第 2 章 信息的表示和处理

1. 大多数计算机使用 8 位的块，或者字节，作为最小的可寻址的存储单位，而不是在存储器中访问单独的位。机器级程序将存储器视为一个非常大的字节数组，称为**虚拟存储器**。存储器的每个字节都由一个唯一的数字来标识，称为它的**地址**，所有可能地址的集合称为**虚拟地址空间**。这个虚拟地址空间只是一个展现给机器级程序的概念性映像。实际的实现是将随机访问存储器、磁盘存储器、特殊硬件和操作系统软件结合起来，为程序提供一个看上去统一的字节数组。
2. 某些机器选择在存储器中按照从最低有效字节到最高有效字节的顺序存储对象，这种最低有效字节在最前面的方式，称为**小端法**。大多数 Intel 兼容机都采用这种规则。
3. 位级运算的一个常见用法就是实现掩码运算，这里掩码是一个位模式，表示从一个字中选出的位的集合。
4. 当执行一个运算时，如果它的一个运算数是有符号的而另一个是无符号的，那么 C 语言会隐式地将有符号参数强制类型转换为无符号数，并假设这两个数都是非负的，来执行这个运算。

第 3 章 程序的机器级表示

1. 以一个 C 程序为例，gcc 命令调用一系列程序，将源代码转化成可执行代码：
（1）**C 预处理器**扩展源代码，插入所有用#include 命令指定的文件，并扩展所有用#define 声明指定的宏；
（2）**编译器**产生所有源代码的汇编代码；
（3）**汇编器**将汇编代码转化成二进制目标代码；**目标代码是机器代码的一种形式**，它包含所有指令的二进制表示，但是还没有填入地址的全局值。
（4）**链接器**将所有目标文件与实现库函数的代码合并，并产生最终的可执行代码文件；**可执行代码是机器代码的第二种形式**，也就是处理器执行的代码格式。
2. IA32 加了一条限制，传送指令的两个操作数不能都指向存储器位置，也不允许将立即数传送到存储器。
3. 在 C 语言中，当执行既涉及大小变化又涉及符号改变的强制类型转换时，操作应该先改变大小。
4. 一个过程调用包括将**数据**（以过程参数和返回值的形式）和控制从代码的一部分**传递**到另一部分。另外，它还必须在进入时为过程的局部变量分配空间，并在退出时释放这些空间。大多数机器，包括 IA32，只提供转移控制到过程和从过程中转移出控制这种简单的指令。**数据传送、局部变量的分配和释放**通过操纵程序**栈**来实现。
5. 机器代码中缺乏类型信息。
6. 函数指针的值是该函数机器代码表示中第一条指令的地址。
7. C 对于数组引用不进行任何边界检查，而且局部变量和状态信息（例如保存的寄存器值和返回地址），都存放在栈中。这两种情况结合到一起就可能导致严重的程序错误，对越界的数组元素的写操作会破坏存储在栈中的状态信息。当程序使用这个被破坏的状态，试图重新加载寄存器或执行 ret 指令时，就会出现很严重的错误。一种特别常见的状态破坏称为**缓冲区溢出**。

第 4 章 处理器体系结构

1. 一个处理器支持的指令和指令的字节级编码称为它的**指令集体系结构**（Instruction-Set Architecture，ISA）。

第 5 章 优化程序性能

1. 在实际的处理器中，是同时对多条指令求值，这个现象称为**指令级并行**。
2. 浮点乘法和加法是不可结合的。大多数编译器不会尝试对浮点运算做重新结合，因为这些运算不保证是可结合的。
3. 循环展开和并行地累积在多个值中，是提高程序性能的更可靠的方法。

