第一章 操作系统概述

操作系统的定义:计算机系统中的一个系统基础软件，它是这样一些程序模块的集合：它们能有效地组织和管理计算机系统中的硬件及软件资源，合理地组织计算机工作流程，控制程序的执行，并向用户提供各种服务功能及相应接口，使用户能够灵活、方便和有效地使用计算机，使整个计算机系统能高效地运行。作为服务提供者（接口）的操作系统：程序开发，程序运行，I/O设备访问，文件访问控制，系统访问，错误检测和响应，记账。作为资源管理者的操作系统：管理CPU(运算器，控制器)，内存，外存，IO操作系统的演化过程：串行处理-特点：用户程序独占所有资源，效率极低，作业准备时间长、必须人工调度，无操作系统。简单批处理系统-监控程序从磁带自动识别一个作业，处理后再取下一个，脱机读入作业卡片。缺点：CPU利用率低考虑问题：内存保护-不能修改监控程序所在的内存，特权指令-只能由监控程序执行的指令，中断-使用监控程序的合理方式，定时器-防止作业独占系统。运行模式：用户模式-特权指令不允许，不能访问受保护内存；内核模式/系统模式/控制模式-特权指令允许，可访问~常驻监控程序：大部分监控程序必须总是处于内存中且可以执行；非常驻监控程序：一些实用程序和公用函数，可以作为用户程序的子程序，在需要时才被载入。多道批处理系统：系统中同时驻留多个作业，共享内存，复用CPU。优点：提高CPU利用率，提高内存和IO设备利用率以及系统吞吐率。特征：无序性，需要内存管理，需要调度算法，缺乏交互能力，作业周转时间长，系统开销大，运行过程不确定。分时系统：相比多道批处理系统，CPU在程序间切换设定了固定时间-时间片；有存储器调度：作业多、主存不能同时容下时，则程序在完成之前，可以写回磁盘上，需要时再读回。嵌入式操作系统：在硬件资源，可靠性，温度湿度，软硬件可裁剪方面有严格要求。特点：1）系统内核小2）专用性强3）系统精简4）多任务操作系统5）需要开发工具和环境。实时操作系统：当外界事件或数据产生时，能够接受并以足够快的速度予以处理，其处理的结果又能在规定的时间之内来控制生产过程或对处理系统做出快速响应，调度一切可利用的资源完成实时任务，并控制所有实时任务协调一致运行的操作系统。核心特点：精确的任务调度、可预测的响应时间、稳定性和高可靠性。通常采用抢占式的优先级高者优先算法。典型实时操作系统：VxWorks，RT-Thread，uCOS，QNX，WinCE。分布式操作系统：若干计算机相互协作共同完成一项任务，系统资源所有用户共享，计算机间相互通讯，数据传输。多级系统好像具有单一存储空间和统一的文件系统操作系统重要理论进展：进程，内存管理，信息保护和安全，调度和资源管理。内存管理：进程隔离：存储空间相互独立；自动分配和管理：按需动态分配，对程序员透明；支持模块化程序设计；保护和访问控制；长期存储。虚存机制：允许程序以逻辑方式访问存储器而不考虑物理内存上可用的空间数量，满足多个进程同时驻留在内存中的需求。分页系统：固定大小的连续块称为页，虚地址由页号和页中偏移量组成，分页系统完成由虚地址到实地址的映射。信息保护和安全：可用性，保密性，数据完整性，认证。调度和资源管理：公平有效，差别响应 微内核体系结构：提供操作系统核心功能的内核的精简版本，提供模块化设计，将通常与内核集成在一起的系统服务层被分离出来，变成可以根据需求加入的选件，仅将最基本的功能作为内核，包括最小的进程管理、内存管理和通信功能，其他功能作为系统程序或用户程序出现。采用CS模式，微内核经常扮演消息传递的功能。客户通过微内核发送消息给文件服务器。多线程：把执行一个应用程序的进程划分为可以同时执行的线程。对称多处理：多个处理器，共享内存和I/O设备，互联。所有处理器可以执行相同的功能操作系统可调度进程或线程到所有的处理器上运行。操作系统负责在多个处理器间调度进程和线程，并负责处理处理器间的同步虚拟化：通过在软件和物理硬件之间提供一个软件转换层（称为抽象层）来管理计算机资源的一组技术。主要包括虚拟机和虚拟化容器两方法。虚拟机：单PC/服务器可同时运行多个操作系统或单个操作系统多个会话。容器只隔离应用程序的运行时环境，但容器之间可以共享同一个操作系统，这里的运行时环境指的是程序运行依赖的各种库以及配置。更加的轻量级且占用的资源更少

第二章 进程

程序运行过程：源代码-（编译，汇编，链接）->可执行文件-（加载，重定位）->程序在内存中运行->寄存器组，CPU 并行：同时执行。并发：同时出发，交替执行（但切换很快，像是同时执行）。并发先保存执行现场，后切换执行现场。进程定义：一个正在执行的程序，计算机中正在运行的程序的一个实例，可分配给处理器并由处理器执行的一个实体，由一个按顺序执行的程序，一个当前状态和一组相关系统资源所表征的活动单元。进程五个基本特征：动态性-进程生命周期，并发性-提高资源利用率，独立性-程序运行的基本单位，系统进行资源分配和调度基本单位，异步性-以独立不可预知速度前进，交互性-可能与其他进程发生直接或间接相互作用。进程五状态：新建，就绪，运行，阻塞，退出。新建-允许进入-就绪-进程调度-运行-等待事件发生-阻塞-事件已发生-就绪；或运行-中断（包括超时等）-就绪，运行-完成/终止-退出。新建状态：至少建立PCB但进程相关其他内容可能未调入主存；就绪：在某时刻，进程已获得除处理机以外的所有资源，一旦分到了处理机就可以立即执行；运行：进程已经获得必要资源，并占有处理机运行；阻塞：正在执行的进程，由于发生某事件而暂时无法执行下去，如IO请求等；退出：进程已经终止，但资源等待父进程或系统回收。创建进程的原因：新的批处理作业，交互登录，为提供服务而由操作系统创建，由现有进程派生。进程队列：为了管理和调度进程，操作系统将具有相同属性或特征的进程保存到的队列数据结构。通常以链表队列形式实现，元素是PCB进程控制块。进程七状态：加上就绪挂起，阻塞挂起。挂起进程的特点：不能立即执行，可能在也可能不在等待一个事件，若等待一个事件，那么阻塞条件不依赖挂起条件，阻塞事件的发生不会使进程立即执行。阻止进程执行可以通过代理使其处于挂起状态。除了代理显示地命令系统进行状态转换，否则该进程无法从这一状态转移。挂起原因：交换-系统要释放足够内存空间，其他OS原因-进程有问题或工具进程，交互式用户请求，定时-周期性执行，父进程请求-编程需要。进程映像：由操作系统建立，包括代码，程序所需相关数据和记录程序执行的上下文环境或运行时信息。PCB：线性的数据结构，用来描述和记录进程的动态变化信息。系统创建一个进程，就是由系统为某个程序设置一个PCB；进程执行完成时，由系统收回其PCB，该进程便消亡了。系统将根据PCB而感知进程的存在，故PCB是进程存在的唯一标志。通过PCB可以访问到进程的所有信息。表征进程的以下信息存在PCB中：标识符（该进程，父进程，用户），状态，优先级，程序计数器，内存指针，上下文数据，IO状态信息，记账信息，进程间通信。进程是资源的拥有者。表：为了管理进程和资源，操作系统构造并维护其管理和维护的每个实体的信息表，如内存，IO，文件和进程表。例子：对于一个进程实体，C语言程序中全局变量在正文段，未赋值局部变量在栈段，未赋值全局变量在正文段，函数调用时参数在栈，malloc动态分配的存储区在堆段，进程优先级在PCB中。降低进程优先级的合适时机是进程时间片用完。进程控制模式：特权模式下软件完全控制处理器及其所有指令，寄存器和内存。作用：分执行模式可以保护操作系统或操作系统表（如PCB）不被用户程序干扰。具体：调用一个操作系统的服务或中断来触发系统例程的执行时，执行模式将被置为内核模式当从系统服务返回到用户进程时，执行模式将被置为用户模式。进程创建：由创建原语实现，主要功能是为被创建进程形成一个PCB并填入相应初始值。过程：为新进程分配一个唯一的进程标识符-为进程分配空间-再根据父进程所提供的参数将子进程的PCB初始化-将此PCB插入就绪队列/就绪挂起/新建队列-创建或扩充其他数据结构。进程的切换：在操作系统从当前运行的进程中获得控制权的任何时刻发生-如中断（时钟，IO，内存失效），陷阱，系统调用。操作过程：（调度问题）保存处理器上下文，更新该进程PCB并移动到相应的队列，选择另一进程，更新内存管理数据结构，恢复选中进程运行现场。进程撤销：由撤销原语实现，两种撤销策略：只撤销一个具有指定标识符的进程/撤销指定进程及子孙进程。过程：找到被撤销进程的PCB-若运行中则停止执行，设置调度标志-对后一种撤销策略，若被撤销进程有子孙进程，还应将该进程的子孙进程予以撤销-被撤销进程所占有的资源，或者归还给父进程，或者归还给系统-撤销PCB。阻塞进程：使用阻塞原语将进程由执行转为阻塞（进程自身调用）。唤醒原语则是由阻塞变为就绪（发现者进程/中断服务程序/OS调用）。过程：中断处理机，保存CPU现场-将该进程插入到等待该事件的队列中-从就绪队列中选择一个进程投入运行。操作系统的控制：无进程内核/例程在用户进程内运行/基于进程的操作系统。

