操作系统课后题整合

第一章

1.1 操作系统的三个主要目的是什么?

管理计算机硬件

为应用程序提供基础

充当计算机用户和计算机硬件之间的中介程序

1.5 内核模式和用户模式作为一种初级的保护（安全）系统，其区别是什么？

只有当CPU处于内核模式时，才能执行某些指令（特权指令）。

如果尝试在用户模式下执行特权指令，则不会执行并视为非法指令。硬件将其陷阱到操作系统。

1.6 下面哪些指令应该被赋予特权？

设置定时器的值、清楚内存、关闭中断、修改设备状态表中的条目、从用户模式切换到内核模式、访问IO设备

1.12 在多道程序和分时环境中，几个用户同时共享系统，会导致安全问题

1. 这两个问题是什么？

2. 我们能否确保分时机器与专用机器具有相同的安全性？

窃取或复制用户的文件、写入内存中属于另一个用户或操作系统的区域

不能，因为人类设计的任何分时环境保护方案都可能被破坏

1.15 描述对称和非对称处理器之间的区别。多处理器系统的三个优点和一个缺点是什么？

对称处理将所有处理器平等处理，任何一个都可以处理IO。进程可以在其中任何一个CPU上安排。非对称处理指定一个CPU作为主CPU，能够唯一执行IO，主CPU将在其他CPU之间分配计算工作。

多处理器系统可以通过共享电源和外设来省钱

可以更快地执行程序（并行）

可以提高可靠性

多处理器系统在硬件和软件上都更加复杂，需要额外的CPU周期来管理协作，因此每个CPU效率会下降

1.19 中断的目的是什么？中断与陷阱有何不同？陷阱可以由用户程序故意产生吗？如果可以，是为了什么目的？

中断是系统中由硬件产生的流变化，可以用来发出IO完成的信号，以消除对设备的轮询。调用中断可以处理程序中造成中断的原因，控制返回到中断的上下文和指令处。

陷阱是软件产生的中断，可以用来调用操作系统进程或捕获错误。

1.22 许多SMP系统有不同级别的缓存;一个级别是每个处理核心的本地级别，另一个级别是所有处理核心之间共享的。为什么缓存系统是这样设计的?

不同级别是基于访问速度和大小的。一般来说，缓存越接近CPU，访问速度就越快，但是价格也会更昂贵。因此更小、更快的缓存被放在CPU的本地，而更大但较慢的共享缓存则在几个不同的处理器之间共享。

1.23 用事例说明内存中的数据在本地缓存中具有不同的值

处理器1从主内存读取A的值为5，保存到本地缓存中。相似的，处理器2从主内存中也读取A。然后处理器将A更改为10，然而，因为A在处理器1的本地缓存中，更新只在这里而不在处理器2的本地缓存。

