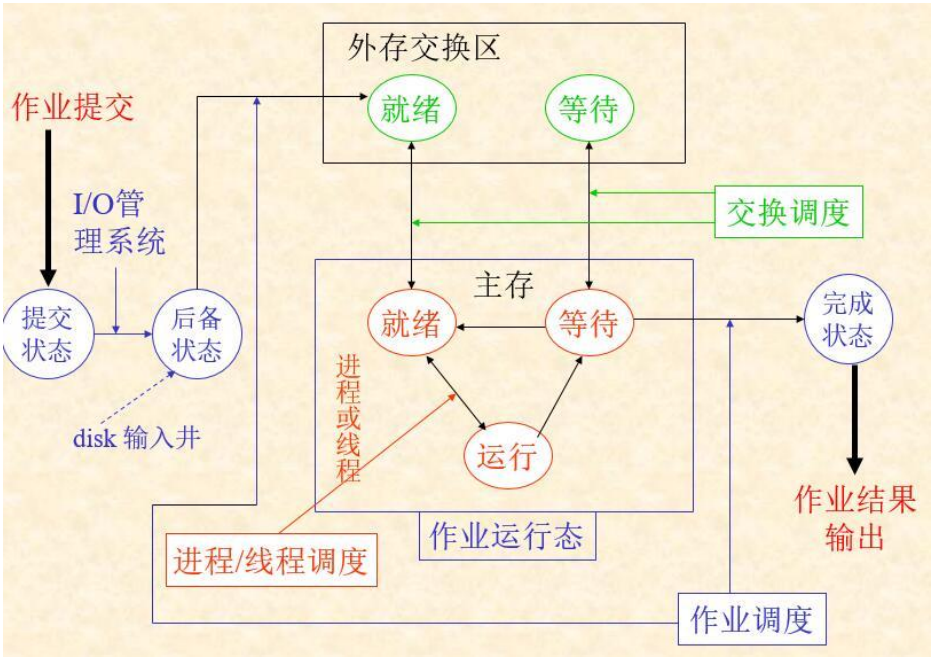
这两个调度程序的主要差别是它们执行的频率。

短期调度程序必须频繁为 CPU 选择新进程。

长期调度程序控制多道程序设计的程度（内存中的进程数量）。

中期调度程序（又名交换）核心思想：调度程序根据一定策略临时把内存中的某个进程换出到外存或者把外存中的进程换入内存的过程。（一定程度上降低长期调度的设计或策略难度）。





短期调度执行的频率最高。短期调度在内存作业中选择就绪执行的作业，并为他们分配 CPU。中期调度作为一种中等程度的调度程序，尤其被用于分时系统，一个交换方案的实施，将部分运行程序移出内存，之后，从中断处继续执行，从而改善进程组合或者因内存要求的改变引起了可用内存的过度使用而需要释放内存。长期调度确定哪些作业调入内存以执行. 它们主要的不同之处是它们的执行的频率。短期调度必须经常调用一个新进程，由于在系统中，长期调度处理移动的作业时，并不频繁被调用，可能在进程离开系统时才被唤起。

4. 上下文切换

将 CPU 切换到另一个进程需要保存当前进程的状态并恢复另一个进程的状态，这一任务称为上下文切换。

5. 进程操作

进程标识符（pid）

通过 fork()系统调用，可创建新进程。

新进程通过复制原来的地址空间形成。

子进程返回 0，父进程返回大于 0，错误返回-1.

6. Fork()的功能

内核为子进程做一个父进程的上下文的拷贝（复制父进程的 PCB 作为子进程的 PCB），子进程与父进程共享子进程创建之前父进程所有的资源，父进程和子进程在不同的地址空间上运行。

7. Fork()的几个要点

父子进程具有独立的内存空间。

父子进程资源的共享与分离：父进程中在 fork 之前创建的变量——先继承，后分离，子进程继承了父进程的私有变量，作为自己的私有变量；Fork 之后各自创建的变量——完全分离。

子进程继承了父进程的所有资源，其中包括父进程的这些私有变量，但继承以后互相不能访问。

进程间通信有两种基本模式

共享内存：允许以最快的速度进行方便的通信，在计算机中可以达到内存的速度，等等。

消息传递：对于交换较少数量的数据很有用，因为不需要避免冲突，对于计算机间的通信，消息传递也比共享内存更易于实现。

第四章 线程

1. 引入线程的原因：进程时空开销大、通信代价大、不能很好的利用多处理器

系统、不适合并行计算和分布计算的要求。

2. 线程的定义：

进程中的一个实体、CPU 调度和分派的基本单位、与同进程内的其它线程共享进程所拥有的资源：线程必须在某个进程内执行，它所需的其它资源，如代码段、数据段、打开的文件和信号等，都由它所属进程拥有。