第三章 线程

进程是资源+指令执行序列，将资源和指令执行分开，得到一套资源和多个指令执行序列，序列就是线程。线程（轻量级进程）保留并发优点，避免进程切换代价，实质是内存映射表和资源不变，PC指针和基本寄存器改变。单进程单线程：MSDOS；单进程多线程：Java；多进程，每个进程一个线程：hpux；多进程多线程：linux，windows，solaris。多线程环境下，进程是资源分配单元和保护单元：容纳进程映像的虚拟地址空间，对处理器，其他进程，文件和IO资源的受保护访问。线程包含：执行状态（运行，就绪），线程上下文，用于局部变量的静态存储空间，与进程内其他线程共享的内存和资源访问。多线程优点：响应度高（部分阻塞其他部分仍能执行），资源共享，经济快捷（节省资源，线程创建终止切换快），方便多CPU处理（由用户编写，充分考虑任务有效分离-多序列，且序列隶属同一个进程地址空间，充分利用多CPU处理）线程状态：运行态，就绪态，阻塞态。与线程状态改变相关的基本操作：派生，阻塞，解除阻塞，结束。线程两种实现方法：用户层用户级线程-受内核支持但不需内核管理，内核层内核级线程-由操作系统直接支持管理。线程创建：线程都是由用户程序创建的。操作系统可以创建对应的数据结构对这些线程直接进行管理，并以线程为调度单位，这是内核线程；完全不“感知”线程的存在，仍然以进程为管理和调度单位，线程的管理与调度完全由用户级程序来完成（当然也可能由编程语言提供用户级的库函数支持，即线程库，用于创建和销毁线程，调度线程执行等）是用户线程。用户级线程：所有活动发生在用户空间。用户级线程优点：不需要切换到内核模式，可以为应用程序量身定做调度算法，可在任何操作系统中运行。缺点：系统调用阻塞进程中所有线程，不能利用多处理器。内核级线程：管理线程所有工作均由内核完成，内核为进程及进程内每一个线程维护上下文信息，调度由内核基于线程完成，内核例程也是多线程的，线程切换需要切换到内核模式。相比于用户级线程的每个线程中各一个线程表，用户级线程的线程表在内核空间里且只有一个。混合方法：线程的创建在用户空间中完成，线程的调度和同步也在应用程序中进行，一个进程的多个用户级线程会被映射到一些内核级线程上，同一应用程序的多个线程可以在多个处理器上并行运行，会引起系统阻塞的系统调用不会阻塞整个进程。“线程可以从用户空间伸展到内核空间”。以上三种对比：用户级线程利用多核差，并发度低，切换代价小，OS代码无，用户灵活性大，CPU资源分配少；核心级线程：好高大大小多；用户加核心：好高中大大中。三种多线程模型优缺点（多用户线程对一内核线程，多对多，一对一）：多对一：优：切换快速，用户线程数量几乎没有限制；缺：一个用户线程阻塞，所有无法执行，多处理器系统上处理器的增多线程性能不会有明显帮助。一对一：优：用户线程具有和内核线程一致优点，提供更好并发性，一个用户线程发起阻塞时允许另一个线程运行。缺：用户线程数量有限制，切换上下文开销大。多对多：优：不限制应用进程树=数，多线程可以并发。

第四章 并发：互斥和同步

进程间的同步：对多个相关进程在执行次序上进行协调，以使并发执行的诸进程之间能有效地共享资源和相互合作，从而使程序的执行具有可再现性。进程/线程同步原因：多个进程/线程共同完成一个任务，合作过程中除了并行的工作还经常出现相互等待的协作过程。竞争条件：多个进程/线程并发访问和操作同一（组）数据（即共享数据）,且执行结果与访问发生的特定顺序有关。临界资源：一次仅允许一个进程使用的资源称为临界资源。如只能独享的物理设备(打印机)、共享变量等等。临界区：在每个进程中，访问临界资源的那段程序称为临界区。临界区定义了进程间的协作协议。互斥：当一个进程在临界区访问共享资源时，其它进程不能进入该临界区访问任何共享资源的情景。解决临界区问题：1.互斥进入：如果一个进程在临界区中执行，则其他进程不允许进入2.有空让进：若干进程要求进入空闲临界区（临界资源空闲）时，应一定能使某个进程进入临界区3.有限等待：从进程发出进入请求到允许进入，不能无限等待。死锁：两个或两个以上的进程因每个进程都在等待其它进程做完某些事情，而不能继续执行的情形。活锁：两个或两个以上的进程为响应其它进程中的变化，而持续改变自己的状态，但不做有用的工作的情形。互斥软件解决方案：Dekker算法：进程P0：临界区：flag[0] = true;while (flag[1]){if(turn==1){flag[0]=false;while(turn==1);flag[0]=true; } }剩余部分：turn=1;flag[0] = false;进程P1：临界区：flag[1] = true;while (flag[0]){if(turn==0) {flag[1]=false;while(turn==0);flag[1]=true; } }剩余部分：turn=0;flag[1] = false;Peterson算法：进程P0：临界区：flag[0] = true;turn = 0;while (flag[1] && turn == 0) ;剩余部分：flag[0] = false;进程P1：临界区：flag[1] = true;turn = 1;while (flag[0] && turn == 1) ;剩余部分：flag[1] = false;多个进程：面包店算法：每个进入商店的客户都获得一个当前号码，号码最小的先得到服务；号码相同时，名字靠前的先服务。进程pi：临界区：choosing[i] = true; num[i] = max(num[0], …, num[n-1])+1;choosing[i] = false; for(j=0; j<n; j++) { while(choosing[j]);while ((num[j] != 0) && (num[j], j)<(num[i], i)); }剩余部分：num[i] = 0;互斥解决硬件方案：专用机器指令：处理器的设计者提出了一些机器指令，用于保证两个动作的原子性；在一个指令周期中完成；在指令执行过程中，其它指令访问内存将被阻止；使用比较和交换指令，如compare\_and\_swap或exchange。专用机器指令方法优缺点：优：适用于多处理器，简单，可支持多个临界区；缺：使用了忙等待，可能饥饿，可能死锁。饥饿：由于别的并发的激活的进程持久占有所需资源,使某个异步进程在可预测的时间内不能被激活的情形。原子操作：不会被进程（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，中间不会有任何切换。信号量semaphere：Dijkstra提出的用于进程间传递信号的一种特殊整形变量。只有三种操作：初始化-初始化成非负数；递减-semwait原子操作，减1，若变成负数则阻塞执行该操作进程(并移入队列)，否则继续运行；增加-semsignal原子操作，加1，若值小于等于0，则被semwait阻塞的进程解除阻塞，移出队列，放到就绪列表。二元信号量：值只能是0或1.只有三种操作：初始化：可以初始化0或1；semwaitB原子操作：检查信号，若值为0，则执行该操作的进程就会被阻塞（并移入队列），若值为1，则变为0，并继续运行。semsignalB原子操作：检查是否有进程在该信号上受阻，若有则受阻的进程则会被唤醒（并移出队列，放到就绪列表），若没有受阻的进程，则信号量的值改为1。非二元信号量也被称为计数信号量(Counting Semaphore)或一般信号量(General Semaphore)。互斥锁（mutex）与二元信号量的区别：同进程解锁。强信号量：遵循先进先出（FIFO）的原则来处理等待信号量的进程，进程从阻塞队列中移出的顺序遵循先入先出（FIFO）策略，多个进程等待同一个信号量时，最早请求该信号量的进程将最先得到服务被阻塞时间最久的进程最先从队列释放，可以避免发生饥饿。

