计算机系统基础(961)

考试题型：问答、分析、编程

总分：50分

一. 处理器体系结构

CPU 中的时序电路

时序系统是控制器的心脏，其功能就是为指令的执行提供各种定时信号

1. 指令周期和机器周期

   指令周期是指从取指令，分析指令到执行完该指令所需的全部时间

   机器周期又称 CPU 周期。通常把一个指令周期划分为若干过机器周期。

   指令周期 = I * 机器周期
2. 节拍

   把一个机器周期分为若干个相等的时间段，每一个时间段对应一个电位信号，称为节拍
3. 工作脉冲

   在一个节拍内常常设置一个或几个工作脉冲，作为各种同步脉冲的来源。

每个指令周期中常采用机器周期，节拍和工作脉冲三级时序系统。

单周期处理器的设计

单周期 CPU 指的是一条指令的执行在一个时钟周期内完成，然后开始下一条指令的执行，即一条指令用一个时钟周期完成。

CPU 在处理指令时，一般需要经过以下几个步骤：

(1) 取指令(IF)：根据程序计数器 PC 中的指令地址，从存储器中取出一条指令，同时，PC 根据指令字长度自动递增产生下一条指令所需要的指令地址，但遇到“地址转移”指令时，则控制器把“转移地址”送入 PC，当然得到的“地址”需要做些变换才送入 PC。

(2) 指令译码(ID)：对取指令操作中得到的指令进行分析并译码，确定这条指令需要完成的操作，从而产生相应的操作控制信号，用于驱动执行状态中的各种操作。

(3) 指令执行(EXE)：根据指令译码得到的操作控制信号，具体地执行指令动作，然后转移到结果写回状态。

(4) 存储器访问(MEM)：所有需要访问存储器的操作都将在这个步骤中执行，该步骤给出存储器的数据地址，把数据写入到存储器中数据地址所指定的存储单元或者从存储器中得到数据地址单元中的数据。

(5) 结果写回(WB)：指令执行的结果或者访问存储器中得到的数据写回相应的目的寄存器中。

单周期 CPU，是在一个时钟周期内完成这五个阶段的处理。

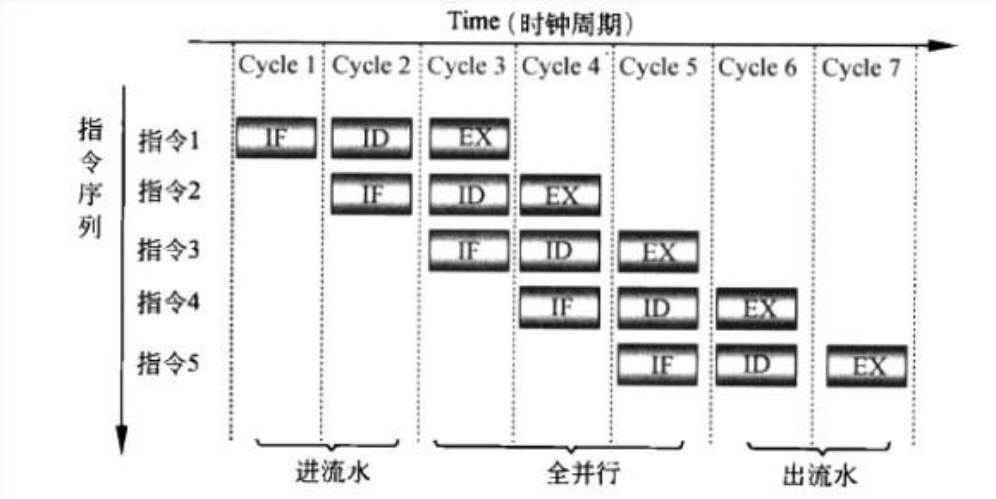
流水线处理器的基本原理

流水线(Pipeline)技术是指程序在执行时候多条指令重叠进行操作的一种准并行处理实现技术。通俗的讲将一个时序过程，分解成若干个子过程，每个过程都能有效的与其他子过程同时执行。旨在提高处理器处理效率，争取在一个时钟周期中完成一条指令。

将处理组织成阶段：取指、译码、执行、访存、写回。

通常一条指令包含很多操作，可以将它们组织成一定的阶段序列，从而便于放入一个通用框架来进行流水线处理。

为什么能提高性能？由于流水线上的各部件并行工作，机器的吞吐量大大提高



Data Hazard 的处理

流水线给处理器带来了效率，当然也有问题。称之为流水线冒险(HaZard)。

1、结构冒险

由于处理器资源冲突，而无法实现某些指令或者阶段的组合实现，就称之为处理器有结构冒险。

比如，早期的处理器中，程序和数据是存储在一起的，那么容易出现下图的情况：在第四个 cycle 中，IF 和 MEM 同时访问存储器导致有一个操作要等待，此时 hazard 就出现了。现在的处理器已经解决了该问题： 指令存储在 L1P cache 中，数据存储 L1D cache 中，单独访问，不会影响相互操作。

2、数据冒险

如果流水线中原来有先后顺序的指令同一时刻处理时，可能会导致出现访问了错误的数据的情况。

在汇编语句中，add R1,R2,R3 将寄存器 R2 和 R3 的和赋予 R1，改变 R1 的值；而紧接着下面的语句：add R4,R1,R5 则会使用 R1 的值，可是 R1 必须在第一条语句中的第 5 个 cycle 才能更新到寄存器中，

语句二是在第 4 个 cycle 就要访问 R1，也就是说第二,条指令此时在使用错误的 R1 的值。这是数据 hazard 出现了。

解决方案：在两条指令中添加一条空指令：nop。但是会影响处理器的指令的执行效率。在现代处理器技术中，已经用 forwarding 的方式解决了。如下图，如果处理器在检测到当前指令的源操作数正好在流水线的 EX 或者 MEM 阶段，接直接将 ex 和 mem 寄存器的值传递给 ALU 的输入，而不是再从寄存器堆中获取数据了。因为此时寄存器堆中的数据可能是没有被及时更新的。

3、控制冒险

在流水线中的的执行指令时，由于并行处理的关系，后面很多指令其实都在流水线中开始处理了，包括预取值和译码。那么，如果此时程序中出现一条跳转语句怎么办呢？因为程序已经跑到其他地址处执行，流水线中之前已经做好的预取值和译码动作都不能使用了。这些会被处理器的专有部件 flush 掉，重新开始新的流水线。此时我们可以称之为出现了控制 hazard。这种情况对于程序和效率来说是存在很大损失的。解决方案：也就是在 jump 指令后面(不会被真正使用，但是会进入流水线)添加 nop。在 MIPS 程序中，经常在 jump 指令后面添加 nop 语句。

在 X86 架构中，是通过硬件来实现 flush，将无效的流水线排空，以保证正确运行流水线。这里会涉及到分支预测技术的使用。

在其他一些处理器中，用软件的方式来处理，添加 nop。同时在编译器中通过乱序的思想用有效指令代替 nop。这样也可以避免转跳带来的性能损失。

流水线设计中的其它问题

1、每个阶段所用的硬件实际并不是相互独立的；增加的寄存器也会导致延迟增大；每阶段的周期划分也很难做到一致。

2、理想的流水线系统，每个阶段的时间都是相等的。实际上，各个阶段的时间是不等的。运行时钟是由最慢的阶段决定的。

3、另外流水线过深，寄存器的增加会造成延迟增大。当延迟增大到时钟周期的一定比例后，也会成为流水线吞吐量的一个制约因素。

二． 优化程序性能

优化程序性能

1、 程序优化的第一步就是消除不必要的内容，让代码尽可能有效地执行它期望的工作。这包括消除不必要的函数调用、条件测试和存储器引用。这些优化不依赖于目标机器的任何具体属性。