只单独拥有运行所必须的资源：如 PC、寄存器、栈

3. 线程的优点：并发程度高、响应度高；易于调度，开销小；资源共享；多处理器体系结构的利用。

4. 进程和线程的区别

一个进程可以有多个线程，但至少有一个线程；而一个线程只能在一个进程的地址空间内活动。

每当创建一个进程时，至少要同时为该进程创建一个线程，否则该进程无法被调度执行。

地址空间和其他资源（如打开文件）：进程间相互独立，同一进程的各线程间共享——某进程内的线程在其他进程不可见。

通信：进程间通信采用 IPC，线程间可以直接读写进程数据段（如全局变量）来进行通信；所有线程可共享进程的主存，不需要特殊的通信机制。

调度：线程上下文切换比进程上下文切换要快得多。

OS 中引入进程目的：使多个程序并发执行，以便改善资源使用率和提高系统效率。

OS 中引入线程目的：减少程序并发执行时所付出的时空开销，并发性更好。

5. 用户级线程

用户线程的维护由应用进程通过线程库来完成；

线程库：应用进程利用线程库提供创建、同步、调度和管理线程的函数来控制用户线程，无需内核支持。

特点：

内核不了解用户线程的存在；

用户线程切换不需要内核特权；

速度快。线程的创建和调度由应用软件内部进行，无需用户态/核心态切换，所以速度特别快。

优点：

线程切换不调用核心；

调度是应用程序特定的：可以选择最好的算法；

ULT 可运行在任何操作系统上（只需要线程库）。

6. 内核线程

有关线程的所有管理工作都在 OS 内核完成，应用程序部分没有线程管理的代码，只有一个到内核线程的 API

线程切换由内核完成；

内核维护进程和线程的上下文信息；

优点：

一个线程发起系统调用而阻塞，不会影响其它线程的运行，内核可以继续调度调度同一个进程中的另一个线程；

内核可以将同一进程中的多个线程调度到多个处理器上。

缺点：

由于有内核的参与，需要额外开销；

线程之间的切换需要内核的模式切换。

7. 多线程模型

多对一模型：将多个用户级线程映射到一个内核线程。

一对一模型：将每个用户线程映射到一个内核线程。一个线程阻塞时，另一个线程可以继续执行。

多对多模型：可以创建任意多的必要用户线程，且相应内核线程能在多处理器系统中并发执行；

一个线程阻塞时，另一个线程可以继续执行。

8. 线程池：

在进程建立时就创建若干线程，将这些线程放在一个“池”中等待工作。当服务器接收到一个请求时，就唤醒池中一个线程，并将要处理的请求传递给它；一旦线程完成了任务，它会返回到池中再等待其它的工作；如果池中没有可用的线程，服务器就会一直等待，指导有空闲线程为止。

优点：用现有线程处理请求通常臂等待创建新线程快；线程池限定了任何时候可存在线程的数量。

第五章 CPU 调度

1. 调度类型

1）长期（长程、高级）调度：从外存的后备队列中选择一个或者多个作业调入内存，并为它们创建进程，分配必要的资源。创建—>就绪/挂起；创建—>就绪。

2）中期（中程、中级）调度：将进程的部分或全部加载到内存中，提高内存利用率。进程状态变化（通过执行挂起和激活操作）：就绪/挂起<—>就绪；阻

塞/挂起<—>阻塞。

3）短期（短程、低级、CPU）调度：选择哪个进程在处理机上执行，执行最频繁)。进程状态：就绪<—>运行。

2. 抢占/非抢占、

1）非抢占（非剥夺、协作）调度：就绪进程不可以从运行进程手中抢占 CPU。一旦进程处于运行状态，它就不断执行直到终止或者为等待 I/O 或请求某些操作系统服务而阻塞自己，才把 CPU 让给别人。

2）抢占（剥夺）调度： 就绪进程可以从运行进程手中抢占 CPU。允许调度程序根据某种策略中止当前运行进程的执行，将其转移到就绪状态，并选择另一个进程投入运行。

3. 调度程序的功能

1）保存现场：记录放弃 CPU 的进程 A 的现场信息（如 PC，通用寄存器的内容等)。

2）选择进程：当进程出让 CPU 或调度程序剥夺执行状态进程占用的 CPU 时，选择适当的进程 B 分派 CPU。

3）完成上下文切换：用户态执行进程 A 代码，进入 OS 内核（通过时钟中断或系统调用)；保存进程 A 的上下文，载入进程 B 的上下文（CPU 寄存器和一些表格的当前指针)；用户态执行进程 B 代码。