1.27 描述为移动设备设计操作系统与为传统PC设计操作系统的一些挑战。

低功耗

易于使用，用户友好，学习简单

适于触摸语音等交互方式

反应快，待机和使用状态转变块

适用长期驻于后台的多功能设计

安全性高

第二章

2.1 系统调用的目的是什么？

为用户或应用程序提供了一种调用操作系统服务的方法

2.5 命令解释器的目的是什么

读取命令（从用户或者命令文件）并执行（转化成系统调用）

为什么它经常与内核分开？

因为它可能会经常改变

2.6 为了启动一个新的进程，命令解释器或外壳必须执行哪些系统调用？

Fork系统调用后exec系统调用来启动一个新进程

2.7 系统程序的目的是什么？

为程序开发和运行提供了方便的环境

给用户提供基本的公共功能函数，为用户不在自己写代码的情况下解决公用问题

2.8 系统设计的分层方法的主要优点是什么？

开发人员可以只关注整个结构中的某一层

可以很容易的用新的实现来替换原有的层次实现

可以降低层与层之间的依赖

有利于标准化

有利于各层逻辑的复用

缺点是什么？

降低了系统的性能，必须通过中间层完成，不允许跨层

有时需要级联的修改，尤其是自上而下的方向。

2.9 列出操作系统提供的五种服务

执行程序：操作系统加载文件的目录到内存中并开始执行。在用户程序不能合理分配CPU时间的情况下不能提供该项服务。

IO操作：程序运行过程中需要IO设备上的数据时通过IO命令或IO指令请求操作系统服务

文件系统操作：文件系统让用户按文件名创建，读写，修改，删除文件，使用方便，安全可靠。当设计多用户访问或共享文件时，操作系统还提供信息保护机制

通信服务：很多情况下，一个进程要与另外的进程交换信息，进程通信可以借助共享内存实现，也可以使用消息传送技术实现

错误检测：操作系统可以不做和处理各种硬件和软件造成的差错和异常

2.12 操作系统提供的服务和功能可以分为两大类。简要描述两个的区别

操作系统提供一类服务是在系统中不同并发运行的进程之间强制保护。进程只允许访问与它们关联的内存位置。此外，进程不允许损坏与其他进程相关联的文件。进程也不允许在没有操作系统介入的情况下直接访问设备。

操作系统提供的第二类服务是提供低层硬件不能直接支持的功能。虚拟内存和文件系统就是操作系统提供的两类服务的例子。

2.13 描述三种向操作系统传递参数的一般方法

寄存器传递参数

内存的块或表存放参数，块或表的地址通过寄存器传递

参数通过程序放在或压入到堆栈，并通过操作系统弹出

2.16 使用同一个系统调用接口来操作文件和设备的优点和缺点是什么?

可以像访问文件系统中的文件一样访问每个设备。因为大多数内核都是通过文件接口来处理设备的，所以相对容易通过实现具体硬件的代码来支持抽象的文件接口来增加一个新的设备驱动器。因此这有利于双方用户代码的发展，它可以以同样的方式来访问设备和文件，并且设备驱动代码可以支持一个定义明确的API。

使用相同的接口的弊端在于它可能很难捕捉在文件访问API上下文中某些设备的功能，进而导致功能损失或性能损失。

2.17 用户是否可以使用操作系统提供的系统调用接口开发一个新的命令解释器？

用户应该能够使用系统提供的系统调用开发新的命令解释器。命令解释器允许用户这样创建和管理进程并且确定它们的通信方式(通过管道和文件)。由于所有这些功能都可以被用户级的程序进行系统调用，因此用户可以开发出一个新的命令行解释器。

2.19 为什么机制和政策分离的政策是可取的？

策略决定做什么，机制决定怎么做，因此机制和策略必须分开，以确保系统易于修改。

没有两个系统的安装是相同的，所以每个安装都可能想调整操作系统以适应其需求。在机制和策略分开的情况下，策略可以随意改变，而机制则保持不变。这种安排提供了一个更灵活的系统。

2.21 **微内核的优点**？

添加新服务器不需要修改内核

由于在用户模式下进行的操作比在内核模式下进行的操作更多，因此更安全

更易拓展操作系统，更容易将操作系统移植到新的平台，更安全可靠。

如何交互？

用户程序和系统服务在微内核框架中通常使用进程间通信机制，例如消息传递来交互。这些消息由操作系统传递。

缺点？

由于框架中的交互频繁使用进程间通信机制，因此用户空间与内核时间开销比较大。

第三章

1. 输出是什么？

A在父进程中仍然为5，当子进程修改其内存中的值时，这些修改不会影响父进程的相应内存位置中的值。子进程和父进程之间的内存是相互独立的，因此它们的变量和数据是隔离的。这是由于操作系统在fork()调用后对父子进程的内存进行了分离和拷贝。