2、程序优化的第二步，利用处理器提供的指令级并行能力，同时执行多条指令。

3、最后对大型程序的优化，使用代码剖析程序，代码剖析程序是测量程序各个部分性能的工具这种分析能够帮助找到代码中低效率的地方，并且确定程序中应该着重优化的部分

4、阿姆达尔定律 Amdahl 定律，它量化了对系统某个部分进行优化所带来的整体效果

串行比例越低且处理器越多，加速比越高，程序优化效率越高。

定义了串行系统并行化后的加速比的计算公式和理论上限。

加速比 = 优化之前系统耗时 / 优化后系统耗时

优化编译器的能力和局限性

编译器遵循的一个优化程序的原则是：安全优化. 编译器都不会对程序进行各种激进的优化，所以程序员必须以一种简化编译器生成高效代码的任务来编写程序

表示程序性能

从程序员的角度来看，用时钟周期来表示度量标准要比用纳秒或者皮秒来表示有用的多。用时钟周期来表示，度量值表示的是执行了多少条指令，而不是时钟运行的有多快。

特定体系结构或应用特性的性能优化

与机器无关:

1. **消除循环的低效率**：将每次循环中执行多次但计算结果不改变的部分提出循环，这样只需计算一次，而不用循环一次，计算一次。以此提高算法效率。
2. **减少过程调用**: 也就是减少函数方法的调用，因为函数方法的调用会带来相当大的开销。但是这样也会带来缺点，就是破坏程序的模块化，所以需要我们权衡利弊。
3. **消除不必要的存储器引用：** 在循环中不停的对指针所指的变量赋值的时候，我们可以用一个中间变量代替指针，以增加速度。
4. **选择合适的算法和数据结构：** 为遇到的问题选择合适的算法和数据结构，避免使用产生糟糕性能的算法或变成技术

与机器相关:

1. **理解现代处理器**： 现代微处理器了不起的成就就是它们采用复杂而奇异的微处理结构，多条指令可以并行执行，同时又呈现出一种简单的顺序执行指令的表象。
2. **提高并行性**: 循环分割，利用功能单元的流水线化的能力提高代码性能。

限制因素

寄存器溢出：循环并行性的好处受到描述计算的汇编代码的能力限制。

特别地，IA32 指令集只有很少量的寄存器来存放积累的值（IA32 只有 4 个，x86-64 可以 12 个）。如果我们的并行度 p 超过了可用的寄存器数量，那么编译器会诉诸溢出（spilling），将某些临时值存放到栈中。一旦出现这种情况，性能会急剧下降。

分支预测和预测错误处罚：当分支预测逻辑不能正确预测一个分支是否要跳转的时候，条 件分支可能会招致严重的预测错误处罚。对于这个问题没有简单的答案，有一些通用原则

1.不要过分关心可预测的分支

2.书写适合用条件传送实现的代码

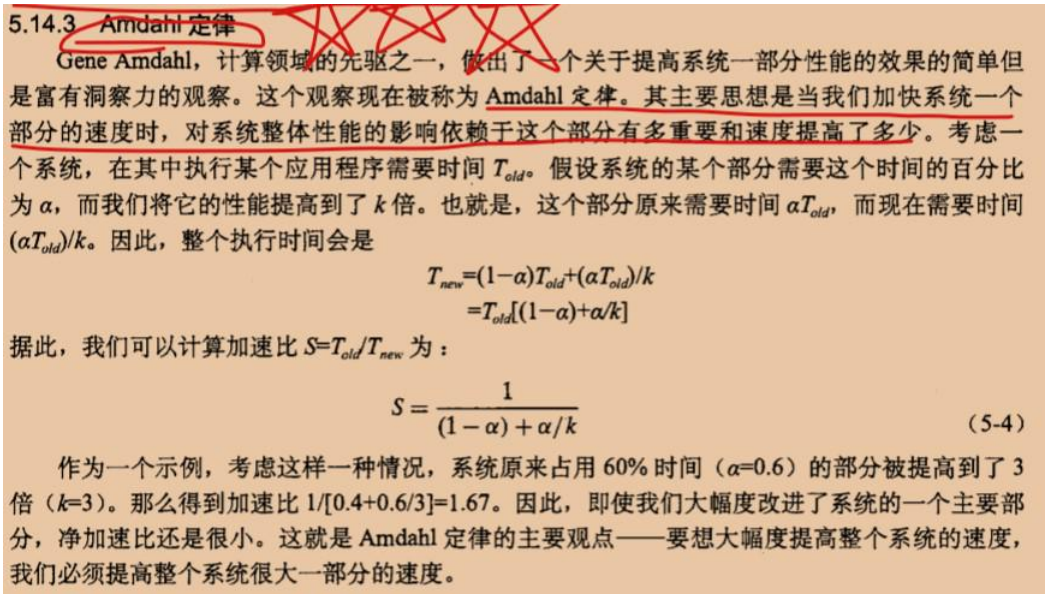
消除性能瓶颈

处理大程序时连知道该优化什么地方都很困难。

- 程序剖析：程序剖析包括运行程序的一个版本，其中插入了工具代码，以确定程序的各个部分需要

多少时间，这对于确认需要集中注意力优化的部分很有用，剖析的一个有力指出在于可以在现实的基准数据上运行实际程序的同时，进行剖析（Unix 系统提供了一个剖析程序 GPROF）。通常，假设在有代表性的数据上运行程序，剖析能帮助我们对典型的情况进行优化，但是我们还应该确保对所有可能的情况，程序都有相当的性能。这主要包括避免得到糟糕的渐近性能的算法（例如插入算法）和坏的编程实践

- Amdahl 定律：其主要思想是当我加快系统一个部分的速度时，对系统整体性能的影响依赖于这个部分有多重要和速度提高了多少。



5.14.3 Amdahl定律

Gene Amdahl，计算领域的先驱之一，做出了一个关于提高系统一部分性能的效果的简单但是富有洞察力的观察。这个观察现在被称为 Amdahl 定律。其主要思想是当我们加快系统一个部分的速度时，对系统整体性能的影响依赖于这个部分有多重要和速度提高了多少。考虑一个系统，在其中执行某个应用程序需要时间 T_{old}。假设系统的某个部分需要这个时间的百分比为 α，而我们将它的性能提高到了 k 倍。也就是，这个部分原来需要时间 αT_{old}，而现在需要时间 $(\alpha T_{old})/k$。因此，整个执行时间会是

$$T_{new}=(1-\alpha)T_{old}+(\alpha T_{old})/k$$
$$=T_{old}[(1-\alpha)+\alpha/k]$$

据此，我们可以计算加速比 $S=T_{old}/T_{new}$ 为：

$$S=\frac{1}{(1-\alpha)+\alpha/k} \quad (5\text{-}4)$$

作为一个示例，考虑这样一种情况，系统原来占用 60% 时间（α=0.6）的部分被提高到了 3 倍（k=3）。那么得到加速比 1/[0.4+0.6/3]=1.67。因此，即使我们大幅度改进了系统的一个主要部分，净加速比还是很小。这就是 Amdahl 定律的主要观点——要想大幅度提高整个系统的速度，我们必须提高整个系统很大一部分的速度。

三. 存储结构及虚拟存储器

局部性

局部性分为时间局部性 temporal(means relating to time) locality 和空间局部性 spatial(空间的) locality:

1. 时间局部性：被引用过一次的**内存位置**可能在短时间内被多次引用；
2. 空间局部性：某个**内存位置**被引用了一次，则**短时间内其附近的内存位置被程序引用。**

局部性在硬件层面、操作系统和应用程序中的应用：

1. 计算机引用小而快的 cache 保存最近使用的指令和数据项；
2. 操作系统中的虚拟内存系统使用内存作为虚拟地址空间最近被引用块的高速缓存；
3. Web 浏览器将最近使用的文档放在本地磁盘上。