4. 调度准则

面向用户的准则：响应时间 (min)；周转时间（结束时间-进入系统时间）(min)；优先级。

面向系统的准则吞吐量(max)；CPU 利用率(max)；公平；资源的平衡使用；系统开销(min)。

5. 调度算法

FCFS、SJF、抢占式 SJF、优先级、RR、最高相应比。给出进程到达机描述，掌握各种算法的调度顺序及个指标计算。

第六章 进程同步

1. 进程同步（直接要求同步)：指系统中一些进程需要相互合作，共同完成一项任务。具体说，一个进程运行到某一点时要求另一伙伴进程为它提供消息，在未获得消息之前，该进程处于等待状态，获得消息后被唤醒进入就绪态。

2. 互斥（间接要求同步)：由于各进程要求共享资源，而有些资源需要互斥使用，因此各进程间竞争使用这些资源，进程的这种关系为进程的互斥。

3. 一些相关概念：

互斥：指多个进程不能同时使用同一个资源；

死锁：指多个进程互不相让，都得不到足够的资源；

饥饿：指一个进程一直得不到资源（其他进程可能轮流占用资源）；

临界资源：系统中某些资源一次只允许一个进程使用，称这样的资源为临界资源或互斥资源或共享变量；

临界区：进程中访问临界资源的一段代码。

4. 使用临界区应遵循的准则：

空闲（有空）让进：当无进程在临界区时，任何有权使用临界区的进程可进入；

互斥（无空等待）：不允许两个以上的进程同时进入临界区；

多中择一：当没有进程在临界区，而同时有多个进程要求进入临界区，只能让其中之一进入临界区，其他进程必须等待；

有限等待：任何进入临界区的要求应在有限的时间内得到满足；

让权等待：处于等待状态的进程应放弃占用 CPU；

平等竞争：任何进程无权停止其它进程的运行进程之间相对运行速度无硬性规定。

5. 互斥的实现-硬件方法

（1）中断禁用 (关中断, Interrupt Disabling)

如果进程访问临界资源时(执行临界区代码)不被中断，就可以利用它来保证互斥地访问。

过程:

关中断原语;

临界区

开中断原语

其余部分

（2）专门的机器指令：设计一些机器指令，用于保证两个动作的原子性 ，如在一个指令周期中实现测试和修改。

6. 信号量

初始化指定一个非负整数值，表示空闲资源总数（又称为“资源信号量”）

若为非负值：表示当前的空闲资源数（s.count>= 0 可用的资源数 ：）

若为负值：其绝对值表示当前等待临界区的进程数(|s.count|为等待的进程数)

操作：操作系统对信号量只能通过初始化和两个标准的原语来访问。对信号量的操作只有三种

原子操作:

初始化：通常将信号量的值初始化为非负整数。

P 操作(wait 操作)：使信号量的值减 1(申请一个单位的资源(s.count--))

如果使信号量的值变成负数，则执行 P 操作的进程被阻塞(当 s.count < 0 时，资源已分配完毕，进程自己阻塞在 S 的队列上----让权等待)

V 操作(signal 操作)：使信号量的值加 1(释放一个单位资源(s.count++))

如果信号量的值不是正数，则使一个因执行 v 操作被阻塞的进程解除阻塞(若 s.count <= 0，则唤醒一个等待进程)。

需要掌握基本概念以及用信号量等机制解决同步问题。

第七章 死锁

1. 定义：一组进程中，每个进程都无限等待被该组进程中另一进程所占有的资源，因而永远无法得到资源，这种现象称为进程死锁，这一组进程就称为死锁进程。

2、产生原因：

资源不足导致的资源竞争：多个进程所共享的资源不足，引起它们对资源的竞争而产生死锁。

并发执行的顺序不当。进程运行过程中，请求和释放资源的顺序不当，而导致进程死锁. 如 P,V

操作的顺序不当。

3、四个必要条件：

互斥条件：指进程对所分配到的资源进行排它性使用，即在一段时间内某资源只能由一个进程占有。如果此时还有其它进程申请该资源,则它只能阻塞，直至占有该资源的进程释放。

占有且等待（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要求，而该资源又已被其它进程占有，此时请求进程阻塞，但又对已经获得的其它资源保持不放。

非抢占（非剥夺）条件：进程已获得的资源，在未使用完之前，不能被剥夺，只能在使用完时由自己释放。

循环等待条件：在发生死锁时，必然存在一个进程-资源的封闭的环形链. 即进程集合{P0, P1,P2,..., Pn}中的 P0 正在等待一个 P1 占用的资源; P1 正在等待 P2 占用的资源，...，Pn 正在等待已被 P0 占用的资源。