第 6 章 存储器层次结构

1. 静态 RAM（SRAM）比动态 RAM（DRAM）更快，但也贵得多。**SRAM 用来作为高速缓存存储器，DRAM 用来作为主存以及图形系统的帧缓冲区。**典型地，一个桌面系统的 SRAM 不会超过几兆字节，但是 DRAM 却有几百或几千兆字节。
2. 虽然 ROM 中有的类型既可以读也可以写，但是它们整体上称都为只读存储器。
3. PROM(Programmable ROM)只能被编程一次。PROM 的每个存储器单元有一种熔丝，它只能用高电流熔断一次。EPROM(Erasable PROM)能够被擦除和重编程的次数的数量级可达到 1000 次。EEPROM(Electrically EPROM)类似于 EPROM，但是它不需要一个物理上独立的编程设备，因此可以直接在印制电路卡上编程，它能够被编程的次数的数量级可达到 100000 次。
4. 存储在 ROM 设备中的程序通常称为**固件**(firmware)。
5. 读事务从主存传送数据到 CPU。写事务从 CPU 传送数据到主存。
6. 总线是一组并行的导线，能携带地址、数据和控制信号。
7. USB2.0 总线的最大带宽为 60MB/s，USB3.0 总线的最大带宽为 600MB/s。
8. 设备可以自己执行读或者写总线事务，而不需要 CPU 干涉的过程，称为直接存储器访问。这种数据传送称为 DMA 传送。
9. 一个编写良好的计算机程序常常具有良好的**局部性**（locality）。也就是说，它们倾向于引用邻近于其他最近引用过的数据项的数据项，或者最近引用过的数据项本身。通常有两种不同的形式：时间局部性和空间局部性。在一个具有良好**时间局部性**的程序中，被引用过一次的存储器位置很可能在不远的将来再被多次引用。在一个具有良好**空间局部性**的程序中，如果一个存储器位置被引用了一次，那么程序很可能在不远的将来引用附近的一个存储器位置。
10. 抖动（thrash）：高速缓存反复地加载和驱逐相同的高速缓存块的组。
11. 任何合理的优化编译器都会把局部变量缓存在寄存器文件中，也就是存储器层次结构的最高层。
12. 旋转磁盘是机械的非易失性存储设备，固态硬盘（SSD,Solid State Disk）基于非易失性的

闪存，闪存基于 EEPROM。
13. 存储器层次结构：
（1）寄存器是最快的；
（2）SRAM 作为高速缓存，在 L1~L3 级；
（3）DRAM 作为主存，在 L4 级；
（4）每一层上的存储器作为低一层存储器的**高速缓存**。
如下图所示：

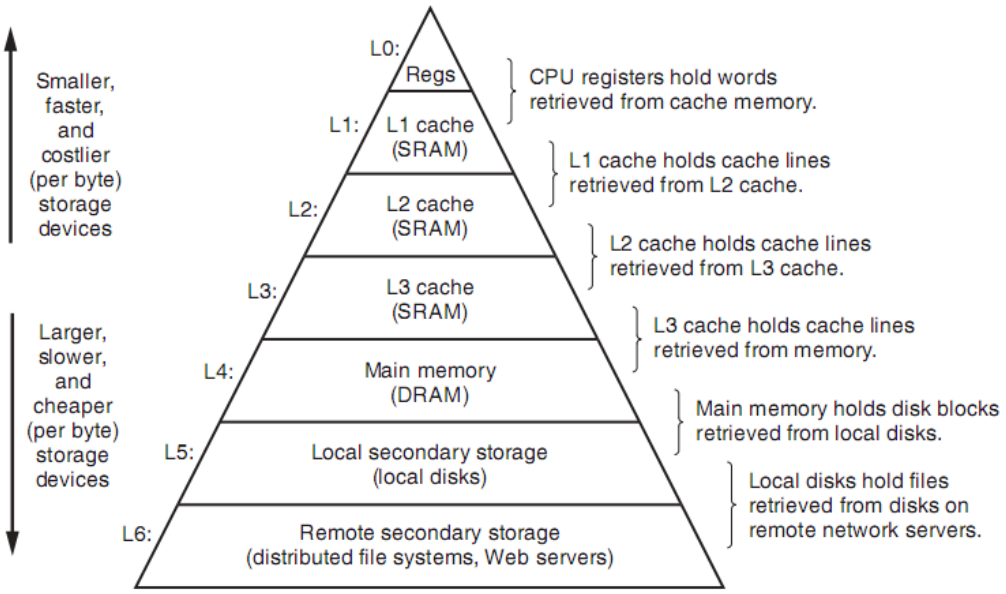

Figure 6.23 **The memory hierarchy.**

第 7 章 链接

1. 局部 C 变量在运行时保存在栈中。
2. 定义为带有 C static 属性的本地过程变量是不在栈中管理的。
3. 将程序拷贝到存储器并运行的过程叫做**加载**。
4. 在 32 位 Linux 系统中，**代码段**总是从地址 **0x08048000** 处开始的（对于 64 位，是从 **0x00400000** 开始）。
5. **共享库**的一个主要目的就是允许多个正在运行的进程共享存储器中相同的库代码，因而节约宝贵的存储器资源。

第 8 章 异常控制流

1. 现代系统通过使控制流发生突变来对系统状态的变化做出反应。一般而言，我们把这些突变称为**异常控制流**（ECF）。
2. 异常处理程序运行在**内核模式**下。
3. 为了使操作系统内核提供一个无懈可击的进程抽象，处理器必须提供一种机制，限制一

个应用可以执行的指令以及它可以访问的地址空间范围。处理器通常是用某个控制寄存器中的**模式位**来提供这种功能的。
内核模式（**超级用户模式**）：设置了模式位，运行在该模式下的进程可以执行指令集中的任何指令，并且可以访问系统中任何存储器位置。
用户模式：没有设置模式位，不允许执行特权指令，如停止处理器、改变位模式发起一个 I/O 操作，必须通过系统调用间接（不能直接）访问内核代码和数据。