二元信号量解决无限缓冲区生产者消费者正确方案：int n；binary semaphore s=1,delay=0;void producer(){while(true){produce()；semwaitB(s);append();n++；if(n==1)semsignalB(delay);semsignalB(s);}}

void consumer(){int m; /\* a local variable \*/semwaitB(delay);while(true){semwaitB(s);take()；n--；m = n；semsignalB(s);consume();if(m==0)semwaitB(delay)；}}

void main(){n = 0;parbegin(producer,consumer);}其中，delay用于实现生产与消费的同步，delay=0表示没有数据在缓冲区等待消费，s=1表示缓冲区空。

使用信号量解决无限缓冲区生产者消费者：semaphore n=0s=1；void producer(){while(true){produce()；semwait(s)；append();semsignal(s)；semsignal(n);}}

void consumer(){while(true){semwait(n)；semwait(s)；take()；semsignal(s);consume();}

void main(){parbegin(producer,consumer);}其中，n用来记录缓冲区数据项数量，s=1表示缓冲区未被占用。

使用信号量解决有限循环缓冲区生产者消费者问题：设e是缓冲区大小，则在上面基础上，生产者在semwait（s）前加上semwait(e)，在消费者semsignal（s）后加上semsignal（e）。

管程：一种程序设计语言结构，由一个/多个过程，一个初始化序列和局部数据组成的软件模块组成，主要特点:局部数据变量只能被管程的过程访问;进程通过调用管程的一个过程来进入管程;任何时候，只能有一个进程在管程中执行，调用管程的其它进程都被阻塞。管程通过使用条件变量来支持同步，这些条件变量包含在管程中，包括：cwait(C), 调用者阻塞在条件C上，且管程可被其它进程使用；csignal(C), 恢复阻塞在C上的进程中的一个执行。对生产者消费者使用管程，所有同步机制都被限制在管程内部，因此不但易于验证同步的正确性，而且易于检测出错误。消息传递：原语：发送消息send(destination, message)，接收消息recieve(source, message)同步机制：阻塞send阻塞receive，无阻塞send阻塞receive，无阻塞send阻塞receive。寻址模式：直接/间接寻址

使用消息解决有界缓冲区生产者消费者问题（无阻塞send，阻塞receive）：const int capacity = /\* buffering capacity \*/;null = /\* empty message \*/;

int i;

void producer(){message pmsg;while (true) {receive(mayproduce, pmsg);pmsg = produce();send(mayconsume, pmsg);}}

void consumer(){message cmsg;while (true) {receive(mayconsume, cmsg);consume(cmsg);send(mayproduce, null);}}

void main(){create\_mailbox(mayproduce);create\_mailbox(mayconsume);for (int i = 1; i <= capacity; i++)send(mayproduce, null);parbegin(producer, consumer);}

读者/写者问题：任意数量的读进程(reader)可以同时读取共享数据区，一次只有一个写进程(writer)可以修改共享数据区，若一个写进程正在修改数据区，则禁止任何读进程读取共享数据区。不需要调整队列指针，谢锦诚不需要判断是否队列已满，读进程不需要判断是否队列已空。

读者优先：/\* program readersandwriters \*/int readcount;semaphore x = 1, wsem = 1;

void reader(){while (true) {semWait(x);readcount++;if (readcount == 1)semWait(wsem);semSignal(x);READUNIT();semWait(x);readcount--;if (readcount == 0)semSignal(wsem);semSignal(x);}}

void writer(){while (true) {semWait(wsem);WRITEUNIT();semSignal(wsem);}}

void main(){readcount = 0;parbegin(reader, writer);}代码允许读者在没有写者等待的情况下优先访问资源。

写者优先：/\* program readersandwriters \*/int readcount, writecount;semaphore x = 1, y = 1, z = 1, wsem = 1,rsem = 1;

void reader(){while (true) {semWait(z);semWait(rsem);semWait(x);readcount++;if (readcount == 1)semWait(wsem);semSignal(x);semSignal(rsem);semSignal(z);READUNIT();semWait(x);readcount--;if (readcount == 0)semSignal(wsem);semSignal(x);}}

void writer(){while (true) {semWait(y);writecount++;if (writecount == 1)semWait(rsem);semSignal(y);semWait(wsem);WRITEUNIT();semSignal(wsem);semWait(y);writecount--;if (writecount == 0)semSignal(rsem);semSignal(y);}}

void main(){readcount = writecount = 0;parbegin(reader, writer);} 信号量 z，用于控制对 readcount 的互斥访问。信号量 rsem，判断是否允许读者读取。