4、处理方法：

预防死锁：通过限制如何申请资源的方法来确保至少有一个条件不成立。

避免死锁：根据有关进程申请资源和使用资源的额外信息，确定对于一个申请，进程是否应该等待。

检测死锁和恢复：通过算法来检测并恢复。

忽视此问题：认为死锁不可能在系统内发生。如 Unix 采用这种方法。

5、死锁避免：不需象死锁预防那样，事先采取限制措施破坏产生死锁的必要条件 ；在资源的动态分配过程中，采用某种策略防止系统进入不安全状态，从而避免发生死锁。

定义：在系统运行过程中，对进程发出的每一个系统能够满足的资源申请进行动态检查，并根据检查结果决定是否分配资源，若分配后系统可能发生死锁，则不予分配，否则予以分配。

6、安全状态：系统能按某种顺序，如<P1，P2，　，Pn>，为每个进程分配所需资源，直到最大需求，　使每个进程都可顺序完成，称系统处于安全状态。

只要系统处于安全状态，必定不会进入死锁状态；死锁状态是不安全状态。

不安全状态不一定是死锁状态（不安全状态可能导致死锁）。

7、银行家算法：

Available[j]：尚未分配的资源 j 的数量；

Max[i，j]（Claim[I,j]）：进程 i 对资源 j 的最大需求量；

Allocation[i，j]：进程 i 获得的资源 j 的数量；

Need[i，j]：进程 i 尚需的资源 j 的数量。

8、死锁恢复方法：

进程终止：终止所有的死锁进程－OS 中常用方法；一次只终止一个进程直到取消死锁循环为止－基于某种最小代价原则。

资源抢占：逐步从进程中强占资源给其它进程使用，直到死锁环被打破为止。

9、资源分配图：

(1)表示方法：

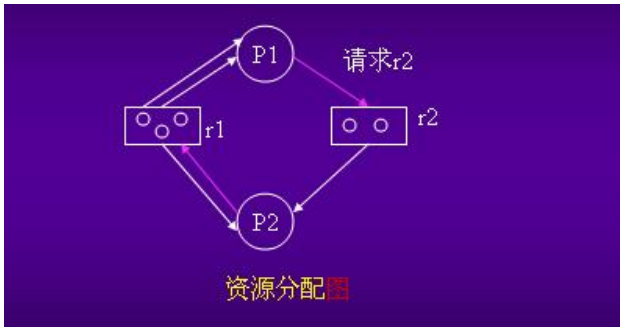
圆圈代表进程，方块代表一类资源。

方框中的点：由于一种类型的资源可能有多个，可用方框中的一个点代表一类资源中的一个资源。

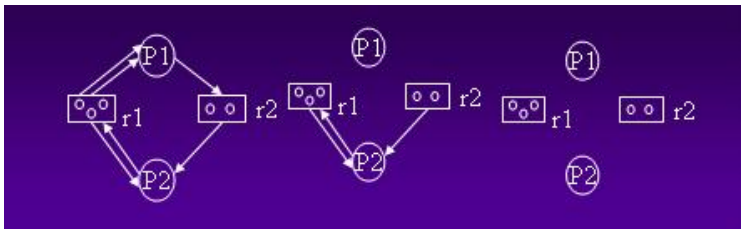
分配边：从资源节点(圆圈)到进程节点(方块)的有向弧表示资源已经分配给进程；

请求边：从进程到资源的有向弧表示进程当前正处于阻塞状态，等待资源变为可用。

(2) 有向图形成环路则形成死锁。



(3)化简方法：假设某个 RAG 中存在一个进程 Pi，此刻 Pi 是非封锁进程，那么可以进行如下化简：当 Pi 有请求边时，首先将其请求边变成分配边(即满足 Pi 的资源请求)，而一旦 Pi 的所有资源请求都得到满足，Pi 就能在有限的时间内运行结束，并释放其所占用的全部资源，此时 Pi 只有分配边，删去这些分配边（实际上相当于消去了 Pi 的所有请求边和分配边)，使 Pi 成为孤立结点。（反复进行）



系统中某个时刻 S 为死锁状态的充要条件是 S 时刻系统的资源分配图是不可完全简化的。

在经过一系列的简化后，若能消去图中的所有边，使所有的进程都成为孤立结点，则称该图是可完全简化的；反之的是不可完全简化的。

第八章 内存管理

1、地址重定位：将逻辑地址转变为物理地址的过程。

（1）静态地址重定位：在目标程序装入内存时，由装入程序对目标程序中的指令和数据的地址进行修改，即把程序的逻辑地址都改成实际的物理内存地址。当用户程序被装入内存时，一次性实现逻辑地址到物理地址的转换，以后不再转换。程序的存储空间只能是连续的一片区域不能再移动。