第 9 章 虚拟存储器

1. 虚拟存储器提供了三个重要的能力：
（1）它将主存看成是一个存储在磁盘上的地址空间的高速缓存，在主存中只保存活动区域，并根据需要在磁盘和主存之间来回传送数据，通过这种方式，它高效地使用了主存。
（2）它为每个进程提供了一致的地址空间，从而简化了存储器管理。
（3）它保护了每个进程的地址空间不被其他进程破坏。
2. 将一个虚拟地址转换为物理地址的任务叫做地址翻译。
3. Linux 为每个进程维持了一个单独的虚拟地址空间。

第 10 章 系统级 I/O

1. size_t 被定义为 unsigned int，而 ssize_t（有符号的大小）被定义为 int。
2. 一个**文本行**就是一个由换行符结尾的 ASCII 码字符序列。
3. **套接字**是一种用来通过网络与其他进程通信的**文件**。

第 11 章 网络编程

1. 认识到**客户端和服务器是进程**，而不是常常提到的机器或者主机，这是很重要的。
2. 一个**套接字**是连接的一个端点。每个套接字都有相应的**套接字地址**，是由一个因特网地址和一个 16 位的整数端口组成的，用“**地址：端口**”来表示。Web 服务器通常使用端口 80，而电子邮件服务器使用端口 25。
3. 显式地关闭已经打开的任何**描述符**（打开文件后，内核返回的一个小的非负整数）是一个良好的编程习惯。
4. HTTP 标准要求每个文本行都由一对回车和换行符（在 C 的表示中为“\r\n”）来结束。

第 12 章 并发编程

1. 线程就是运行在进程上下文中的逻辑流。**每一个线程都有它自己的线程上下文**，包括一个唯一的整数**线程 ID**（TID）、**栈**、**栈指针**、**程序计数器**、**通用目的寄存器**和**条件码**。所有的运行在一个进程里的线程共享该进程的整个虚拟地址空间。**多个线程**运行在单一进程的上

下文中，因此**共享**这个**进程虚拟地址空间的整个内容**，包括它的**代码**、**数据**、**堆**、**共享库**和**打开的文件**。（注意每个线程都有自己的栈、共享同一进程的堆。区别栈和堆，可以类比局部和全局的概念：“局部是自己的，全局是可以共享的”。）

2. 一个线程的上下文要比一个进程的上下文小得多，线程的上下文切换要比进程的上下文切换快得多。

3. 每个进程开始生命周期时都是单一线程，这个线程称为**主线程**。某一时刻由主线程创建一个**对等线程**（peer thread），从这个时间点开始，两个线程并发地运行。

4. 和一个进程相关的线程组成一个对等（线程）池（pool），独立于其他线程创建的线程。主线程和其他线程的区别仅在于它总是进程中第一个运行的线程。对等（线程）池概念的主要影响是，一个线程可以杀死它的任何对等线程，或者等待它的任意对等线程终止。**每个对等线程都能读写相同的共享数据**。

5. 如果想传递多个参数给线程例程，那么你应该将参数放到一个结构中，并传递一个指向该结构的指针。相似地，如果你想要线程例程返回多个参数，你可以返回一个指向一个结构的指针。

6. 在任何一个时间点上，线程是**可结合的**（joinable）或者是**分离的**（detached）。一个可结合的线程能被其他线程收回其资源和杀死。在被其他线程回收之前，它的存储器资源（例如栈）是没有被释放的。相反，一个分离的线程是不能被其他线程回收或杀死的。它的存储器资源在它终止时由系统自动释放。默认情况下，线程被创建成可结合的。为了避免存储器泄露，每个可结合线程都应该要么被其他线程显式地回收，要么通过调用 pthread_detach 函数被分离。

7. 因为不同的线程栈是不对其他线程设防的。所以，如果一个线程以某种方式得到一个指向其他线程栈的指针，那么它就可以读写这个栈的任何部分。

8. 无论是在单处理器还是多处理器上运行程序，都要**同步**你对**共享变量**的访问。

9. 饥饿就是一个线程无限期地阻塞，无法进展。

10. Java 线程是用一种叫做 Java 监控器（Java Monitor）的机制来同步的。

11. 有一类重要的线程安全函数，叫做可重入函数（reentrant function）：当它们被多个线程调用时，不会引用任何共享数据。

12. 一个函数被称为线程安全的，当且仅当被多个并发线程反复地调用时，它会一直产生正确的结果。如果一个函数不是线程安全的，我们就说它是线程不安全的。

13. 可重入函数是线程安全函数的一个真子集，它不访问任何共享数据。