第五章 调度

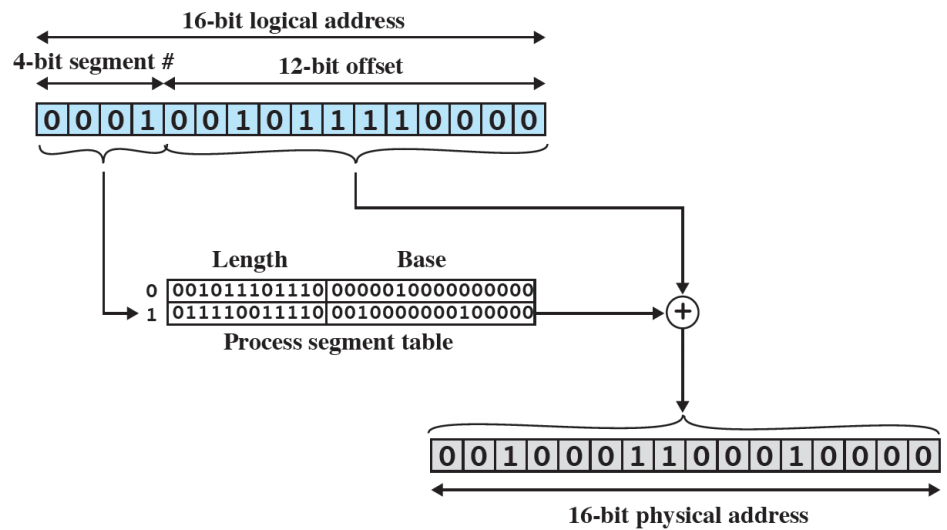
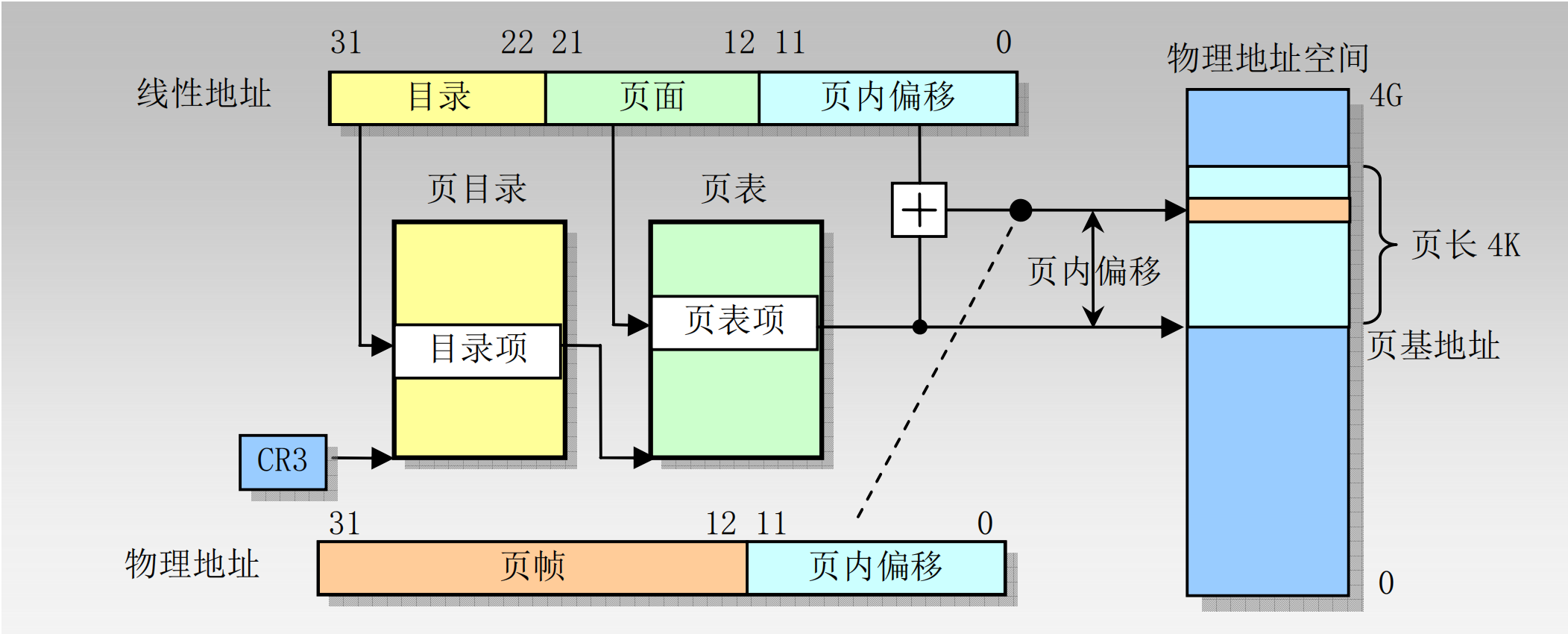
调度层次：长程调度（作业调度，高级调度）：将已进入系统并处于后备状态的作业按某种算法选择一个或一批，为其建立进程，并进入主机，当该作业执行完毕时，还负责回收系统资源。批处理系统需要作业调度，以便将进程分批地装入内存，在分时系统和实时系统中，通常不需要长程调度。它的频率比较低，主要用来控制内存中进程的数量。中程调度（交换调度，中级调度）：将进程从内存或从CPU竞争中移出，之后它能被重新调入内存，并从中断处继续执行。这种交换的操作可以调整进程在内存中存在的数量和时机。其主要任务是按照给定的原则和策略，将处于外存交换区中的就绪状态或等待状态的进程调入内存，或把处于内存就绪状态或等待状态的进程交换到外存交换区。主要发生于不使用虚存的系统。短程调度（近程调度，低级调度或微观调度）：使用最多，主要任务是按照某种策略和算法将处理机分配给一个处于就绪状态的进程。两种类型：抢占式：有效提高系统吞吐率和响应效率，三个常见策略：优先权-如果一个高优先级的进程变为就绪状态，操作系统可以中断当前正在执行的低优先级进程，让高优先级进程开始执行，短进程优先-如果一个新进程到达并且其预计执行时间比当前正在执行的进程短，操作系统可能会中断当前进程，转而执行新到达的短进程，时间片-每个进程都被分配一个时间片，即允许进程在CPU上执行的最长时间。当时间片用完时，如果进程还没有完成，它会被放回就绪队列的末尾，等待下一次调度。非抢占式：一个进程只有在自愿放弃CPU或者因为等待某个事件而被阻塞时，操作系统才会将 CPU分配给另一个进程。调度的目标：通过按一定策略分配处理器时间，来优化系统一个或多个方面的行为。面向用户是单个用户或进程感知到的系统行为，面向系统是处理器的使用的效果和频率。与性能直接相关方面是定量的，其他是定性的。面向用户性能相关：周转时间，响应时间，最后期限；面向用户其他：可预测性；面向系统性能相关：吞吐量，处理器利用率；面向系统其他：公平性，强制优先级，平衡资源。典型调度算法：先来先服务（FCFS）：最简单，既可用于作业调度，又可用于进程调度选择最先进入队列的进程，将处理器分配给它效率低，对长作业有力，对短作业无力。短作业优先（SJF/SPN）调度算法：选择（估计）运行时间最短的作业，将它们调入内存运行，平均等待时间、平均周转时间最少，对长作业不利。最短剩余时间（SRT）：抢占+SPN，也称最早截止时间。高响应比优先（HRRN）调度算法：主要用于作业调度，同时考虑每个作业的等待时间和估计的运行时间，每次选择响应比最高的进程投入运行。响应比=（等待时间+要求服务时间）/要求服务时间。优先级调度算法：每次从就绪队列中选择优先级最高的进程，将处理器分配给它，既可用于作业调度，又可用于进程调度。此外还有多优先级队列和动态优先级。时间片轮转调度算法：主要适用于分时系统，所有就绪进程按照FCFS策略排成一个就绪队列，调度程序总是选择队列中第一个进程执行，但又仅能运行一个时间片，被剥夺的进程返回就绪队列的末尾重新排队。特点：时间片的大小对系统的性能影响很大时间片太小，则进程的切换过于频繁，切换的开销增加时间片太大，进程可能得不到及时的响应。多级队列调度算法：设置多个就绪队列，将不同类型或性质的进程固定分配不同的就绪队列，每个队列可实施不同的调度算法。多级反馈队列调度算法：是时间片轮转调度算法和优先级调度算法的综合发展。设置多个就绪队列，为每个队列赋予不同的优先级，赋予各个队列的进程运行的时间片大小各不相同，优先级越高的队列，时间片越小。在队列内进行FCFS。每当一个进程被抢占时，就转入下一级队列等待调度。仅当优先级高队列为空时，才能调度下一级队列。下图中w-停留时间，s-需要的总服务时间，e-执行时间。



周转时间：作业完成时刻-作业到达时刻；带权周转时间：周转时间/服务时间；平均周转时间：作业周转总时间/作业个数；平均带权周转时间：带权周转总时间/作业个数。多核处理器调度算法：对称多处理器（SMP）：主流处理器，每个CPU自己决定自己的调度算法。非对称处理器（ASMP）：一般仅一个处理器能处理系统数据结构，减轻对数据的共享要求。负载平衡问题：每个处理器处理内容尽量平均。亲和性：进程在某个指定的CPU上尽量长时间地运行而不被迁移到其他处理器的倾向性。软亲和性：进程尽量不切换；硬亲和性：进程不切换。