3.5 当进程使用 fork（） 操作创建新进程时，父进程和子进程之间共享以下哪种状态？

共享共享内存，栈和堆均不共享

3.8 长期调度、中期调度、短期调度的区别？

短期（CPU调度器）：从内存中准备执行的进程中选择一个进程，并将CPU分配给它。

中期（内存管理器）：从准备好的或阻塞的队列中选择一个进程，把它们从内存中换到磁盘上，然后再把它们换进去继续运行。

长期（作业调度器）：决定哪些作业从磁盘上的回存带入内存进行处理。

一个主要的区别在于其执行的频率。

3.12 包括最初的进程，一共创造了多少个进程？

16个

3.13 exec函数

当子进程创建成功后，调用execlp()加载/bin/ls程序的代码和静态数据，若调用成功，则永远不会执行printf(“line J”)；若调用失败（如不存在/bin/ls这样的可执行程序），才会执行printf(“line J”)

3.14

Fork在子进程中为0，父进程中为子进程的pid

3.17

子程序是父程序的副本，子程序数的更改不会影响父程序

第四章

4.2 用户线程和内核线程的区别是什么？在哪种情况下某种类型更好？

用户级线程是操作系统内核所不知道的线程，它们完全存在于一个进程中，并被安排在该进程的时间片中运行（由线程库本身管理）。用户级线程的切换速度更快，因为没有上下文切换。用户级线程不能在不同CPU上并行运行。

操作系统内核知道内核级线程。内核线程由操作系统的调度算法来安排，每个线程都可以被授予自己的时间片。内核级线程可以在不同的CPU上并行运行。

针对内核是单线程的情况，内核线程优于用户线程。因为任何执行阻塞系统调用的用户线程都会导致整个进程阻塞，即使应用程序中可以运行其他线程。

针对多处理器环境，内核线程优于用户线程，因为内核线程可以在不同的处理器上同时运行，而即使有多个处理器可用，进程的用户线程也只能在一个处理器上运行。

对于时间共享型的内核，用户线程优于内核线程，因为共享系统上下文切换经常发生。内核线程之间的上下文切换具有很高的开销，几乎与进程相同，而与内核线程相比，用户线程之间的上下文切换几乎

4.4 创建线程时使用哪些资源？它们与创建流程时使用的流程有何不同？

创建一个用户或内核线程需要分配一个小的数据结构TCB，即线程控制块，用来保存一个寄存器集、堆栈、优先级和状态。因为线程比进程小，所以创建线程通常比创建进程使用更少的资源，并且效率更高

创建一个进程需要分配一个进程控制块，一个相当大的数据结构。PCB中包括进程编码、进程状态、程序计数器、寄存器、内存上下界等等，是非常耗时的活动

4.7 在什么情况下，使用多个内核线程的多线程解决方案比单处理器系统上的单线程解决方案提供更好的性能？

执行一些高并发式的任务。当内核线程发生页面错误时，可以切换到另一个内核线程，以一种有用的方式使用交错时间。另一方面，当发生页面错误时，单线程进程将无法执行有用的工作。因此，在程序可能出现频繁的页面错误或必须等待其他系统事件的情况下，多线程解决方案甚至在单处理器系统上也能执行得更好。

4.8 在多线程进程中，程序状态的下列哪些组件是跨线程共享的?a寄存器值 b堆内存 c全局变量 d栈内存

每个线程单独拥有 寄存器、堆栈、程序计数器；而数据（即全局变量）、通用代码、特定数据结构（打开文件列表）等都是共享的

因此堆内存和全局变量都是共享的

4.9 使用多个用户级线程的多线程解决方案能否在多处理器系统上比在单处理器系统上获得更好的性能?

不能。一个由多个用户级线程组成的多线程系统不能同时利用多处理器系统中的不同处理器。操作系统只能看到一个单一的进程，不会将进程的不同线程安排在不同的处理器上，因此不会获得更好的性能。

第五章

5.4 为什么自旋锁不适用于单处理器系统，但是经常在多处理器系统中使用

在单处理器系统中，使用自旋锁会造成其他进程忙等待（任何其他试图进入临界区的进程都必须在其进入代码中连续地循环），这样就浪费了CPU时钟。而对于多处理器系统，当一个进程在一个处理器自旋时，另一个进程可以在另一处理器上执行，且使用自旋锁，进程在等待锁时还在运行，不用进行上下文切换

