2.3 线程

1．线程基本概念

线程定义为进程内一个执行单元或一个可调度实体。在不拥有线程的进程概念中，进程既是一个拥有资源的独立单位，它可独立分配虚地址空间、主存和其它，又是一个可独立调度和分派的基本单位。在有了线程以后，资源拥有单位称为进程（或任务），调度的单位称为线程、又称轻进程（Light Weight Process，LWP）。如果进程有多个控制线程，那么它能同时做多个任务。图2.7 说明了传统单线程进程和多线程进程的差别。

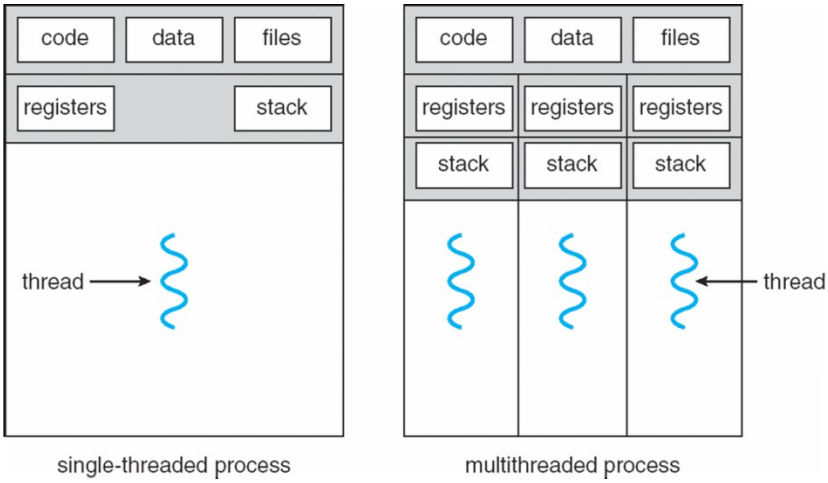

图 2.7 单线程进程和多线程进程

多线程的进程在同一地址空间内包括多个不同的控制流，也即属于同一进程下的线程，它们共享进程拥有的资源，如代码、数据、文件等。线程也独占一些资源，如堆栈、程序计数器等。多线程系统的优点包括对用户响应的改进，进程内的资源共享，以及利用多处理器体系结构的便利。

2．多线程模型

从实现的角度看，把线程分为用户级线程和内核级线程。

用户级线程对程序员来说是可见的，而对内核来说是未知的，用户空间的线程库通常用以管理用户级线程，线程库提供对线程创建、调度和管理的支持。因为内核并不知道用户级线程的存在，所有的线程创建和调度工作都在用户空间完成，而且整个过程不受内核的干涉。所以，用户级线程的创建和管理通常很快；然而，它们也有一些缺点。例如：如果内核是单线程的，那么任何用户级线程执行一个导致阻塞的系统调用时就会导致整个进程阻塞，即使程序内部的其它线程可以运行。

内核级线程由操作系统支持和管理，在内核空间实现线程创建、调度和管理。因为线程管理由操作系统完成，所以内核线程的创建和管理要比用户线程慢。然而，由于线程由内核管理，如果一个线程执行一个导致阻塞的系统调用，那么内核可以调度程序中的其它线程执行。同样，在多处理机环境中，内核能够在不同的处理器中调度线程。大多数现代操作系统都支持内核线程。

有三种不同模型将用户级线程和内核级线程关联起来：多对一模型将多个用户线程映

射到一个内核线程；一对一模型将每个用户线程映射到一个相应的内核线程；多对多模型将多个用户线程在同样（或更少）数量的内核线程之间切换。

3．线程库

线程库(thread library)为程序员提供创建和管理线程的 API 。主要有两种方法来实现线程库。第一种方法是在用户空间中提供一个没有内核支持的库，此库的所有代码和数据结构都存在于用户空间中。调用库中的一个函数只是导致了用户空间中的一个本地函数调用，而不是系统调用。第二种方法是执行一个由操作系统直接支持的内核级的库。此时，库的代码和数据结构存在于内核空间中。调用库中的一个 API 函数通常会导致对内核的系统调用。

目前使用的三种主要的线程库是: (1) POSIX Pthread、(2) Win32 、(3) Java 。Pthread 作为 POSIX 标准的扩展，可以提供用户级或内核级的库。 Win32 线程库是适用于 Windows 操作系统的内核级线程库。Java 线程 API 允许线程在 Java 程序中直接创建和管理。然而，由于大多数 JVM 实例运行在宿主操作系统之上， Java 线程 API 通常采用宿主系统上的线程库来实现。这意味着在 Windows 系统上，Java 线程通常用 Win32 API 实现，而在 UNIX 和 Linux 系统中采用 Pthread。

接下来介绍采用 Pthread 和 Win32 两种线程库来创建基本的线程。作为一个说明性的例子，设计一个多线程程序，在独立的线程中完成非负数整数的加法功能: $sum = \sum_{i=0}^{N} i$

例如，如果 N 为 5 ，此函数表示从 0~5 的数相加起来，为 15 。三个程序都要求在命令行输入加法的上限。例如，如果用户输入 8 ，将会输出从 0~8 的整数值的和。