3.7　过程

过程是软件中一种很重要的抽象。它提供了一种封装代码的方式，用一组指定的参数和一个可选的返回值实现了某种功能。然后，可以在程序中不同的地方调用这个函数。设计良好的软件用过程作为抽象机制，隐藏某个行为的具体实现，同时又提供清晰简洁的接口定义，说明要计算的是哪些值，过程会对程序状态产生什么样的影响。不同编程语言中，过程的形式多样：函数(function)、方法(method)、子例程(subroutine)、处理函数(handler)等等，但是它们有一些共有的特性。

要提供对过程的机器级支持，必须要处理许多不同的属性。为了讨论方便，假设过程 P 调用过程 Q，Q 执行后返回到 P。这些动作包括下面一个或多个机制：

传递控制。在进入过程 Q 的时候，程序计数器必须被设置为 Q 的代码的起始地址，然后在返回时，要把程序计数器设置为 P 中调用 Q 后面那条指令的地址。

传递数据。P 必须能够向 Q 提供一个或多个参数，Q 必须能够向 P 返回一个值。

分配和释放内存。在开始时，Q 可能需要为局部变量分配空间，而在返回前，又必须释放这些存储空间。

x86-64 的过程实现包括一组特殊的指令和一些对机器资源(例如寄存器和程序内存)使用的约定规则。人们花了大量的力气来尽量减少过程调用的开销。所以，它遵循了被认为是最低要求策略的方法，只实现上述机制中每个过程所必需的那些。接下来，我们一步步地构建起不同的机制，先描述控制，再描述数据传递，最后是内存管理。

3.7.1　运行时栈

C 语言过程调用机制的一个关键特性(大多数其他语言也是如此)在于使用了栈数据结构提供的后进先出的内存管理原则。在过程 P 调用过程 Q 的例子中，可以看到当 Q 在执行时，P 以及所有在向上追溯到 P 的调用链中的过程，都是暂时被挂起的。当 Q 运行时，它只需要为局部变量分配新的存储空间，或者设置到另一个过程的调用。另一方面，当 Q 返回时，任何它所分配的局部存储空间都可以被释放。因此，程序可以用栈来管理它的过程所需要的存储空间，栈和程序寄存器存放着传递控制和数据、分配内存所需要的信息。当 P 调用 Q 时，控制和数据信息添加到栈尾。当 P 返回时，这些信息会释放掉。

如 3.4.4 节中讲过的，x86-64 的栈向低地址方向增长，而栈指针 %rsp 指向栈顶元素。可以用 pushq 和 popq 指令将数据存入栈中或是从栈中取出。将栈指针减小一个适当的量可以为没有指定初始值的数据在栈上分配空间。类似地，可以通过增加栈指针来释放空间。

当 x86-64 过程需要的存储空间超出寄存器能够存放的大小时，就会在栈上分配空间。这个部分称为过程的*栈帧*(stack fram)。图 3-25

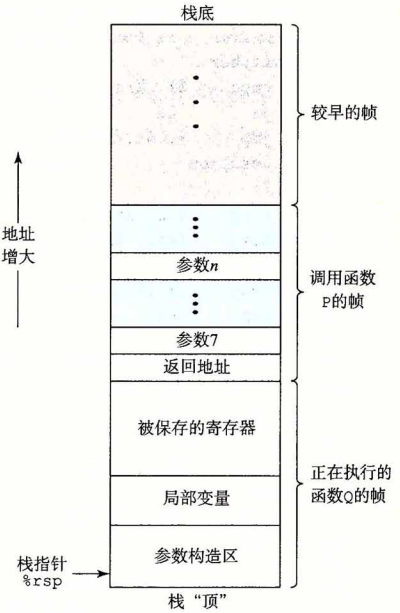

图 3-25　通用的栈帧结构(栈用来传递参数、存储返回信息、保存寄存器，以及局部存储。省略了不必要的部分)