5.10 如果同步原语要用于用户级程序，请解释为什么在单处理器系统中通过禁用中断来实现同步原语是不合适的

如果一个用户级程序被赋予禁用中断的能力，那么它可以禁用定时器中断，防止上下文切换，从而允许它使用处理器而不让其他进程执行。

5.20

更改number时可能产生竞争

更改值前后要加锁

第六章

6.2 抢占式调度和非抢占式调度的区别

抢占式调度允许一个进程在其执行过程中被打断，带走CPU并将其分配给另一个进程。非抢占式调度确保一个进程只有在完成其当前的CPU突发时才放弃对CPU的控制。

6.10 为什么对调度员来说，区分绑定I/O的程序和绑定CPU的程序很重要

I/O绑定的程序具有在执行I/O之前只进行少量计算的特性。这样的程序通常不会用尽其整个CPU quantum。

而另一方面，受CPU约束的程序在不执行任何阻塞性I/O操作的情况下使用其整个CPU quantum。

因此，人们可以通过给绑定I/O的程序以更高的优先级，让它们在绑定CPU的程序之前执行，从而更好地利用计算机的资源（CPU和I/O设备）。

6.19 导致饥饿的算法

SJF、优先级调度

第七章

7.10 有可能出现只涉及一个单线程进程的死锁吗

有可能，比如自己请求了自己还未释放的资源

7.17 考虑一个由四个相同类型的资源组成的系统，由三个进程共享，每个进程最多需要两个资源。证明该系统是无死锁的。

假设该系统是死锁的。这意味着每个进程持有一个资源，并在等待另一个资源。由于有三个进程和四个资源，一个进程必须能够获得两个资源。这个进程不需要更多的资源，因此，它将在完成后归还其资源。

第八章

8.1 逻辑地址和物理地址的区别

逻辑地址和物理地址的基本区别是，逻辑地址是由CPU从运行程序的角度产生的。另一方面，物理地址是存在于内存单元中的一个位置，可以被物理访问。

由CPU为一个程序生成的所有逻辑地址的集合被称为逻辑地址空间。然而，所有物理地址映射到相应的逻辑地址的集合被称为物理地址空间。相同的逻辑地址和物理地址是由编译时和加载时的地址绑定方法产生的。由运行时地址绑定方法（内存管理单元）生成的逻辑地址和物理地址彼此不同