第六章 内存管理

内存管理的需求：需要将内存划分成不同部分进行管理：操作系统和用户程序要分开，不同进程要分开，同一进程不同模块也要分开。同时内存分区也是内存保护必要条件：给不同分区不同保护级别，不同分区有不同访问授权，特定分区有特定限长。需要将进程换入/换出内存：动态重定位。动态重定位与静态重定位：静态重定位发生在程序装入内存时，它会将程序中的逻辑地址一次性转换为物理地址，这个过程在程序开始运行前完成。静态重定位的优点在于它不需要额外的硬件支持，使得程序的静态连接更加容易实现。与静态重定位不同，动态重定位是在程序执行过程中进行的。程序装入内存后，逻辑地址不会立即转换，而是在程序执行时才进行地址转换，所有地址仍然保持逻辑状态。需要特定的硬件支持，如寄存器来存储当前执行程序在内存中的起始地址。动态重定位的优点包括内存空间可以移动，用户进程可以共享内存中同一程序的副本，这对于存储器紧缩和解决碎片问题非常有利。缺点是动态重定位增加了机器成本，并且实现存储管理的软件算法比较复杂。分区方法：固定分区：大小相等/不等。固定分区问题：程序太大无法装载到一个分区中-解决办法部分装载+覆盖。内部碎片：由于分配给进程的内存块大小与进程实际需要的大小不完全匹配，导致内存块内部未被使用的空间。固定分区放置算法：把进程放置在能容纳他的最小分区中。动态分区：分区长度和数量可变。动态分区问题：外碎片：在内存中存在许多不连续的空闲内存块，这些空闲块太小，无法满足某些进程的内存需求，即使所有这些空闲块的总和大于进程所需的内存大小。解决方案：不时移动进程使空闲空间连成一片。动态分区放置算法：最佳适配：选择与要求大小最接近的块；首次适配：选择大小足够的第一个可用块；下次适配：在上次放置位置后选择大小足够的第一个可用块。伙伴系统：可用内存块大小为2*K*，每个大小为2*i*的内存块记录在一个*i*列表中请求大小为s的内存块，确定一个*i*，有2*i-1*≤*s*≤2*i*，采用递归的放置算法，找到内存块，最后合并i列表中伙伴并移入i+1列表。（相当于找满足所需大小的最小2i容量块）重定位：逻辑地址：每个用户进程所看到的独立地址空间叫逻辑地址空间。逻辑地址空间里的地址是逻辑地址。相对地址：相对于当前部分（如程序起始位置）的地址。线性地址：逻辑地址到物理地址变换之间的中间层，一般采用分页系统才会存在线性地址。分页：将内存划分成大小固定、相等且较小的块。将进程划分成同样大小的块。进程的块称为页（page），内存中的块成为页框（frame）通过将进程的各页放置在可用的页框中，一个进程可以占据多个不连续的页框。仅最后一页中存在很小的内碎片，没有外碎片。操作系统需要为每个进程维持一个页表，页表描述了该进程每页对应页框位置。页和页框的大小必须是2的幂。页表的起始地址存放在寄存器。页面大小对分页系统影响：页面大则内碎片大；页面小则页表长。多级页表优点：不需要连续的内存空间存放页表。一级页表劣势：页表必须连续存放，因此当页表很大时，需要占用很多个连续的页框；没有必要让整个页表常驻内存，因为进程在一段时间内可能只需要访问某几个特定的页面。线性地址到物理地址映射：

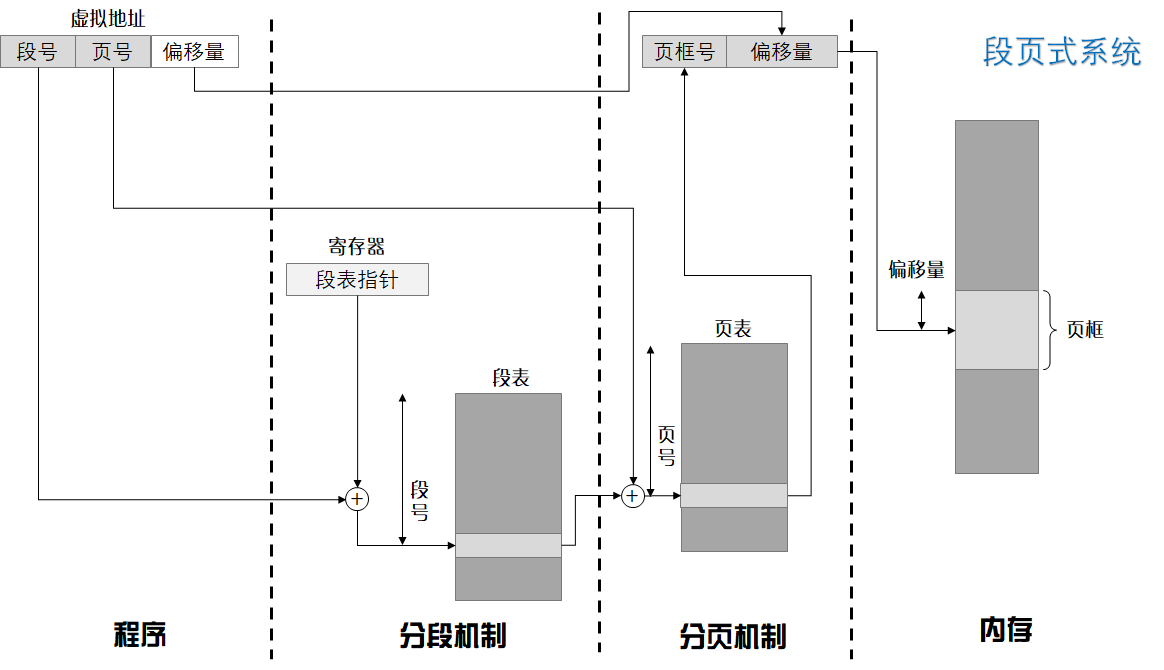
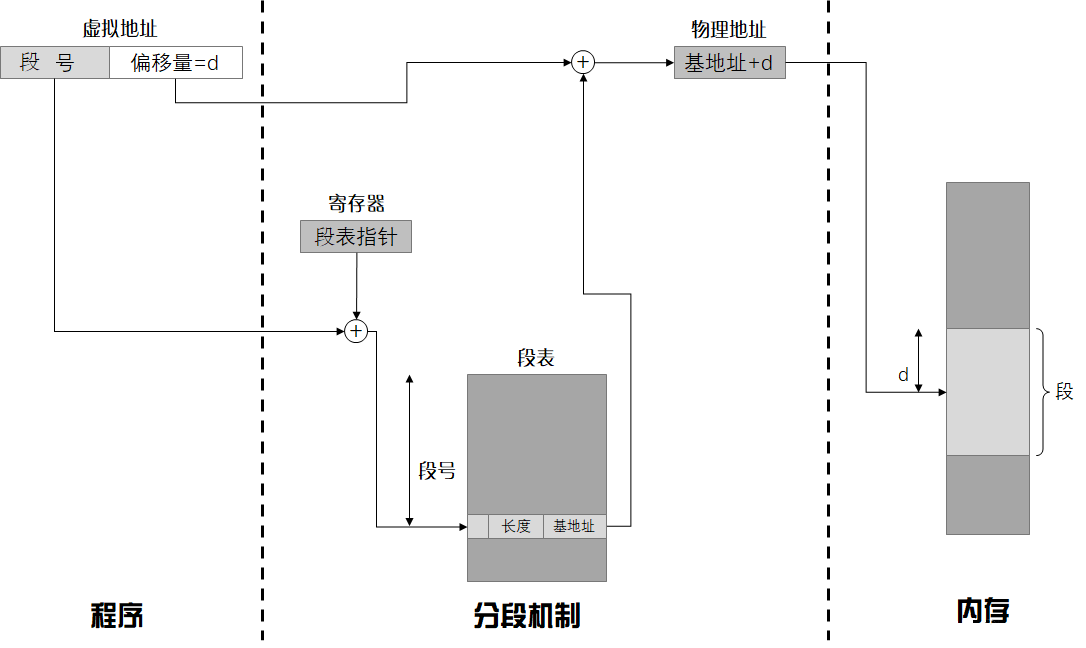
<-段地址与物理地址

分段：类似于动态分区，一个程序可以占据多个段且不要求连续。没有内碎片。由于进程可以被分成多个小块，外碎片也会较小。分页对程序员透明但分段通常可见，是组织程序和数据的一种手段。需要段表。分段的地址由段号和段内偏移构成。