怎样提高时间局部性和空间局部性

一个连续的向量，每个 K 个元素进行访问，我们就成为步长为 K 的引用模式 stride-k reference pattern，**步长越长，空间局部性越差，因此我们应该按照数据在内存中的存储位置，按照步长为 1 的模式读取数据**；

取指令的局部性/编写高速缓存友好的代码：

指令不同于数据，是不能被修改的，cache friendly 的代码编写的基本方法：

1. **核心函数(常用函数)的循环应该运行的块，针对其进行优化，其他可以忽略**；
2. 尽量减少核心循环内部的缓存不命中数量；
3. 对局部变量的反复引用是好的，因为他们缓存在寄存器中（时间局部性）；
4. 一旦存储器读取了一个对象，就尽可能多的引用他。

存储器层次结构

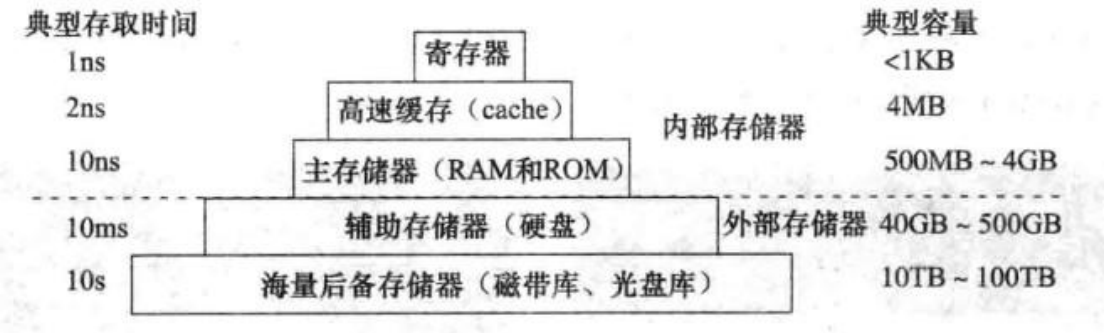


图 6.3　存储器层次化体系结构示意图

计算机高速缓存器原理

高速缓存器是介于中央处理器和主存储器之间的高速小容量存储器

实现原理: 把 CPU 最近最可能用到的少量信息（数据或指令）从主存复制到 CACHE 中，当 CPU 下次再用到这些信息时，它就不必访问慢速的主存，而直接从快速的 CACHE 中得到，从而提高了速度。

高速缓存对性能的影响

高速缓冲存储器的性能常用命中率来衡量. 影响命中率的因素是高速存储器的容量、存储单元组的大小、组数多少、地址联想比较方法、替换算法、写操作处理方法和程序特性等。

1. CACHE 的容量与命中率的关系: cache 容量越大，命中率就越高。虽然容量大一些好，但 CACHE 容量达到一定大小之后，再增加其容量对命中率的提高并不明显。

2. 命中率与主存块的大小有关。采用大的交换单位能很好地利用空间局部性，但是，较大的主存块需要花费较多的时间来存取，因此，缺失损失会变大。由此可见，主存块的大小必须适中，不能太大，也不能太小。

3. 命中率与关联度有关。关联度越高，命中率越高。关联度反映一个主存块对应的 cache 行的个数，显然， 直接映射的关联度为 1；2 路组相联映射的关联度为 2，4 路组相联映射的关联度为 4，全相联映射的关联度为 cache 行数。

地址空间

物理地址空间：程序中访问的内存地址都是实际的物理内存地址。缺点: 1. 进程地址空间不隔离 2. 内存使用效率低; 3. 程序运行的地址不确定

虚拟地址空间: 程序中访问的内存地址不再是实际的物理内存地址，而是一个虚拟地址，然后由操作系统将这个虚拟地址映射到适当的物理内存地址上

映射方式: 分页

主存中的每个字节都有一个虚拟地址空间的虚拟地址，和一个物理地址空间的物理地址。

虚拟存储器

虚拟存储器是指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。

虚拟内存的管理

虚拟存储器具有虚拟性、离散性、多次性及强对换性等特征，其中最重要的特征是虚拟性。
虚拟内存中，允许将一个作业分多次调入内存。采用连续分配方式时，会使相当一部分内存空间都处于暂时或“永久”的空闲状态，造成内存资源的严重浪费，

而且也无法从逻辑上扩大内存容量。因此，虚拟内存的实现需要建立在离散分配的内存管理方式的基础上。

虚拟内存的实现有以下三种方式：

1. 请求分页存储管理
2. 请求分段存储管理
3. 请求段页式存储管理

Cache vs. 虚拟存储器

相同： 1. 出发点相同，都是为了提高存储系统的性价比而构造的分层存储系统；2. 原理相同，都是利用程序运行的局部性原理把最近常用的信息调入高速存储

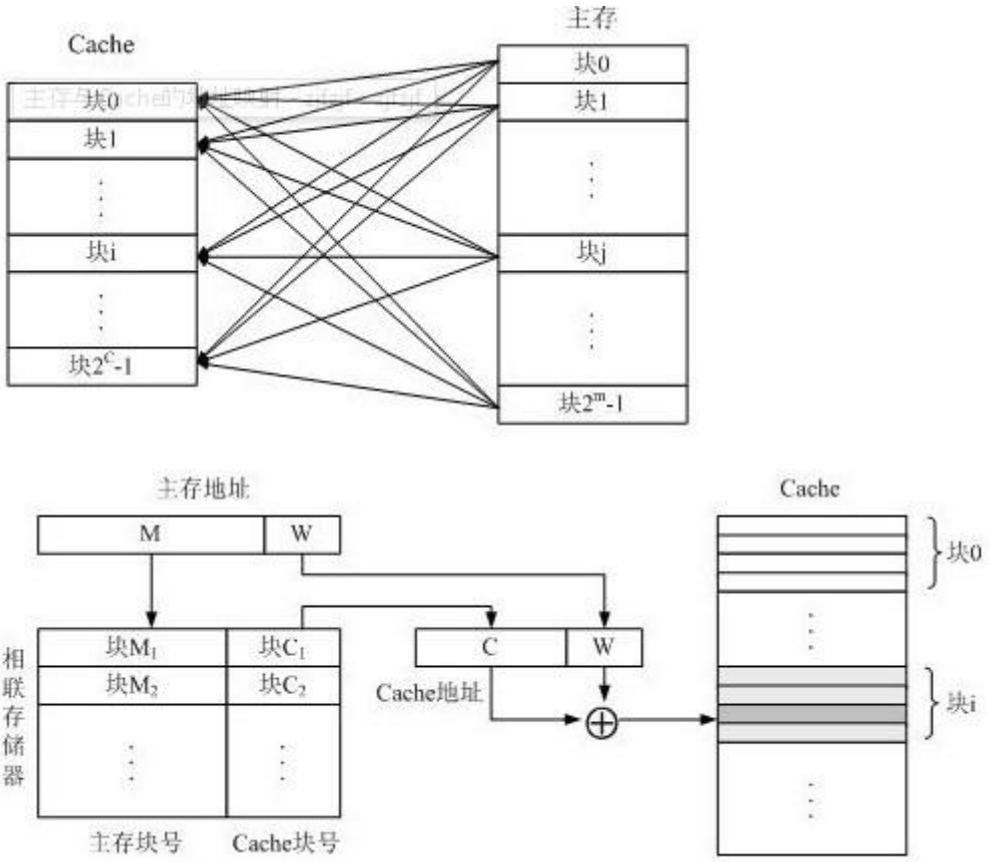
不同: 1. 侧重点不同，catch 解决主存与 CPU 速度差异；虚拟存储器主要解决存储容量问题； 2. 数据通路不同，cache 可以直接访问 CPU/主存，虚拟存储器只能通过主存调页才能访问 CPU; 3. 透明性不同，cache 透明由硬件完成，虚存不透明，由操作系统和硬件完成； 4. 未命中损失不同， 主存未命中时系统损失远大于 cache 未命中