8.4 考虑一个由64个页面组成的逻辑地址空间，每个页面有1,024个字，映射到32个帧的物理存储器上。

逻辑地址位数：16

物理地址位数：15

8.5 允许页表中的两个条目指向内存中的同一个页框有什么效果？

节省内存空间

解释一下如何利用这种效果来减少将大量内存从一个地方复制到另一个地方所需的时间。

这样就不需要花费时间复制内存了

更新一个页面上的某些字节对另一个页面有什么影响？

相同的变化

8.9 内部碎片和外部碎片的区别

内部碎片是填不满页表产生的碎片，目前没有很好地办法解决；外部碎片是内存分配中产生的碎片，目前可以用分页方案完美解决。

8.13 比较连续内存分配、纯分段和纯分页的内存组织方案。

1.外部碎片：连续分配，有；纯分段，有；纯分页，无

2.内部碎片：连续，无；纯分段，无；纯分页，有

3.跨进程共享代码：连续，不行；纯分段，可；纯分页，可

8.14 在一个有分页的系统中，一个进程不能访问它不拥有的内存，为什么？

保护进程，防止错误

操作系统能允许访问其他内存吗？

能

为什么？

操作系统要管理内存

8.20 假设页面大小为1KB，以下地址参考的页码和偏移量是多少（以十进制数字提供）。

a. 3085 3,13

b. 42095 41, 111

c.215201 210, 161

d.650000 634, 784

e.2000001 1953, 129

8.21 BTV操作系统有一个21位的虚拟地址，然而在某些嵌入式设备上，它只有一个16位的物理地址。它也有一个2KB的页面大小。以下每个条目有多少个？

a. 一个传统的、单层的页表

2^10

b. 一个倒置的页表

2^5

8.23 考虑一个具有4KB页大小的256页的逻辑地址空间，映射到64帧的物理内存上。

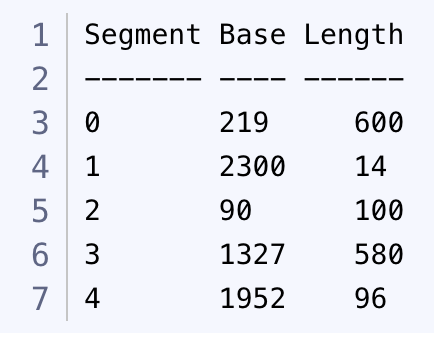
a. How many bits are required in the logical address?

12+8=20

b. How many bits are required in the physical address?

12+6=18

8.28



a. 0, 430: 219+430 = 649

b. 1,10: 2300 + 10 = 2310

c. 2, 500: X

d. 3, 400: 1327+400 = 1727

e. 4, 112: X

8.29 分页页表的目的是什么

由于大多数现代计算机系统支持大的逻辑地址空间（32/64位），页表本身变得过分庞大（空间成本）。我们知道的事实是，虚拟地址空间的整个部分经常是未使用的。分页表没有这些空间的条目，大大减少了存储虚拟内存数据结构所需的内存量。

第九章

9.1 在什么情况下会发生页面故障？描述一下当发生页面故障时，操作系统采取的行动。

发生缺页时。缺页中断，操作系统通过预定的磁盘操作，在磁盘中查询所需页，找到之后将页面交换到物理内存的页中，更新页表和帧表，重新执行被中断的指令

9.4 考虑以下的页面替换算法。根据这些算法的页面错误率，在从 "糟糕 "到 "完美 "的五级评分中对其进行排名。把那些受到贝拉迪异常现象影响的算法和那些没有受到影响的算法分开。

LRU、最佳置换、二次机会页面置换都没有贝拉迪错误；FIFO有

9.5 讨论支持请求调页所需的硬件支持

对于每个内存访问操作，都需要查询页表，以检查相应的页是否驻留在内存中，并确定是否触发了页故障。这些检查必须在硬件（MMU）中进行。一个TLB可以作为一个缓存，并提高查询操作的性能。

9.7 其中A[0][0]在一个页大小为200的分页内存系统中位于位置200。一个操作矩阵的小进程驻扎在第0页（位置0到199）。因此，每条指令的获取都来自于第0页。对于三个页框，下面的数组初始化循环会产生多少个页面错误？使用LRU替换，并假设第1页包含进程，其他两个最初是空的。

在这种情况下，数组A被逐行访问，因此每行产生2个页面故障，因为对一个页面的第一次引用总是产生一个页面故障。使用LRU，它将产生200个页面故障。

在这种情况下，数组A被逐列访问，因此进程在每个外部循环（I）中引用了100个页面，这就是程序的工作集。但是我们只有2个框架，因此每个数组引用将产生一个页面故障。使用LRU，它将产生100\*100=10,000个页面故障。

9.8 假设有一个、两个、三个、四个、五个、六个和七个 frames，下列替换算法会发生多少个页面故障？记住，所有的 frames 最初都是空的，所以你的第一个独特的页面将花费每个故障。

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6

Number of frames LRU FIFO Optimal

1 20 20 20

2 18 18 15

3 15 16 11

4 10 14 8

5 8 10 7

6 7 10 7

7 7 7 7

9.18 某台计算机为其用户提供了一个2^32 字节的虚拟内存空间。该计算机有2^22字节的物理内存。虚拟内存是通过分页实现的，分页大小为4,096字节。一个用户进程产生了虚拟地址11123456。解释一下系统如何建立相应的物理位置。区分软件和硬件的操作。

2^12页面大小；所以2^20页表大小。前面为页表偏移；后面为页面偏移，因此11123位页表号；找到11123位置对应的实际地址；456代表页面偏移，即实际物理地址中对应的偏移量

9.21 假设有三个框架的需求分页，对于下面的替换算法，会出现多少个页面故障？

7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0 , 1.