第七章 虚拟内存

局部性原理：时间局部性：某条指令被执行后，不久可能再次被执行。某数据被访问后，不久之后可能再次访问（Cache）;空间局部性：一段时间内所访问的地址集中在一定范围内。发展历史（引入原因）：早期整个程序装入内存执行，内存不够不能运行。后来，内存不足时以进程为单位在内外存之间交换。对内存采用分页/分段管理后：进程中所有内存访问都是逻辑地址，运行时动态转换为物理地址使重定位成为可能，进程被划分成许多块（页/段）不需要连续的位于内存中。虚拟内存：虚拟内存是一种计算机系统内存管理技术，它允许应用程序认为自己拥有一块连续的、足够大的内存空间。实际上，这块“内存”可能是由多个物理内存碎片组成的，部分内容甚至存储在外部磁盘上。当程序运行时，需要的数据会从磁盘交换到物理内存中，这个过程对应用程序来说是透明的。虚拟内存中：在页/段表项中，加入一个标志位，使处理器知悉该页/段是否在内存中。调页时机：请求调页（实际虚拟内存采用）：运行中需要的页面不在内存，便立即提出请求，由OS将其调入内存。优：有请求调页策略所确定调入的页，一定会被访问且比较容易实现。缺：每次仅调入一页，需花费较大的系统开销，增加了磁盘的IO的启动频率。预调页：以预测为基础，将预计不久后便会被访问的若干页面预先调入内存。优：一次调入若干页，效率较好；缺：预测不一定准确，与调入的页面可能根本不被执行到。主要用于进程的首次调入，由程序员指出应该先调入那些页。缺页中断：处理器需要访问一个不在内存中的逻辑地址时，会产生缺页中断。解决过程：如果内存已满，先进行置换。进程被阻塞，操作系统发出磁盘IO请求。当块被载入内存后，修改页表，并通过IO中断，唤醒该进程。在一条指令的执行过程中，可能多次发生缺页中断。驻留集：给一个进程分配的物理页框的集合被称之为驻留集。或者叫进程已装入内存的页面的集合，与系统采用的页面装入和页面置换算法有关。驻留集尺寸：进程驻留在内存中的页面数量。与系统采用的页面分配策略有关。分配给一个进程的物理内存越小，驻留在主存中的进程数就越多，从而可以提高处理器的利用率。如果一个进程分配的物理内存（页框）过少，尽管有局部性原理，页错误率仍然会相对较高如驻留集较大，由于局部性原理，当给特定进程分配更多的主存空间时，对进程的缺页错误率没有明显的影响。内存分配策略：固定分配：为每个进程分配一组固定大小的物理块，不允许改变。可以平均分配，或按程序大小比例分配，或按优先权分配。可变分配：动态地分配内存空间，每个程序可以获得所需的内存大小。按照缺页率动态调整，性能较好，但会增加算法运行的开销。内存置换策略：全局置换：发生缺页时，可以在所有进程的物理块中进行置换。也就是说，置换页面的选择范围是所有可换出的物理页面。优：可以动态调整每个进程的物理页面数目，根据实际需求分配内存资源，提高内存利用率。缺：可能导致进程间的相互干扰，一个进程的内存需求增加可能会影响其他进程的性能。常见全局置换算法：工作集置换算法：根据进程的工作集动态调整物理页面数目；缺页率置换算法：根据缺页率动态调整物理页面数目。局部置换：在发生缺页时，只能在当前进程的物理块中进行置换。也就是说，每个进程有固定数量的物理页面，置换时只能在这些页面中选择。适用于进程间内存需求差异较大的情况，可以避免进程间的相互干扰。优：每个进程的页面数目是固定的，不会因为其他进程的内存需求而受到影响。常见局部置换算法：最优算法OPT：置换未来最长时间不访问的页面；先进先出算法FIFO：置换最早进入内存的页面；最近最久未使用算法LRU：置换最长时间未被访问的页面。虚拟内存从何处调入：系统有足够交换区则全部从交换区调入；交换区不足不会修改的文件直接从文件区调入；对于UNIX系统，按是否运行过分别处理。虚拟内存效果：支持更有效的系统并发度：在内存中保留更多的进程；解除用户与内存之间没必要的紧密约束：进程可以比内存全部空间都大。虚存最大容量由计算机地址机构决定。虚拟内存实现手段：（段）页管理+部分加载+按需调页+换入换出。系统抖动：内存几乎所有空间都被各个进程占据时，当操作系统读取一块时，必须把另一块换出。如果此块正好在将要用时被换出，操作系统不得不很快将其取回。处理器大部分时间用于交换块，而不是执行指令。解决方法：基于局部性原理推断将来最可能用到的块。基于分页的/分页和分段的虚拟内存已经成为当代操作系统的一个基本构件。基于分页的虚拟内存：虚拟内存通常与使用分页的系统联系在一起，每个进程有自己的页表。地址映射：虚拟地址中包括页和偏移。其中页部分与页表部分进行相加，在页表中找到对应页表项，读取页表项包含的物理页帧号（frame number），找到对应的物理页帧，与虚拟地址后面的offset部分进行拼接，最终得到实际物理地址。页表占据很大内存空间解决办法：将部分页表放在虚存中或采用多级页表（根页表-用户页表-用户地址空间）。倒置页表：倒置页表为物理内存中的每个帧（Frame）维护一个条目，而不是为每个虚拟页维护一个条目。每个条目包含三个部分：页号（Page #）：与虚拟地址中的页号相对应。进程ID（Process ID）：标识该页属于哪个进程。链表（Chain）：用于处理哈希冲突，当多个页号经过哈希函数后得到相同的索引时，它们会被链接到同一个链表中。优点：减少对页表存储需求，对具有大量物理内存和少量活跃虚拟页的系统中特别有效。此外，倒置页表可以减少页表的碎片化，并且可以更容易地实现物理内存的共享。缺点：倒置页表的实现复杂度较高，因为它需要处理哈希冲突，并且可能需要更复杂的数据结构来维护链表。转换检测缓冲区/快表（TLB）：在MMU(内存管理单元)中，功能与cache相似，包含有最近用过的页表项。支持虚拟内存的硬件和控制结构：页表项可能在TLB，内存或磁盘中。被访问的数据可能在Cache，内存或磁盘中。页尺寸：页越小，内碎片总量越少，有助于减少缺页中断。页越小，页表越大，可用内存减少，会增加缺页中断的次数，有可能导致两次缺页中断。页越大，越有利于磁盘读写。页越大，代码（数据）越集中，缺页中断有增加的可能。页尺寸接近整个进程大小时，缺页率为零。总结：页尺寸从小到大过程中，缺页中断比率先增大后下降，直到页尺寸等于整个进程大小。

分段系统中的地址映射和段页式系统：



支持虚拟内存的操作系统机制：页面置换算法：见上（局部置换算法），此外还有最近未使用算法NUR：每页设置一个访问位，再将内存中的所有页面都通过链接指针链接成一个循环队列；当某个页面被访问时，其访问位置1。淘汰时，检查其访问位，如果是0，就换出；若为1，则重新将它置0；再按FIFO算法检查下一个页面，到队列中的最后一个页面时，若其访问位仍为1，则再返回到队首再去检查第一个页面。时钟置换算法clock：对于页替换算法，用于替换的候选帧集和看作一个循环缓冲区，并且有一个指针与之相关联。当某一页被替换时，该指针被设置成指向缓冲区的下一帧。当需要替换一页时，操作系统扫描缓冲区，以查找使用位被置为0的一帧。每当遇到一个使用位为1的帧时，操作系统就将该位重新置为0；如果这个过程开始时，缓冲区所有帧使用位均为0，则选择遇到的第一个帧替换；如果所有的使用为均为1，则置身在缓冲区中完整的循环一周，把所有使用位都置为0，并且停留在最初的位置上，替换该帧中的页。PPT中clock描述：将可用页框按顺序组成环形队列，引用位初始置0，并置指针初始位置；缺页时从指针位置扫描引用位，将1变0，直到找到引用位为0的页；当为某页初始分配页框，或页被替换，则指针移到下一页。LRU的实现：解决方法1：维护一个时间戳(time stamp)，即计数器。每次页引用时，计数器加1，并将该值复制到相应页表项中。当需要置换页时，选则计数值最小的页。存在的问题：每次地址访问都需要修改时间戳，需维护一个全局时钟(该时钟溢出怎么办?)，需要找到最小值…这样的实现代价较大。解决方法2：建立一个容量为有效页框数的页码栈。每当引用一个页时，该页号就从栈中上升到栈的顶部，栈底为LRU页。当需要置换页时，直接置换栈底页即可。存在的问题：每次地址访问都需要修改栈，实现代价仍然较大。请求调页系统中的外存分为：存放可执行文件的文件区（原始存放位置）以及存放交换页面的交换区。交换区：基于文件系统（如windows中的pagefile.sys），独立的磁盘分区-生磁盘（RAW），不需要文件系统和目录结构，如Linux中的swap分区。页面置换目的地：系统拥有足够的交换区空间：可以全部从交换区调入所需页面，以提髙调页速度。在进程运行前，需将与该进程有关的文件从文件区复制到交换区。系统缺少足够的交换区空间：凡不会被修改的文件都直接从文件区调入；而当换出这些页面时，由于它们未被修改而不必再将它们换出。对于可能被修改的部分，将它们换出时须放到交换区，需要时再从交换区调入。UNIX方式：未运行过的页面，都应从文件区调入。曾经运行过但又被换出的页面，被放在交换区，下次调入时从交换区调入。影响请求分页系统平均访存时间的是：缺页率，磁盘读写时间，内存访问时间，执行缺页处理程序的CPU时间。