（2）动态地址重定位：在程序运行过程中要访问内存数据时再进行地址变换，即在逐条指令执行时完成地址映射。OS 可以将一个程序分散存放于不连续的内存空间，可以移动程序。

2、覆盖与交换

（1）覆盖：

原理：在任何时候只在内存中保留所需的指令和数据；当需要其它指令时，它们会装入到刚刚不再需要的指令所占用的内存空间。

特点：覆盖不需要 OS 提供特殊的支持，但程序员必须适当地设计和编写覆盖结构。

（2）交换：

原理：暂停执行内存中的进程，将整个进程的地址空间保存到外存的交换区中（换出），而将外存中由阻塞变为就绪的进程的地址空间读入到内存中，并将该进程送到就绪队列（换入）。交换单位为整个进程的地址空间。

与覆盖的比较：与覆盖技术相比，交换技术不要求用户给出程序段之间的逻辑覆盖结构。交换发生在进程或作业之间，而覆盖发生在同一进程或作业内。此外，覆盖只能覆盖那些与覆盖段无关的程序段。

3、连续内存分配—固定分区：

把内存分为一些大小相等或不等的分区(partition)，每个应用进程占用一个或几个分区。操作系统占用其中一个分区。分区的划分原则一般由系统操作员或操作系统决定，分区一旦划分结束，在整个执行过程中每个分区的长度和内存的总分区个数将保持不变 。

特点：适用于多道程序系统和分时系统、支持多个程序并发执行、难以进行内存分区的共享。

问题：可能存在内碎片和外碎片。

分区大小相等和分区大小不相等两种情况。

4、连续内存分配—可变分区：

内存不是预先划分好的，而是当进程装入时，根据进程的需求和内存空间的使用情况来决定是否分配。若有足够的空间，则按需要分割一部分分区给该进程；否则，令其等待主存空间。

评价：没有内碎片；在开始时是很好的，但最后，导致在存储器中出现很多空洞－外部碎片；

解决方法：压缩。

压缩（compaction）：OS 不时地移动进程，将它们放在一起，并且使所有空闲空间连成一片。

压缩非常费时，浪费了处理器的时间。压缩需要动态重定位的能力，即必须能够将程序从主存的一块区域移动到另一块区域，而不会使程序中的存储器访问无效。

分配算法：首次适配法（分区按起始地址递增的顺序排列）、最佳匹配法（按空闲区大小从小到大的顺序排列）、最差匹配法（按空闲区大小从大到小的顺序排列）、临近匹配法（到最后分区时再回到开头）。