FIFO：17

LRU：18

OPR：13

第十章

10.9 除了FCFS之外，没有一个磁盘调度规则是真正公平的（可能会出现饥饿）。

a.解释为什么这个论断是真的

它们在优化磁盘访问性能的同时，可能会导致某些请求优先得到服务而延迟其他请求的情况发生

b. 描述一种修改SCAN等算法以确保公平的方法。

所有超过预定年龄的请求可以被 "强制 "到队列的顶部，每个请求的相关位可以被设置，以表明没有新的请求可以被移到这些请求之前。对于SSTF来说，队列的其他部分将不得不根据这些 "老 "请求中的最后一个进行重新组织。

c. 解释为什么公平性是分时系统的一个重要目标。

保证所有的响应得到应答，不会出现长时间等待

d. 举出三个或更多的例子，说明操作系统在为I/O请求提供服务时的不公平是很重要的

分页和交换应该优先于用户请求。

其他由内核发起的I/O，例如文件系统元数据的写入，可能需要优先于用户I/O。

如果内核支持实时进程的优先级，这些进程的I/O请求应该被优先考虑。

10.11 假设一个磁盘驱动器有5,000个柱区，编号为0到4,999。该硬盘目前正在为2,150缸的请求提供服务，前一个请求是在1,805缸。待处理的请求队列，按照先进先出的顺序，是

2,069, 1,212, 2,296, 2,800, 544, 1,618, 356, 1,523, 4,965, 3681

a. FCFS

The FCFS schedule is 2150, 2069, 1212, 2296, 2800, 544, 1,618, 356, 1,523, 4,965, 3681.

b. SSTF

The SSTF schedule is 2150，2069，2296, 2800, 3681, 4965, 1618, 1523, 1212, 544, 356.

c. SCAN

The SCAN schedule is 2150，2296,2800,3681,4965,4999，2069,1618,1523,1212,544,356.

d. LOOK

The LOOK schedule is 2150，2296,2800,3681,4965,2069,1618,1523,1212,544,356.

e. C-SCAN

The C-SCAN schedule is 2150,2296,2800,3681,4965,4999,0,356,544,1212,1523,1618,2069.

第十一章

11.2 为什么有些系统会记录文件的类型，而其他系统则让用户自己来记录，还有一些系统干脆不实现多文件类型？哪个系统 “更好”？

一些系统允许根据文件的类型进行不同的文件操作（例如，一个ascii文件可以作为一个流来读，而一个数据库文件可以通过一个索引来读一个块）。

其他系统则把对文件数据的这种解释留给进程，并且不提供访问数据的帮助。

哪种方法 “更好”，取决于系统中进程的需求，以及用户对操作系统的要求。

如果一个系统主要运行数据库应用程序，操作系统实现数据库类型的文件并提供操作可能更有效，而不是让每个程序实现同样的东西（可能以不同的方式）。

对于通用系统来说，只实现基本的文件类型可能更好，这样可以使操作系统的规模更小，并允许系统中的进程获得最大的自由。

11.3 同样，有些系统支持许多类型的文件数据结构，而另一些系统只是支持字节流。每种方法的优势和劣势是什么？

让系统支持不同的文件结构的一个好处是，这种支持来自于系统；不需要个别应用程序提供支持。此外，如果系统提供对不同文件结构的支持，它可以比应用程序更有效地实现支持。

让系统为定义的文件类型提供支持的缺点是，它增加了系统的大小。此外，可能需要不同的文件类型而不是系统提供的文件类型的应用程序可能无法在这种系统上运行。

另一种策略是操作系统不定义对文件结构的支持，而是将所有文件视为一系列的字节。这就是UNIX系统所采取的方法。这种方法的优点是，它简化了操作系统对文件系统的支持，因为系统不再需要为不同的文件类型提供结构。此外，它允许应用程序定义文件结构，从而缓解了系统可能不提供特定应用程序所需的文件定义的情况。