翻译和映射

高速缓存（Cache）与主存地址映射

1. 全相连映射

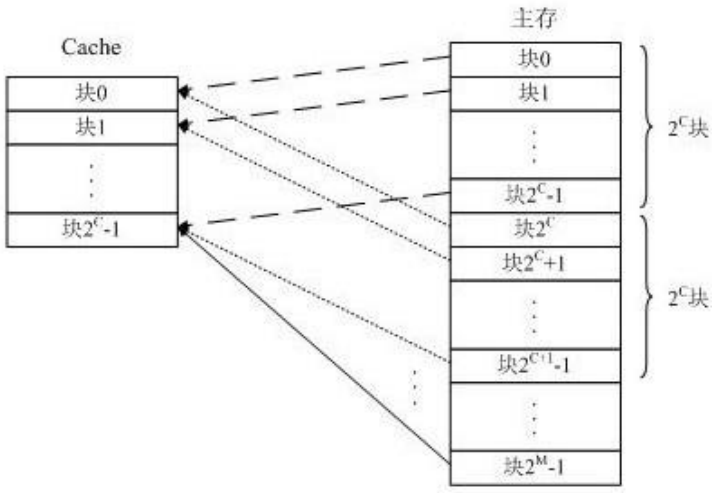
全相联映射是指主存中任一块都可以映射到 Cache 中任一块的方式。

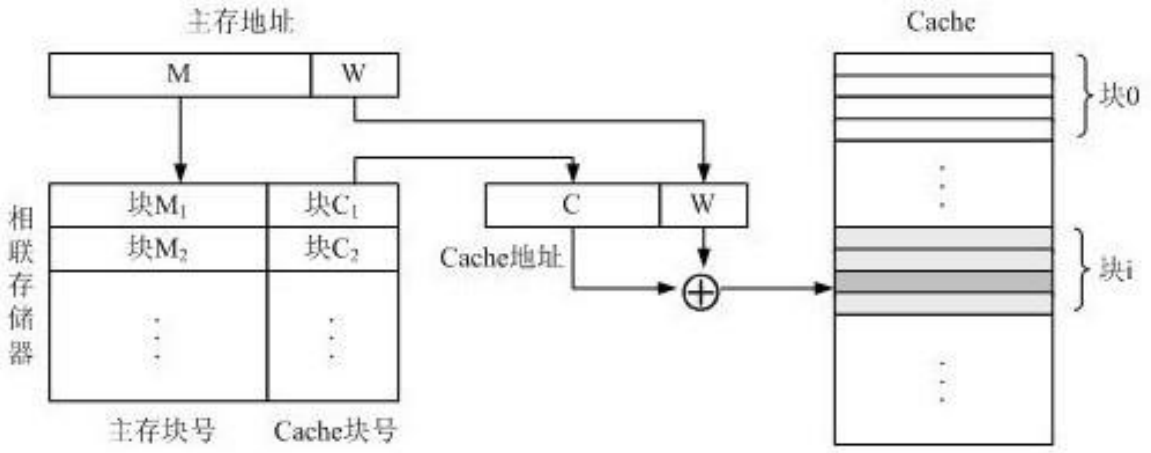


全相联映射方式的优点是 Cache 的空间利用率高，但缺点是相联存储器庞大，比较电路复杂，因此只适合于小容量的 Cache 之用。

2. 直接映射方式

直接相联映射方式是指主存的某块 j 只能映射到满足如下特定关系的 Cache 块 i 中i = j mod 2C





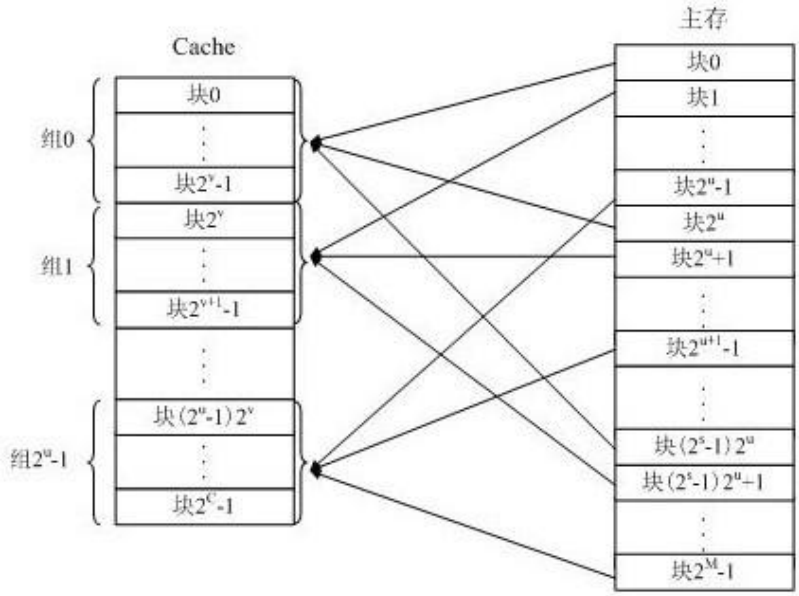
直接相联映射方式的优点 是比较电路最简单，但缺点是 Cache 块冲突率较高，从而降低了 Cache 的利用率。由于主存的每一块只能映射到 Cache 的一个特定块上，当主存的某块 需调入 Cache 时， 如果对应的 Cache 特定块已被占用，而 Cache 中的其它块即使空闲，主存的块也只能通过替换的

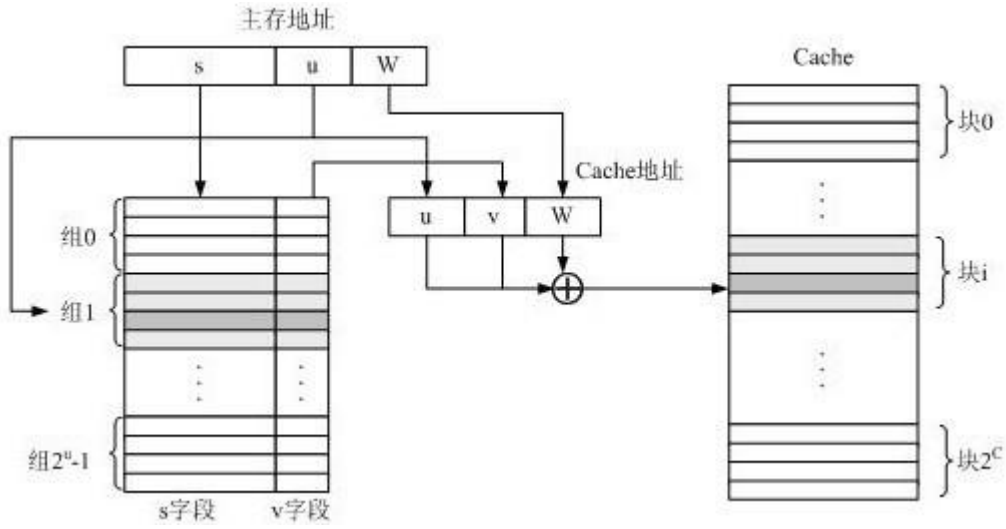
方式调入特定块的位置，不能放置到 其它块的位置上。

3. 组组相连映射方式

将 Cache 分成 2u 组，每组包含 2v 块。主存的块与 Cache 的组之间采用直接相联映射，而与组内的各块则采用全相联映射。也就是说，主存的某块只能映射到 Cache 的特定组中的任意一块。主存的某块 j 与 Cache 的组 k 之间满足如下关系：

k = j mod 2u





TLB

地址转换过程中，访存时首先要到主存查页表，然后才能根据转换得到的物理地址再访问主存以存取指令或数据。如果缺页，则还要进行页面替换、页表修改等，访问主的次数就更多。因此，采用虚拟存储机制后，使得访存次数增加了。为了减少访存次数，往往把页表中最活跃的几个页表项复制到高速缓存中，这种在高速缓存中的页表项组成的页表称为后备转换缓冲器（Translation Lookaside Buffer，简称 TLB），通常称为快表，相应地称主存中的页表为慢表。