第八章 文件管理

文件：以磁盘为载体的、存储在计算机上的信息集合，大多数应用程序的输入输出都是通过文件实现的通过文件可以长期保存信息，以便将来访问。需要文件管理系统，对文件进行访问、修改和保存。文件的内涵：是存储空间上的数据（域，记录，文件，数据库等），有用于分类和索引的标签，包含关于访问权限的信息。文件的外延：文件系统是操作系统重要组成部分，提供了与二级存储相关的资源的抽象。文件管理系统：一组系统软件，为使用文件的应用程序和用户提供服务。文件系统功能：实现对文件的基本操作（创建，删除，打开，关闭，读，写），按名存储和查找文件，组织成合适的结构，具有基本的文件共享和文件保护功能，管理与磁盘的信息交互，完成文件逻辑结构和物理结构的变换，组织文件在磁盘上存放，采用好的文件排放顺序和磁盘调度方法以提升系统的调度性能。文件系统架构：堆，顺序，索引顺序，索引，目录哈希，逻辑文件系统，文件组织模块，基本文件系统，IO控制。IO控制：设备驱动程序和中断处理程序，在内存和磁盘之间传送信息。基本文件系统：也被称为物理IO层，读取和写入磁盘的物理块，管理内存缓冲区。关注的是这些快宰辅存忠和内存缓冲区中的位置。基本IO管理（文件组织模块）：将逻辑块地址转化成物理块地址，空闲空间管理。逻辑IO（逻辑文件系统）：管理和维护文件系统的所有逻辑结构信息，负责文件保护。文件管理功能：从用户和程序命令开始，通过目录管理和文件结构，用户可以指定操作和文件名。文件操作函数根据访问方法对文件进行操作。操作系统通过文件分配和自由存储管理来处理文件的存储。I/O操作将文件数据在内存缓冲区和磁盘之间传输。磁盘调度优化磁盘I/O操作，以提高效率。关注文件管理两个层面：用户层面-关注文件的组织和访问，以及操作系统层面-关注文件的物理存储和IO操作。堆：按数据到达顺序收集，每条记录由一串数据组成。记录可以有不同的域，每个域应该有自我描述。顺序文件：每条记录有固定的格式，相同的长度，包含相同数量和长度固定的域。关键域按顺序排列。设置一个Log File，把试图增加，删除或者修改的信息记录于其中。索引顺序文件：用索引提供了快速接近目标记录的查找能力。增加了溢出文件，用来记录新增，删除和修改的记录。索引文件：对于可变长记录文件，建立表项大小相等的索引表，可以多索引，可以部分索引。文件控制块（FCB）：存放控制文件需要的各种信息的数据结构。包括：基本信息：文件名、文件的物理位置、文件的逻辑结构、文件的物理结构等；访问控制信息：文件的存取权限、所有者、授权用户的存取权限、一般用户的存取权限等；使用信息：文件建立时间、上次修改时间等。目录：FCB的有序集合，FCB就是集合中的一个文件目录项。文件目录也被视为文件---目录文件。目录的树状结构：每个子目录都在上级目录文件中有一个目录项（FCB），每个子目录都是一个目录文件。文件组织和访问：当文件很多时，文件目录会占用大量的盘块。在检索目录的过程中，只用到了文件名。其它描述信息不会被用到，也不需要调入内存。有的操作系统中，将除文件名之外的其它描述信息单独形成一个数据结构，称为索引结点，简称i结点（inode）。目录文件中每个目录项仅由文件名和指向i结点的指针构成。每个文件在磁盘中有唯一的索引结点。当文件被打开时，要将磁盘索引结点复制到内存的索引结点中。且有编号、状态、访问计数等信息。打开文件表：对于用户进程：包括编号（Unix的文件描述符，windows的文件句柄），文件名，读写指针，访问权限和系统表索引号；对于系统的打开文件表（整个系统只有一张）：包括编号，文件名，外存地址和打开计数器（在多少个程序中打开）等。文件系统在外存中结构：整个磁盘包括主引导区，分区表，分区1（主分区），分区2，分区3......，按照unix文件系统布局，主分区中包括引导区，超级数据块，空闲区管理，I-节点区，根目录区以及文件和目录区，这些被称作逻辑分区。MBR：位于磁盘的0号扇区，记录着硬盘本身的相关信息以及硬盘各个分区的大小及位置信息。引导块：引导块中的程序负责启动该分区中的操作系统。超级块：包含文件系统中的所有关键信息。启动时会被读入内存，是文件系统的第一个块，存放文件系统本身的结构信息描述和文件系统整体信息的数据结构。空闲块信息：用位示图或指针链接的形式给出。文件共享：在多用户系统中总是允许文件在多个用户间共享，文件系统可以授予用户/用户组某些权限：知道，执行，读，追加，更新，改变保护，删除。通常所有者是创建者，所有者具有所有权限，并可以给其它用户授权。文件共享机制：基于索引节点的文件共享（打开文件表），利用符号链接实现的文件共享。在设计文件共享时，必须解决互斥和死锁问题。逻辑组块：在外存管理中，为了方便对文件数据的管理，文件的逻辑地址空间也被分为了一个一个的文件“块”。三种组块方式：定长组块，变长跨越式组块，变长非跨越式组块。定长组块是记录定长顺序文件的常用方式，变长跨越式组块存储效率高，对文件大小没有限制，但实现难度大。变长非跨越式组块会浪费空间。文件系统两大问题：文件分配：如何为文件分配磁盘块；辅存空闲空间管理：空闲块的组织，分配和回收。连续文件分配：要求每个文件在磁盘上占用一组连续的块，顺序读/写时速度快，可直接访问，文件长度不宜动态增加，增删记录不但需要移动其它盘块，还需改变文件长度，有外碎片问题。隐式链接：离散分配，无外碎片；除最后一个盘块外，每个盘块末尾都含有指向文件下一个盘块的指针，只适合顺序访问，稳定性差。显式链接：指将用于链接各物理块的指针显示地存放在一张表中。即文件分配表（FAT，File Allocation Table）。整个磁盘仅一张FAT，FAT还可以标注空闲的磁盘块，操作系统启动时会将FAT读入内存，从逻辑块号转换为物理块号不需要读磁盘，FAT会占用较大的磁盘空间。索引分配：系统会为每个文件建立一张索引表，索引表中记录了文件的各个逻辑块对应的物理块。索引表放在索引块中，每个文件必须有一个索引块。索引块太小：采用链接方案，多层索引，混合索引。混合索引分配：UNIX系统采用的文件分配方式。混合索引指将多种分配方式与索引分配相结合的分配方式。如既采用连续分配的方式，又采用索引分配的方式，即一个文件前面的数据块采用连续分配，后面的数据块采用索引分配。空闲盘块管理：方法1：空闲位图：空闲置1，占用置0。可快速分配连续盘块组，但位相量很大；方法2：空闲链表：依次指向下一个空闲位置。分配一个/少量的空闲盘块是可以高效工作，但分配多个则慢。MINIX文件系统：一种基于微内核架构的类UNIX计算机操作系统，于1987年由Andrew S. Tanenbaum教授发布，它启发了Linux。硬盘设备上的分区及文件系统：前到后：主引导扇区，FAT32系统，NTFS文件系统，MINIX文件系统，EXT2文件系统。MINIX文件系统1.0逻辑结构布局（名称+大小）：引导块-1，超级块-1，i节点位图-1，逻辑块位图-1，i节点-i，数据区-一个盘块两个扇区。引导块：计算机加电启动时，由ROM BOIS程序自动读入MBR，MBR找到引导块并读入引导代码和数据。对于不是引导分区，该盘块空闲不用。超级块：保存分区中文件系统的结构信息。超级块表示出每部分的数量和位置：