11.5 解释open()和close()操作的目的。

open( )操作的目的。open( )操作的目的是打开一个活动文件。为此，操作系统维护一个开放文件表，它包含所有开放文件的信息。open()操作可以执行三个任务：取文件名，搜索整个目录，最后将目录条目复制到open-file表中。open()操作还可以验证访问模式。它们是只读、只写、读/写，等等。当请求模式可以接受时，那么该文件就被打开，供进程使用。

close（）操作的目的。close()操作的目的是当进程完成了对一个开放文件的操作后，执行close()操作。close()操作减少了开放计数，它可以与每个文件相关联，以表明有多少进程打开了该文件。当打开次数达到零时，该文件就不再被使用，该文件条目就会从打开文件表中删除。

11.10 open-file表是用来维护当前打开的文件信息的。操作系统应该为每个用户维护一个单独的表，还是只维护一个包含所有用户当前正在访问的文件的表？如果同一个文件被两个不同的程序或用户访问，在开放文件表中是否应该有单独的条目？解释一下。

通过保留一个中央开放文件表，操作系统可以执行以下操作，否则是不可行的。考虑到一个文件目前正被一个或多个进程访问。如果该文件被删除，那么在所有访问该文件的进程都关闭该文件之前，它不应该从磁盘上删除。只有在对访问该文件的进程数量进行集中核算时，才能进行这种检查。

另一方面，如果有两个进程在访问该文件，那么就需要维持两个独立的状态，以跟踪文件的哪些部分正在被两个进程访问的当前位置。这就要求操作系统为这两个进程维护不同的条目。

11.12 提供通常按照以下方法访问文件的应用程序的例子。

顺序：连续访问文件的应用程序–文字处理器、视频播放器、音乐播放器和网络管理。

随机：任意访问文件的应用程序 - 数据库、视频和声音编辑器。

第十二章

12.1 考虑一个目前由100个块组成的文件。假设文件控制块（和索引块，在索引分配的情况下）已经在内存中。计算一下，对于一个块，如果以下条件成立，连续、链接和索引（单级）分配策略需要多少次磁盘I/O操作。在连续分配的情况下，假设在开始时没有增长空间，但在结束时有增长空间。还假设要增加的区块信息存储在内存中。

a. The block is added at the beginning.

b. The block is added in the middle.

c. The block is added at the end.

d. The block is removed from the beginning.

e. The block is removed from the middle.

f. The block is removed from the end.

contiguous linked indexed

a. 201 1 1

b. 101 52 1

c. 1 3 1

d. 198 1 0

e. 98 52 0

f. 0 100 0

12.3 为什么文件分配的位图必须保存在大容量存储器上，而不是保存在主存中？

在系统崩溃（或系统重启）的情况下，自由空间列表不会丢失，因为如果位图被存储在主内存中就会丢失。

12.4 考虑一个支持连续、链接和索引分配策略的系统。在决定哪种策略最适合于某个特定文件时，应该使用什么标准？

* 毗连 - 如果文件通常是按顺序访问的，如果文件相对较小。
* 链接 - 如果文件很大并且通常是按顺序访问。
* 索引的 - 如果文件很大，而且通常是随机访问。

12.8 Explain how the VFS layer allows an operating system to support multiple types of file systems easily.

抽象去耦

VFS在文件系统的实现中引入了一层间接性。在许多方面，它类似于面向对象的编程技术。系统调用可以通用地进行（与文件系统类型无关）。每个文件系统类型都向VFS层提供其函数调用和数据结构。一个系统调用在VFS层被翻译成目标文件系统的适当的特定功能。调用程序没有针对文件系统的代码，而系统调用结构的上层也同样与文件系统无关。在VFS层的翻译将这些通用调用变成了文件系统特定的操作。