动态存储器分配和垃圾收集

需要额外的虚拟存储器时，使用一种动态存储器分配器（dynamic memory allocator）。一个动态存储器分配器维护着一个进程的虚拟存储器区域，称为堆（heap）

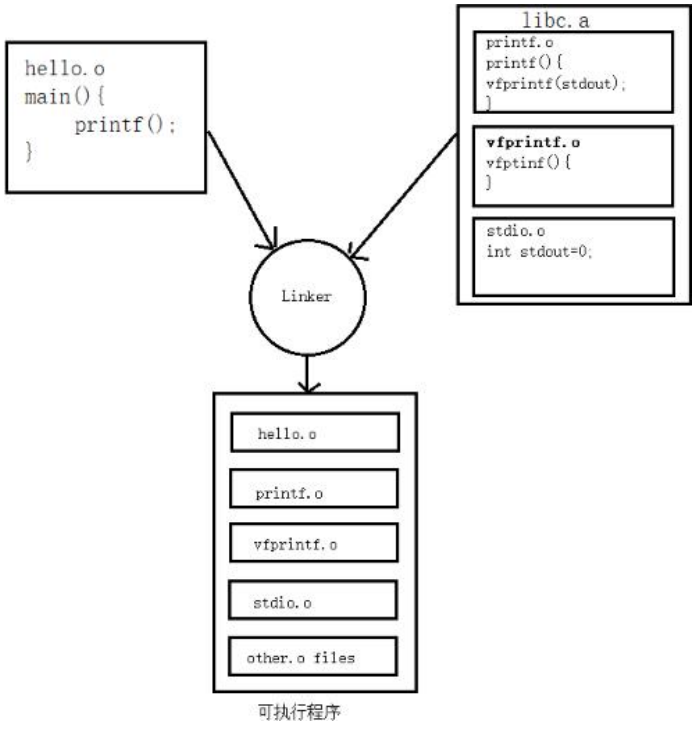
分配器将堆视为一组不同大小的块（block）的集合来维护。每个块就是一个连续的虚拟存储器组块（chunk），要么是已分配的，要么是未分配的。
1）显式分配器（explicit allocator）：如通过 malloc,free 或 C++中通过 new,delete 来分配和释放一个块。
2）隐式分配器（implicit allocator）：也叫做垃圾收集器（garbage collector）。自动释放未使用的已分配的块的过程叫做垃圾回收（garbage collection）。

四．链接，进程以及并发编程

静态链接

为什么要进行静态链接？
在实际的开发过程中，不可能将所有的文件都放在一个源文件中，所以会有多个源文件，而且多个源文件之间不是互相独立的，而会存在多种依赖关系。为了满足这种依赖关系，就需要将源文件与目标文件进行链接，从而形成可以执行的程序。这个链接的过程就是静态链接。
由很多目标文件进行链接形成的是静态库，反之静态库也可以简单地看成是一组目标文件的集合，即很多

目标文件经过压缩打包后形成的一个文件。



静态链接**优点**：在可执行程序中已经具备了所有执行程序所需要的任何东西，在执行的时候运行速度快。

静态链接**缺点**：一是浪费空间；另一方面就是更新比较困难，因为每当库函数的代码修改了，这个时候就需要重新进行编译链接形成可执行程序

目标文件

每个源文件都是可以独立编译的，编译出来的文件就是目标文件。
目标文件是指源文件经过编译程序产生的能被 cpu 直接识别二进制文件

符号和符号表

一个符号表（symbol table），它存放在程序中被定义和引用的函数和全局变量（包括引用到的外部变量和函数，不含有局部变量）的信息。

重定位和加载

运行地址是在程序运行时决定的，在编译链接阶段没有权利也没有办法决定程序的运行地址。链接地址是程序在编译链接阶段由-Ttext 或者 lds 链接文件决定。链接地址和运行地址有时候不能相同，而且不能全部使用位置无关指令，则需要重定位来解决该问题

加载: 将程序拷贝到存储器并运行的过程，由加载器（loader）执行。

动态链接库

原理： 动态链接的基本思想是把程序按照模块拆分成各个相对独立部分，在程序运行时才将它们链接在一起形成一个完整的程序，而不是像静态链接一样把所有程序模块都链接成一个单独的可执行文件。

动态链接的优点显而易见，就是即使需要每个程序都依赖同一个库，但是该库不会像静态链接那样在内存中存在多分，副本，而是这多个程序在执行时共享同一份副本；另一个优点是，更新也比较方便，更新时只需要替换原来的目标文件，而无需将所有的程序再重新链接一遍

动态链接也是有缺点的，因为把链接推迟到了程序运行时，所以每次执行程序都需要进行链接，所以性能会有一定损失。

异常

异常（Exception）就是控制流中的突变，用来响应处理器状态中的某种变化。

在我们的系统中，系统为异常的每种类型都分配了一个唯一的非负整数的异常号。这个异常号相当于一个指针，指向具体的异常。在系统启动时，操作系统分配和初始化一张异常表，它是一张跳转表

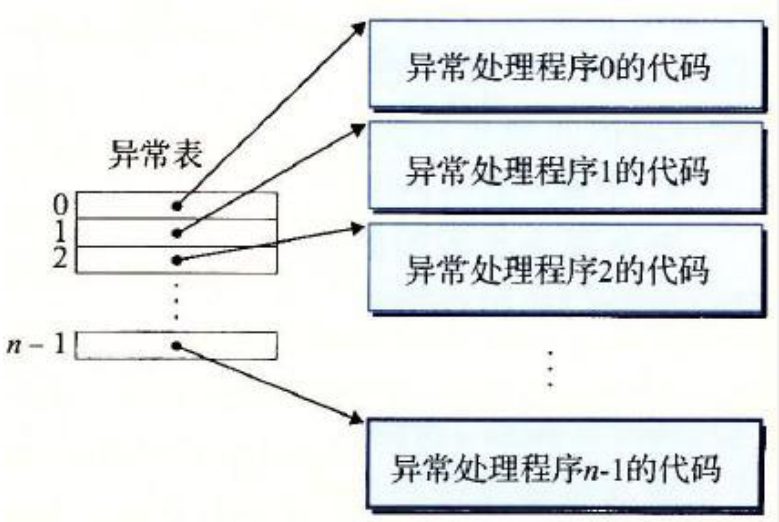


图 8-2 异常表。异常表是一张跳转表，其中表目 k 包含异常 k 的处理程序代码的地址

异常的类别可以分为四类：中断（interrupt）、陷阱（trap）、故障（fault）和终止（abort）

类别	原因	异步 / 同步	返回行为
中断	来自 I/O 设备的信号	异步	总是返回到下一条指令
陷阱	有意的异常	同步	总是返回到下一条指令
故障	潜在可恢复的错误	同步	可能返回到当前指令
终止	不可恢复的错误	同步	不会返回