5、紧缩（compaction）：移动内存内容，以便所有空闲空间合并成一整块。

条件：允许进行动态重定位可以首先移动程序和数据，然后再根据新基地址的值来改变基地址寄存器。

实现：一种简单的方法是将所有进程移动到内存的一端，而将所有的空闲区(孔)移动内存的另一端，以生成一个大的空闲块。

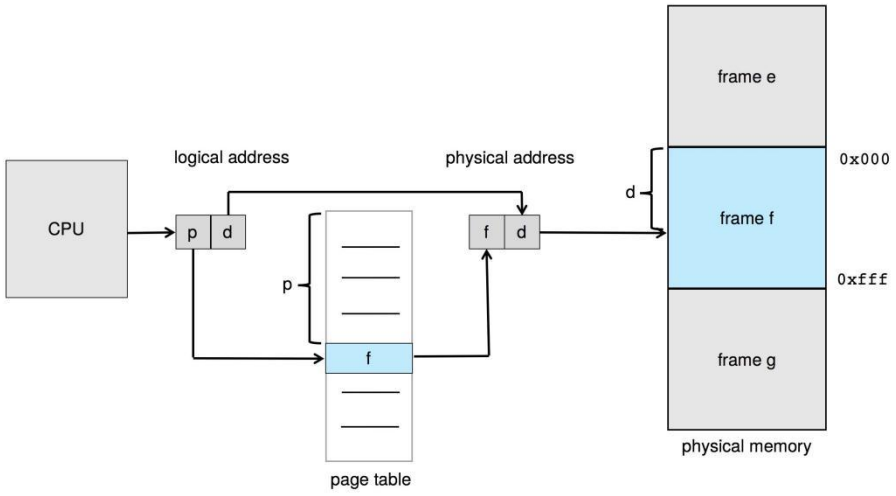
开销：开销大。

6、分页管理

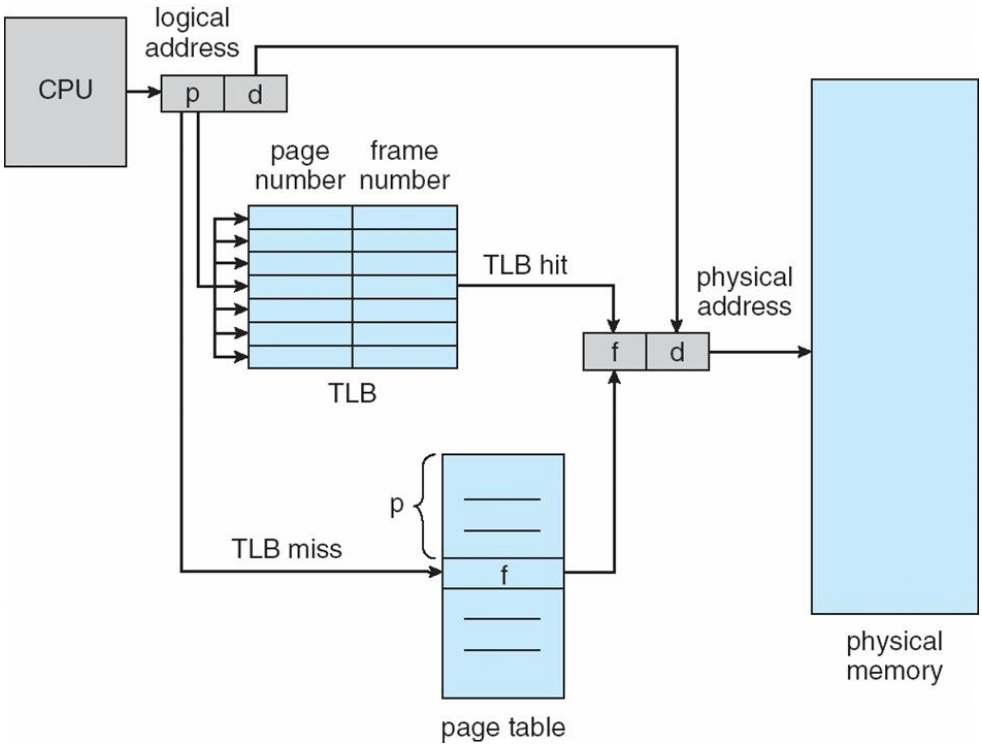
1）分页机制概述：

进程的物理地址空间可以是非连续的；只要物理内存可用，就为进程分配物理内存，这样避免外部碎片，避免了大小不一的内存块问题，将物理内存划分为称为帧的固定大小的块(帧，页框)，大小是 2 的幂，介于 512 字节和 16 MB 之间将逻辑内存划分为大小相同的块，称为页(页)，跟踪所有可用帧，要运行大小为 N 页的程序，需要找到 N 个可用帧并加载程序；设置页面表(页表) 将逻辑地址转换为物理地址；备份存储同样拆分为页面。仍有很少的内部碎片。