12.10 对比一下三种分配磁盘块的技术（连续的、链接的和索引的）在顺序和随机文件访问方面的性能。

毗连顺序–工作得很好，因为文件是毗连存储的。

毗连随机 - 工作良好

链接的顺序–工作得很好，因为你是按照从一个区块到下一个区块的链接进行的。

链接随机–这将是很差的，因为你可能需要绕过链接。

索引式顺序–工作得很好，因为顺序访问只涉及到对每个索引的顺序访问

索 引式随机–效果很好，因为很容易从索引块中确定索引

12.11 使用FAT将文件块链在一起的链接分配的变体有什么优点？

其优点是，在访问存储在文件中间的块时，可以通过追逐存储在FAT中的指针来确定其位置，而不是以顺序的方式访问文件的所有单个块来找到目标块的指针。通常情况下，大部分的FAT可以被缓存在内存中，因此可以只通过内存访问来确定指针，而不必访问磁盘块。

12.15 考虑一个磁盘上的文件系统，它的逻辑和物理块大小都是512字节。假设每个文件的信息已经在内存中。对于三种分配策略（连续的、链接的和索引的）中的每一种，回答这些问题：

a.在这个系统中，逻辑到物理地址的映射是如何完成的？对于索引分配，假设一个文件的长度总是小于512块）

b. 如果我们目前在逻辑块10（最后访问的是块10），想要访问逻辑块4，必须从磁盘上读取多少个物理块？

毗连。用逻辑地址除以512，X和Y分别为所得商和余数。

a. 将X加到Z，得到物理块的编号。Y是进入该块的位移。

b. 1

链接（FAT方法）。将逻辑物理地址除以512，X和Y分别为所得的商和余数。

a. 向下追赶链接列表（得到X+1个块）。Y是进入最后一个物理块的位移。

b. 4

索引的。用逻辑地址除以512，X和Y分别为所得商和余数。

a. 在内存中获取索引块。物理块地址包含在索引块的位置X，Y是进入所需物理块的位移。

b. 2

12.16 考虑一个使用inodes来表示文件的文件系统。磁盘块的大小为8KB，一个磁盘块的指针需要4个字节。这个文件系统有12个直接磁盘块，以及单个、两个和三个间接磁盘块。在这个文件系统中，可以存储的最大文件大小是多少？

(12 \* 8 /KB/) + (2048 \* 8 /KB) + (2048 \* 2048 \* 8 /KB/) +(2048 \* 2048 \* 2048 \* 8 /KB) = 64 terabytes

第十三章

13.5 DMA是如何提高系统并发性的？它是如何使硬件设计复杂化的？

DMA通过允许CPU执行任务，同时DMA系统通过系统和内存总线传输数据来增加系统并发性。硬件设计很复杂，因为DMA控制器必须被集成到系统中，而且系统必须允许DMA控制器成为总线主站。为了让CPU和DMA控制器共享内存总线的使用，可能还需要进行周期窃取。

13.8 当来自不同设备的多个中断大约在同一时间出现时，可以用一个优先级方案来确定中断的服务顺序。讨论在给不同的中断分配优先级时需要考虑哪些问题。

为了确定优先级方案，需要考虑一些问题。

由设备引发的中断应该比由用户程序产生的陷阱有更高的优先权；因此，设备中断可以中断用于处理系统调用的代码。

第二，控制设备的中断可能被赋予更高的优先级，而不是简单的执行任务的中断，例如将设备上的数据复制到用户/内核缓冲区，因为这样的任务总是可以被延迟。

第三，对于那些对其数据处理时间有实时限制的设备，应该比其他设备有更高的优先权。

另外，没有任何形式的数据缓冲的设备将不得不被分配更高的优先级，因为数据可能只在短时间内可用。

13.12 与服务中断有关的各种性能开销是什么？

当一个中断发生时，当前执行的进程被中断，其状态被存储在适当的进程控制块中。然后，中断服务例程被分派，以处理中断。在完成对中断的处理后，进程的状态被恢复，进程被恢复。因此，性能开销包括保存和恢复进程状态的费用，以及冲洗指令流水线和在进程重新启动时将指令恢复到流水线的费用。