图 8-4 异常的类别。异步异常是由处理器外部的 I/O 设备中的事件产生的。同步异常是执行一条指令的直接产物

进程

进程：一个具有一定独立功能的程序关于某个数据集合的一次运行活动，是系统进行资源分配和调度运行的基本单位

进程的特征：

1. 动态特征：进程对应于程序的运行，动态产生、消亡，在其生命周期中进程也是动态的、

2. 并发特征：任何进程都可以同其他进程一起向前推进

3. 独立特征：进程是相对完整的调度单位，可以获得 CPU，参与并发执行

4. 交往特征：一个进程在执行过程中可与其他进程产生直接或间接关系

5. 异步特征：每个进程都以相对独立、不可预知的速度向前推进

6. 结构特征：每个进程都有一个 PCB 作为他的数据结构

进程最基本的特征是并发和共享特征

进程的三种基本状态

a. 运行状态：获得 CPU 的进程处于此状态，对应的程序在 CPU 上运行着

b. 阻塞状态：为了等待某个外部事件的发生（如等待 I/O 操作的完成，等待另一个进程发来消息），暂时无法运行。也成为等待状态

c. 就绪状态：具备了一切运行需要的条件，由于其他进程占用 CPU 而暂时无法运行

进程的三个组成部分

a. 程序

b. 数据

c. 进程控制块（PCB）：为了管理和控制进程，系统在创建每个进程时，都为其开辟一个专用的存储区， 用以记录它在系统中的动态特性。系统根据存储区的信息对进程实施控制管理。进程任务完成后，系统收回该存储区，进程随之消亡，这一存储区就是进程控制块

PCB 随着进程的创建而建立，撤销而消亡。系统根据 PCB 感知一个进程的存在，PCB 是进程存在的唯一物理标识（这一点可以类比作业控制块 JCB

进程控制和信号

信号是在软件层次上对中断机制的一种模拟，是一种异步通信方式，信号可以在用户空间进程和内核之间直接交互。内核也可以利用信号来通知用户空间的进程来通知用户空间发生了哪些系统事件。信号事件有两个来源：

1）硬件来源，例如按下了 cltr+C，通常产生中断信号 sigint

2）软件来源，例如使用系统调用或者命令发出信号。最常用的发送信号的系统函数是 kill,raise,setitimer,sigation,sigqueue 函数。软件来源还包括一些非法运算等操作。

进程间的通信

八种通信方式

1.无名管道(pipe)：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。

2.高级管道(popen)：将另一个程序当做一个新的进程在当前程序进程中启动，则它算是当前程序的子进程， 这种方式我们成为高级管道方式。

3.有名管道 (named pipe) ： 有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。

4.消息队列(message queue) ： 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

5.信号量(semophore) ： 信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

6.信号 (sinal) ： 信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

7.共享内存(shared memory) ：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号两，配合使用，来实现进程间的同步和通信。

8.套接字(socket) ： 套解字也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同机器间的进程通信。

进程间信号量的控制、信号量

信号量的本质是一种数据操作锁,它本身不具有数据交换的功能,而是通过控制其他的通信资源(文件,外部设备)来实现进程间通信,它本身只是一种外部资源的标识。信号量在此过程中负责数据操作的互斥、同步等功能。

对信号量的操作

当请求一个使用信号量来表示的资源时,进程需要先读取信号量的值来判断资源是否可用：大于 0,资源可以请求,将信号量的值-1(P 操作);

等于 0,无资源可用,进程会进入睡眠状态直至资源可用;

当进程不再使用一个信号量控制的共享资源时,信号量的值+1(V 操作)。

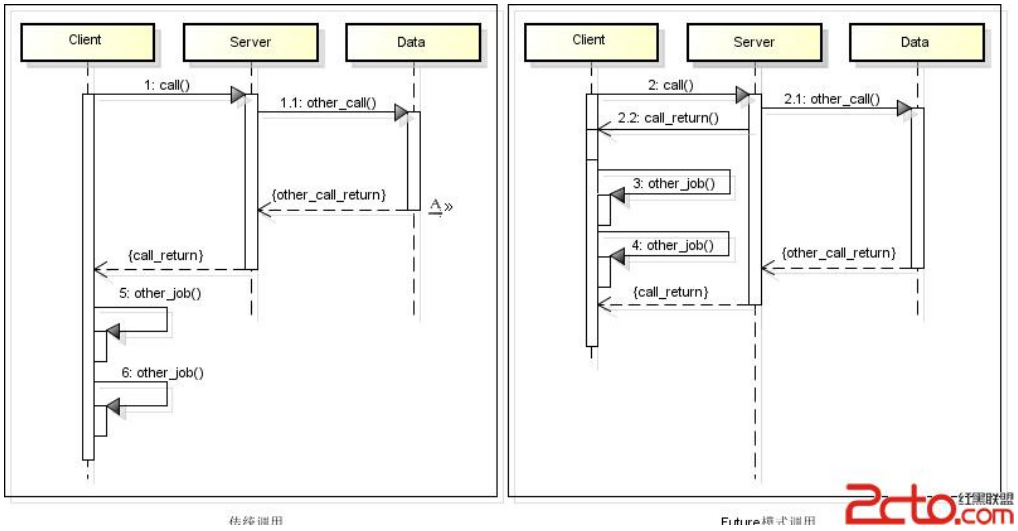
PV 操作均为原子操作。这是由于信号量主要的作用是维护资源的互斥或多进程的同步访问。而在信号量的创建及初始化上,不能保证操作均为原子性(SystemV 版本信号量的缺陷)，因为这个版本的信号量的创建与初始化是分开的。。

各种并发编程模式

一、future 模式

在网上购物时，提交订单后，在收货的这段时间里无需一直在家里等候，可以先干别的事情。类推到程序设计中时，当提交请求时，期望得到答复时，如果这个答复可能很慢。传统的是一直等待到这个答复收到时再去做别的事情，但如果利用 Future 设计模式就无需等待答复的到来，在等待答复的过程中可以干其他事情。

future 模式核心思想就是异步调用，去除了主函数的等待时间，并使得原本需要等待的时间段可以用于处理其他业务逻辑。

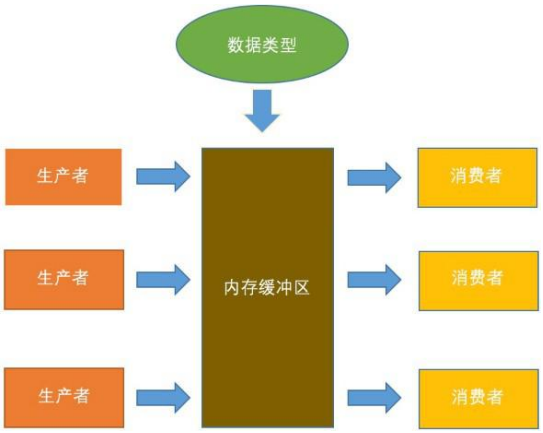


二、Master-Worker 模式

Master-Worker 模式是常用的并行计算模式。它的核心思想是系统由两类进程协作工作：Master 进程和 Worker 进程。Master 负责接收和分配任务，Worker 负责处理子任务。当各个 Worker 子进程处理完成后，会将结果返回给 Master，由 Master 进行归纳和总结。其好处是能将一个大任务分解成若干个小任务，并行执行，从而提高系统的吞吐量。

三、生产者-消费者模式

生产者-消费者模式是一个经典的多线程设计模式。它为多线程间的协作提供了良好的解决方案。 在生产者-消费者模式中，通常由两类线程，即若干个生产者线程和若干个消费者线程。生产者线程负责提交用户请求，消费者线程则负责具体处理生产者提交的任务。生产者和消费者之间则通过共享内存缓冲区进行通信。



共享变量

共享变量：如果一个变量在多个线程的工作内存中都存在副本，那么这个变量就是这几个线程的共享变量。所有的变量都存储在主内存中。

线程同步

线程同步：即当有一个线程在对内存进行操作时，其他线程都不可以对这个内存地址进行操作，直到该线程完成操作， 其他线程才能对该内存地址进行操作，而其他线程又处于等待状态，目前实现线程同步的方法有很多，临界区对象就是其中一种