2）单级页表



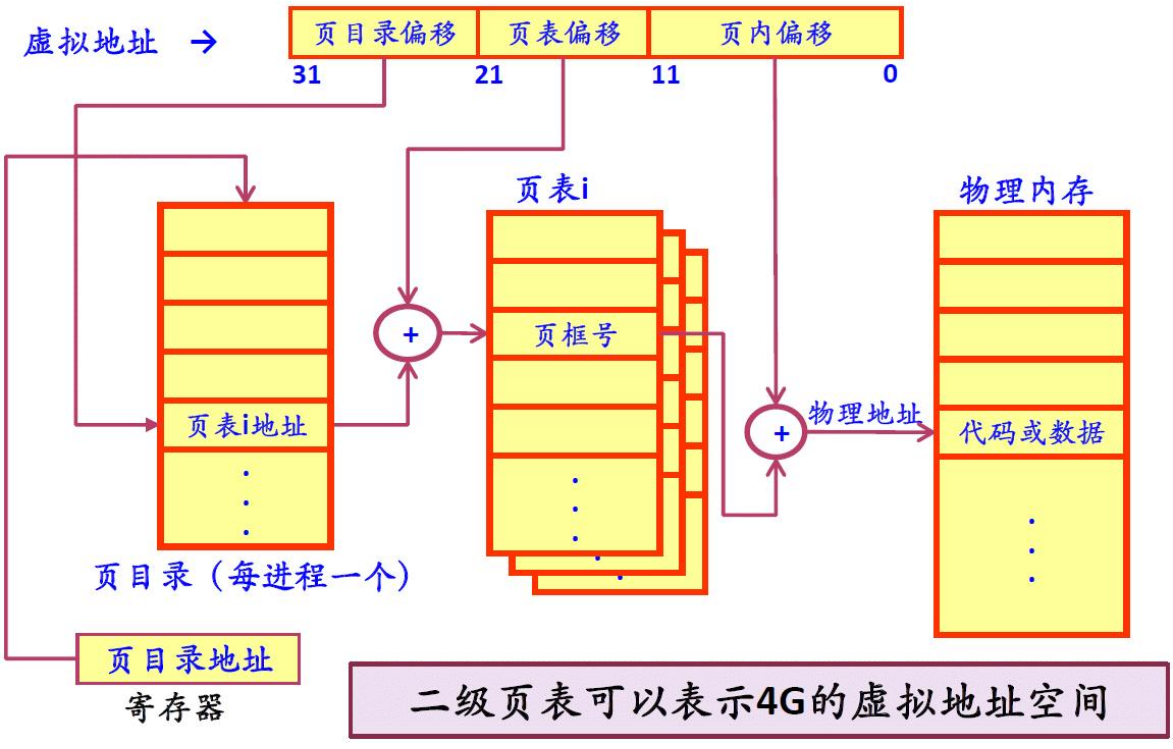
访问内存变为两次，效率下降，采用 TLB 缓存提高效率：



3）多级页表（二级以上）

随着内存容量的扩大，页表变大，因页表要求连续内存，造成内存管理变得困难，采用多级页表，使得内层页表不必要连续。

二级页表示意如下：



需要根据要求给出页表设计方案。

第九章 虚拟内存管理

1、虚拟内存及其作用

虚拟内存是计算机系统内存管理的一种技术。它使得应用程序认为它拥有连续的可用的内存（一个连续完整的地址空间），而实际上，它通常是被分隔成多个物理内存碎片，还有部分暂时存储在外部磁盘存储器上，在需要时进行数据交换。这样可以在内存中同时加载更多的进程，提高资源的利用率，尤其是 CPU 的利用率，目前，大多数操作系统都使用了虚拟内存技术。

2、局部性原理

程序在执行的一段较短的时期，所执行的指令地址、相应操作数据地址，分别局限于一定区域内，这样不必要把整个程序加载到内存。

3、虚拟内存情况下页表的改变：增加标志 P：表示该页是否在内存中，如果在，对应页条目中包含了对应的物理帧号；M：修改位，表示该页内容在上次装入后修改过；还有其他一些用于保护和共享的位。

4、缺页中断：在地址映射过程中，在页表中发现所要的页不在内存，则产生缺页中断，缺页中断处理程序会申请物理帧，并把相应页调入，并修改页表项。然后恢复指令执行。

5、Beladay 异常：有些情况下，缺页数会随着所分配的物理帧数的增加而增加。

6、掌握几种替换页选择方法：最佳算法、先进先出、最近最久未使用，能给出页面替换序列和次数。

7、系统颠簸。（现象描述、产生原因、解决办法）。

8、请求段式管理

9、虚拟段页式管理。

第十章 文件系统

1、基本概念

1）文件：文件是命名的数据流、连续的逻辑地址空间。

2）文件的几种类型：数据文件、程序文件、目录文件...

3）文件属性：名字、标识符、类型、位置、大小、权限、时间戳等描述文件的元数据。

4）目录：用于组织文件的文件，每个条目对应一个目录或普通文件。

5）文件共享：同步锁（强制锁、建议锁）

6）文件保护：权限。

7）虚拟文件系统

2、目录的结构

1）单层：存在命名问题、分组问题。

2）两层：目录名、不同用户可用同文件名（部分解决应用命名问题）、搜索快，不支持分组。

3）树结构：解决命名、分组问题，搜索快。

4）无环图：在树结构基础上可以用别名，更方便共享。

掌握以上概念

第十一章 文件系统实现

1、文件系统结构

- 应用程序：用户程序进行文件系统操作
- 逻辑文件系统：管理文件系统元数据(文件系统结构，FCB)，为应用提供 API(应用程序接口)
- 文件组织模块：转换逻辑块地址到物理块地址；空闲空间管理
- 基本文件系统：向设备驱动程序发出通用的命令，读写物理数据块；管理文件系统的缓冲区及 cache
- 控制：设备驱动程序及中断处理程序，输入输出操作控制，发出的指令为特定设备的格式
- 设备：提供永久存储媒介，数据块。