i节点位图：标记i节点使用情况，每bit位表示1个i节点，即1个目录文件或普通文件。可以占用多个盘块。逻辑块位图：标记磁盘分区中每个数据盘块的使用情况，每bit位代表1个盘块，只标记数据区的盘块。i节点：存放着文件系统中目录文件或普通文件的i节点，每个目录文件或普通文件都有1个i节点结构，每个i节点结构中存放着对应文件的相关信息，第1个i节点为根目录文件的i节点结构。i节点数据结构见上（32bytes，盘块号需要2B）。如何访问文件/usr/bin/文件名3（编辑工具）：根目录文件的i节点位置是固定的，即第1号i节点。1号i节点的数据块内容为根目录下的目录项列表，通过该目录项列表匹配目录名“usr”。若找到，则可得到文件“/usr”的i节点号i1，根据i1号i节点的数据块，可以取得目录文件“/usr”的内容，即子目录usr的文件目录项列表。通过该目录项列表匹配目录名“bin”，若找到，则可得到文件“/usr/bin”的i节点号i2，根据i2号i节点的数据块，可以取得目录文件“/usr/bin”的内容，即子目录bin的文件目录项列表。通过该目录项列表匹配文件名“文件名3”，若找到，则可得到文件“/usr/bin/文件名3”的i节点号i3，根据i3号i节点的数据块，可以取得文件“文件名3”的内容。如何删除文件/usr/bin/vi：根目录文件的i节点位置是固定的，即第1号i节点。1号i节点的数据块内容为根目录下的目录项列表，通过该目录项列表匹配目录名“usr”；；若找到，则可得到文件“/usr”的i节点号i1。根据i1号i节点的数据块，可以取得目录文件“/usr”的内容，即子目录usr的文件目录项列表，通过该目录项列表匹配目录名“bin”。若找到，则可得到文件“/usr/bin”的i节点号i2，根据i2号i节点的数据块，可以取得目录文件“/usr/bin”的内容，即子目录bin的文件目录项列表。通过该目录项列表匹配文件名“vi”，若找到，则可得到文件“/usr/bin/vi”的i节点号i3，根据文件长度从i3节点中找到每个盘块的索引号，将i节点位图和逻辑块位图中相应位置0（释放）。删除目录文件bin中对应目录项。MINIX文件系统1.0m\_node数据结构见上。

第九章 IO设备管理

IO设备分类：按信息交换的主体分：人可读设备：适用于与人的交互，如：打印机，键盘，显示器；机器可读设备：适用于设备间的数据交互如：磁盘，各控制器；通信设备：适用于远程设备通信，如：网卡。按照信息交换的单位分：块设备：信息交换以数据块为单位，有结构设备，可寻址；字符设备：信息交换以字符为单位，无结构，不可寻址。按传输速率分：低俗设备：低于每秒千字节如键盘，鼠标；中速设备：每秒千字节-万字节，如打印机；高速设备：每秒百千字节到数千兆字节，如磁盘。IO控制方式变更：程序直接控制方式-增加设备控制器（I/O接口），处理器使用非中断的程序控制IO，但从IO设备的具体细节中分离处理。采用设备控制器，采用中断方式--设备控制器通过DMA直接控制存储器—IO通道：IO模块有一个单独的处理器，有专门为IO指令设计的指令集，CPU指导IO处理器执行内存中的一个IO程序，完成一系列的IO活动—IO处理机：IO模块有自己的局部存储器，本身就是一台计算机。IO逻辑结构：用户进程-逻辑IO-设备IO-调度与控制。逻辑IO：把设备当作一个逻辑资源来处理，提供统一的接口，允许用户进程根据设备标识符，进行打开、关闭、读、写等操作。设备IO：请求的操作和数据被转换成适当的I/O指令序列、通道命令。调度和控制：I/O操作的排队、调度，处理中断，收集I/O状态，是与I/O模块和设备硬件真正发生交互的软件层。还有其他的设备无关层：通信架构，目录管理-通过文件描述符表或索引表直接或简介访问文件，文件系统-文件的逻辑结构，处理用户指定的操作，物理组织-将逻辑访问转化成物理外存地址。IO缓冲：提高CPU与IO设备的并发性，缓和CPU与IO设备间速度不匹配的矛盾，解决数据粒度不匹配的问题。单缓冲：在内存中开辟一个固定大小的区域作为缓冲区。外设和CPU交换数据时，先将被交换的数据写入缓冲区，然后再由需要数据的CPU或外设从缓冲区中取出。该方式中，外设与CPU对缓冲区的操作是串行的。双缓冲：内存中设置两个大小相同的缓冲区，外设和CPU可以交替使用这两个缓冲区，从而在一定程度上实现并行交换数据。循环缓冲：在内存中设置大小相等的多个缓冲区，并将它们连接成为一个环形链表。缓冲队列：队列中每一项包括：缓冲区号，设备号，设备上的数据块号，互斥标识位，队列指针，数据区。缓冲池工作流程：当输入设备需要进行数据输入时，则从空缓冲队列的队首取下一个空缓冲区，将它作为收容输入工作缓冲区，当它被输入装满数据后，则被链接到输入缓冲队列的队尾；当某进程需要从缓冲池读入数据时，则从输入缓冲队列的队首取一个缓冲区作为提取输入工作缓冲区，该进程从中提取数据，取完后，则将该缓冲区链接到空缓冲区队列的队尾；当某进程需要输出数据到缓冲池时，则从空缓冲队列的队首取下一个空缓冲区，将它作为收容输出工作缓冲区，该进程向该缓冲区中存放数据，当它被装满数据后，则被链接到输出缓冲队列的队尾；当输出设备需要进行数据输出时，则从输出缓冲队列的队首取一个缓冲区作为提取输出工作缓冲区，并从中提取数据输出，取完后，则将该缓冲区链接到空缓冲区队列的队尾。磁盘调度：一次磁盘的读写操作时间：寻道时间+旋转延迟+传输时间，其中寻道时间Ts=跨越一条磁道所需时间m\*n+启动磁臂时间s，旋转延迟时间Tr=1/2r，r为转速。，传输时间Tt=要读写的字节数b/（r\*磁道上的字节数N）。总平均存取时间Ta=Ts+1/2r+b/rN。磁盘调度算法：先进先出（FIFO）算法磁盘调度算法：进程少时有较好性能，进程多时接近随机调度；最短服务时间优先（SSTF）算法：选择从当前位置开始移动最少的磁盘IO请求，存在饥饿问题。电梯（SCAN）算法：磁头臂仅沿一个方向移动，直到到达该方向最后一个磁道，或该方向没有其他请求为止（LOOK策略），接着反转服务方向。对最近跨越过的区域不公平，不能很好利用局部性，偏向对靠里或靠外磁道进行请求的作业。循环电梯算法：把扫描限制在一个方向上，扫描方向没有其他请求时，返回相反方向末端。举例：请求磁道55，58，39，18，90，160，150，38，184，从磁道100开始：FIFO：55-58-39-18-90-160-150-38-184；SSTF:90-58-55-39-38-18-150-160-184;SCAN:150-160-184-90-58-55-39-38-18;C-SCAN:150-160-184-18-38-39-55-58-90。磁盘高速缓存：提高磁盘IO速度，利用内存中的存储空间，来暂存从磁盘中读出的一系列盘块中的信息。逻辑上属于磁盘，物理上驻留在内存中。磁盘高速缓存两种形式：在内存中开辟一个单独的空间作为磁盘高速缓存，大小固定。或把未利用的内存空间作为一个缓冲池，供请求分页系统和磁盘IO共享。