线程同步的方式和机制

临界区、互斥量、事件、信号量四种方式

临界区（Critical Section）、互斥量（Mutex）、信号量（Semaphore）、事件（Event）的区别

1、临界区：通过对多线程的串行化来访问公共资源或一段代码，速度快，适合控制数据访问。在任意时刻只允许一个线程对共享资源进行访问，如果有多个线程试图访问公共资源，那么在有一个线程进入后，其他试图访问公共资源的线程将被挂起，并一直等到进入临界区的线程离开，临界区在被释放后，其他线程才可以抢占。

2、互斥量：采用互斥对象机制。 只有拥有互斥对象的线程才有访问公共资源的权限，因为互斥对象只有一个，所以能保证公共资源不会同时被多个线程访问。互斥不仅能实现同一应用程序的公共资源安全共享，还能实现不同应用程序的公共资源安全共享

3、信号量：它允许多个线程在同一时刻访问同一资源，但是需要限制在同一时刻访问此资源的最大线程数目

4、事 件： 通过通知操作的方式来保持线程的同步，还可以方便实现对多个线程的优先级比较的操作

其它并行问题

死锁：多个进程因竞争资源而形成一种僵局若无外力作用，这些进程都**将永远不能再向前推进**

互斥：指当有若干进程都要使用某一共享资源时，任何时刻最多允许一个进程使用，其他要使用该资源的进程必须等待，直到占用该资源者释放了该资源为止

死锁的解决办法：

（1）按同一顺序访问对象。

如果所有并发事务按同一顺序访问对象，则发生死锁的可能性会降低。例如，如果两个并发事务获得 Supplier 表上的锁，然后获得 Part 表上的锁，则在其中一个事务完成之前，另一个事务被阻塞在 Supplier 表上。第一个事务提交或回滚后，第二个事务继续进行。不发生死锁。将存储过程用于所有的数据修改可以标准化访问对象的顺序

（2）避免事务中的用户交互。

避免编写包含用户交互的事务，因为运行没有用户交互的批处理的速度要远远快于用户手动响应查询的速

度，例如答复应用程序请求参数的提示。例如，如果事务正在等待用户输入，而用户去吃午餐了或者甚至回家过周末了，则用户将此事务挂起使之不能完成。这样将降低系统的吞吐量，因为事务持有的任何锁只有在事务提交或回滚时才会释放。即使不出现死锁的情况，访问同一资源的其它事务也会被阻塞，等待该事务完成。

（3）保持事务简短并在一个批处理中。

在同一数据库中并发执行多个需要长时间运行的事务时通常发生死锁。事务运行时间越长，其持有排它锁或更新锁的时间也就越长，从而堵塞了其它活动并可能导致死锁。保持事务在一个批处理中，可以最小化事务的网络通信往返量，减少完成事务可能的延迟并释放锁

（4）使用低隔离级别。

确定事务是否能在更低的隔离级别上运行。执行提交读允许事务读取另一个事务已读取（未修改）的数据，而不必等待第一个事务完成。使用较低的隔离级别（例如提交读）而不使用较高的隔离级别（例如可串行读）可以缩短持有共享锁的时间，从而降低了锁定争夺

（5）使用绑定连接。

使用绑定连接使同一应用程序所打开的两个或多个连接可以相互合作。次级连接所获得的任何锁可以象由主连接获得的锁那样持有，反之亦然，因此不会相互阻塞。

五．系统级 I/O 和网络编程

I/O 相关概念

输入输出系统：

I/0 设备与主机交换信息时的物种控制方式：

程序查询方式

程序中断方式，

直接存储器存取方式(DMA)

i/0 通道方式

I/O 处理机方式

输入输出设备：键盘，鼠标，扫描仪，此卡，语音，显示设备，绘图仪，打印机

文件及文件操作

一个 Unix 文件就是一个 m 字节的序列 B0,B1,⋯,Bk,B(m-1)
所有的 I/O 设备，例如网络、磁盘和中断，都被模型化为文件，而所有的输入和输出都被当做对相应文件的读和写来执行。这种将设备优雅地映射为文件的方式，允许 Unix 内核以一种统一的且一致的方式来执行

文件操作：

- 打开文件
- 改变当前的文件位置
- 读写文件
- 关闭文件
- 共享文件

Unix 共享文件的内核用三种相关的数据结构来表示打开的文件：

描述符表（descriptor table）：每个文件都有它独立的描述符表，它的表项是由进程打开的文件描述符来索引的。每个打开的描述符表项指向文件表中的一个表项。

文件表（file table）：打开文件的集合是由一张文件表来表示的，所有的进程共享这张表。

v-node 表（v-node table）：同文件表一样，所有的进程共享 v-node 表。

多个描述符也可以通过不同的文件表项来引用同一个文件。关键思想是每个描述符都有它自己的文件位置，所以对不同描述符的读操作可以从文件的不同位置获取数据。

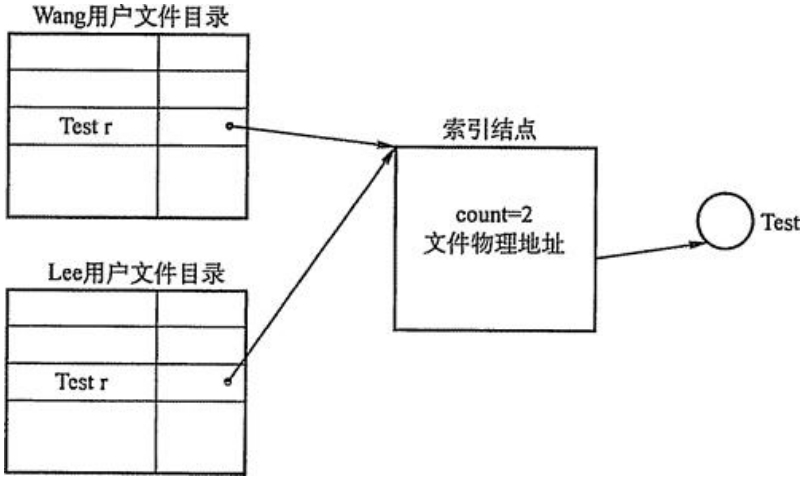
文件共享

文件共享使多个用户（进程）共享同一份文件，系统中只需保留该文件的一份副本。如果系统不能提供共享功能，那么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

现代常用的两种文件共享方法有：

⑴ 基于索引结点的共享方式（硬链接）

在树形结构的目录中，当有两个或多个用户要共享一个子目录或文件时，必须将共享文件或子目录链接到两个或多个用户的目录中，才能方便地找到该文件，如下图所示。



在这种共享方式中引用索引结点，即诸如文件的物理地址及其他的文件属性等信息，不再是放在目录项中，而是放在索引结点中。在文件目录中只设置文件名及指向相应索引结点的指针。在索引结点中还应有一个链接计数count,用于表示链接到本索引结点（亦即文件） 上的用户目录项的数目。当count=2时，表示有两个用户目录项链接到本文件上，或者说是有两个用户共享此文件。

当用户A创建一个新文件时，它便是该文件的所有者，此时将count置为1。当有用户 B要共享此文件时，在用户B的目录中增加一个目录项，并设置一指针指向该文件的索引结点。此时，文件主仍然是用户A，count=2。如果用户A不再需要此文件，不能将文件直接删除。因为，若删除了该文件，也必然删除了该文件的索引结点，这样便会便用户B的指针悬空，而用户B则可能正在此文件上执行写操作，此时用户B会无法访问到文件。因此用户A不能删除此文件，只是将该文件的count减1，然后删除自己目录中的相应目录项。用户B仍可以使用该文件。当COunt=0时，表示没有用户使用该文件，系统将负责删除该文件。如图4-8给出了用户B链接到文件上的前、后情况。