2、文件系统实现

1）文件系统对应结构

- 引导控制块
- 超级块
- 目录结构
- 文件控制块

2）内存中的结构

- 安装表
- 系统范围打开文件表
- 单个进程打开文件表

3、目录实现

目录项目存储的线性表、Hash 表。

4、分配方法：连续分配、链接分配、索引分配，各种分配方法的优缺点。实际实现一般用混合机制。

5、空闲空间管理：位图、链表、块组链表、第一空闲块+计数。

6、效率与性能：提高性能的方法（缓冲/缓存），包括目录缓存、预读取、后台写，内存磁盘等。

第十二章 大规模存储

1、大规模存储设备

磁盘、固态盘、可移动磁盘、光盘、磁带、存储阵列、网络存储、云存储等。

2、磁盘调度

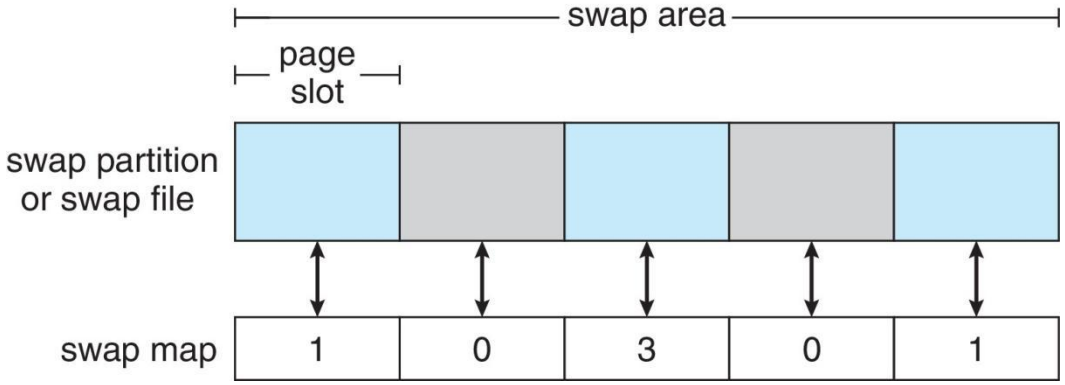
寻道延迟（Seek Time）、Rotate Latency。

FCFS、SSTF、SCAN、C-SCAN、LOOK、C-LOOK

3、磁盘管理

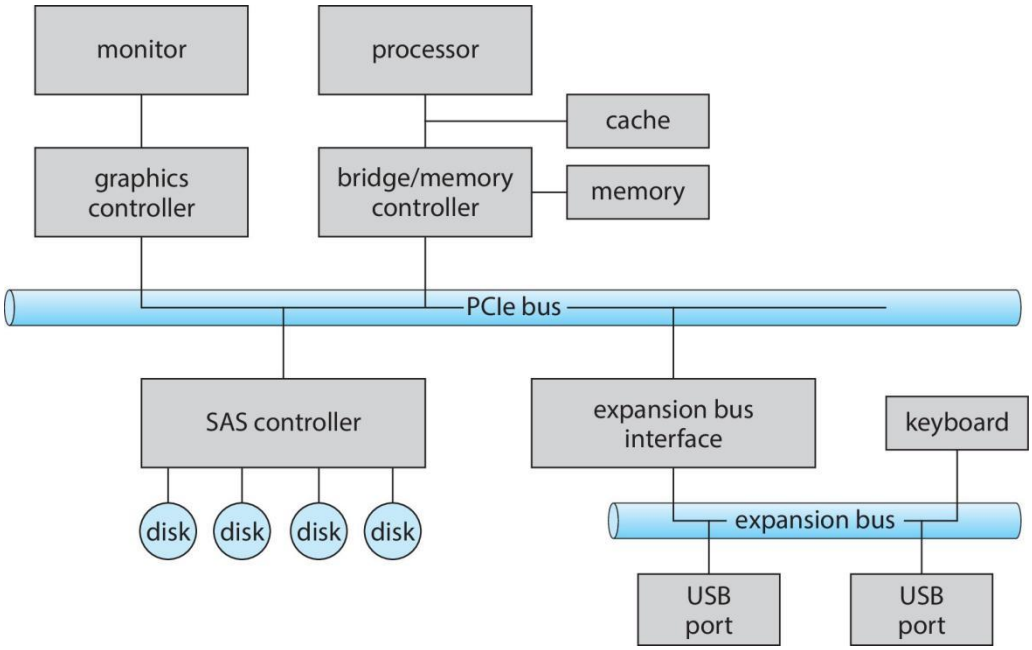
低级（物理）格式化、分区、逻辑格式化。

4、交换空间管理



第十三章 IO 系统

1、IO 硬件



2、IO 方式

轮询、中断、DMA。

3、IO 应用接口

1）两种方式接口：阻塞（同步）、非阻塞（异步）。

2）不同设备类型的抽象