(2) 利用符号链实现文件共享（软链接）

为使用户B能共享用户A的一个文件F,可以由系统创建一个LINK类型的新文件，也取名为F，并将文件F写入用户B的目录中，以实现用户B的目录与文件F的链接。在新文件中只包含被链接文件F的路径名。这样的链接方法被称为符号链接。

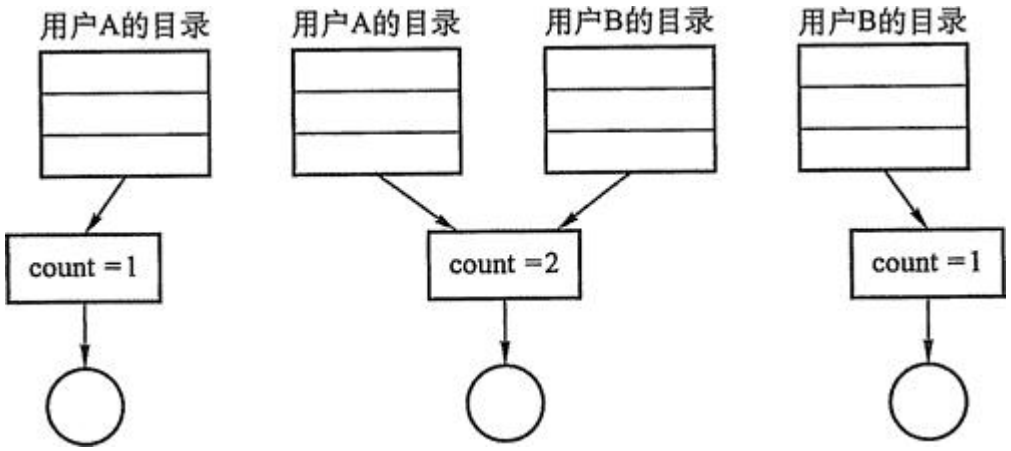


图4-8 文件共享中的链接计数

新文件中的路径名则只被看做是符号链，当用户B要访问被链接的文件F且正要读 LINK类新文件时，操作系统根据新文件中的路径名去读该文件，从而实现了用户B对文件 F的共享。

在利用符号链方式实现文件共享时，只有文件的拥有者才拥有指向其索引结点的指针。而共享该文件的其他用户则只有该文件的路径名，并不拥有指向其索引结点的指针。这样，也就不会发生在文件主删除一共享文件后留下一悬空指针的情况。当文件的拥有者把一个共享文件删除后，其他用户通过符号链去访问它时，会出现访问失败，于是将符号链删除，此时不会产生任何影响。当然，利用符号链实现文件共享仍然存在问题，例如：一个文件采用符号链方式共享，当文件拥有者将其删除，而在共享的其他用户使用其符号链接访问该文件之前，又有人在同一路径下创建了另一个具有同样名称的文件，则该符号链将仍然有效，但访问的文件已经改变，从而导致错误。

在符号链的共享方式中，当其他用户读共享文件时，需要根据文件路径名逐个地查找目录，直至找到该文件的索引结点。因此，每次访问时，都可能要多次地读盘，使得访问文件的开销变大并增加了启动磁盘的频率。

此外，符号链的索引结点也要耗费一定的磁盘空间。符号链方式有一个很大的优点，即网络共享只需提供该文件所在机器的网络地址以及该机器中的文件路径即可。

上述两种链接方式都存在一个共同的问题，即每个共享文件都有几个文件名。换言之，每增加一条链接，就增加一个文件名。这实质上就是每个用户都使用自己的路径名去访问共享文件。当我们试图去遍历整个文件系统时，将会多次遍历到该共享文件。

硬链接和软链接都是文件系统中的静态共享方法，在文件系统中还存在着另外的共享需求，即两个进程同时对同一个文件进行操作，这样的共享可以称为动态共享。

网络编程

网络编程从大的方面说就是对信息的发送到接收，中间传输为物理线路的作用。

网络编程最主要的工作就是在发送端把信息通过规定好的协议进行组装包，在接收端按照规定好的协议把包进行解析，从而提取出对应的信息，达到通信的目的。中间最主要的就是数据包的组装，数据包的过滤，数据包的捕获，数据包的分析，当然最后再做一些处理，代码、开发工具、数据库、服务器架设和网页设计这 5 部分你都要接触。

客户端-服务器模型

一、浏览器/服务器模型的概念

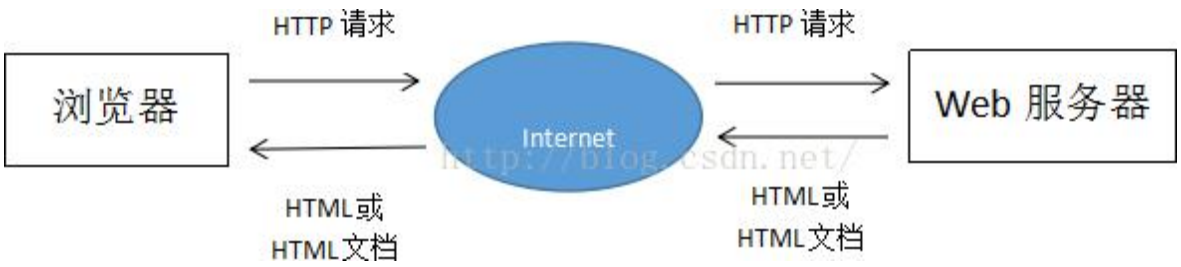
浏览器/服务器模型（B/S 模型）是一种对 C/S 模型的变化和改进，在这种模型中，用户界面完全通过 WWW 浏览器实现，一部分事务逻辑在浏览器实现，大部分事务逻辑在服务器中实现，它是一种特殊的客户/服务器模型，这种模型的客户是某种浏览器，采用 HTTP 协议通信。B/S 模型通常由下面三层架构部署实施：

① 客户端表示层：由 Web 浏览器组成，它不存放任何应用程序。

② 应用服务器层：由一台或多台服务器组成，处理应用中的所有事务逻辑等，具有良好的扩展性，可以随应用的需要增加服务器。

③ 数据中心层：由数据库系统组成，用于存放业务数据。

二、B/S 模型工作的过程



① 用户通过浏览器向 Web 服务器提出 HTTP 请求

② Web 服务器根据请求调用相应的 HTML、XML 文档或 ASP、JSP 文件。如果为 HTML、XML 文档，则直接返回给浏览器；若为 ASP、JSP 文件，Web 服务器首先执行文档中的服务器脚本程序，然后把执行结果返回给浏览器。

③ 浏览器接收到 Web 服务器发回的页面内容，显示给用户。

三、浏览器/服务器模型的优缺点

B/S 模型的优点：

（1）具有分布性特点，可以随时进行查询，浏览等业务处理

（2）业务扩展简方便，通过增加网页即可增加服务器的功能

（3）维护简单方便，只需要改变网页，就可以实现所有用户的同步更新

（4）开发简单，共享性强

B/S 模型的缺点：

（1）操作是以鼠标为最基本的操作方式，无法满足快速操作的要求

（2）页面动态刷新，相应速度明显降低

（3）功能弱化，难以实现传统模式下的特殊功能要求

HTTP 请求

HTTP(HyperText Transfer Protocol)是一套计算机通过网络进行通信的规则。计算机专家设计出 HTTP，使 HTTP 客户（如 Web 浏览器）能够从 HTTP 服务器(Web 服务器)请求信息和服务

Web 服务器

Web 服务器一般指网站服务器，是指驻留于因特网上某种类型计算机的程序，可以向浏览器等 Web 客户端提供文档， [1] 也可以放置网站文件，让全世界浏览；可以放置数据文件，让全世界下载