# 进程与线程

**林祥，PB16020923**

## 1、进程的概念

我们知道，人们通过操作系统来更方便地操作硬件，提高使用计算机的效率。早期的计算机只允许一次执行一个程序，这样的程序完全控制了系统，能访问所有的资源。但随着操作系统的功能愈加复杂，计算机的应用更加广泛，人们往往需要同时进行多个作业，因此就有了进程的概念，不同的进程同时运行，互相独立，进行不同的工作。

进程（Process），可以认为是执行中的程序。值得注意的是，程序本身不是进程。程序是被动实体，是存储在磁盘上的包含一系列指令的文件，而进程是活动实体，当一个程序被载入内存后并进行了相关资源分配后，才可以说进程开始运行了。

## 2、进程的信息

一个进程开始运行后，用户内存中将有一段空间被分配给这个进程，具体结构见右图 1。其中，文本段（Text）存放程序代码，数据段（Data）存放全局变量，栈（Stack）存放函数参数、局部变量等，堆（Heap）用于动态分配空间。栈和堆朝着相反的方向增长，这是为了提高内存空间利用率。

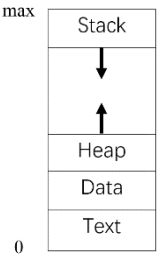

图 1

同时，操作系统会为每个进程在内核空间中设置一个进程控制块（Process Control Block，PCB），其结构如图 2。进程状态由当前进行的活动定义，包括 new（正在被创建）、Running（正在执行指令）、Waiting（等待某事件发生，如 I/O 完成）、Ready（等待处理器分配）、Terminated（终止），它们之间的相互转化见图 3。进程编号是操作系统为每个进程分配的唯一的身份标识。进程计数器表示将要执行的下一个指令的地址。寄存器则存放进程执行时的中间信息。

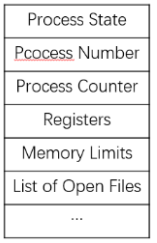

图 2

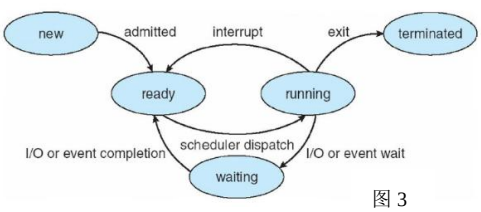

图 3

PCB 的一个重要作用就是多道程序设计中操作系统在切换执行的进程时，可以将当前进程状态保存在 PCB 中，并从 PCB 读取新的进程的信息。

## 3、进程的操作

进程的操作主要包括获取进程 ID、创建进程、终止进程和进程间的通信，这些操作都可以通过系统调用来实现，下面介绍前三者的实现。

**（1）获取进程的 ID。**进程独一无二的身份标识（PID）可以通过 `getpid()`直接获取。

**（2）创建进程。**一个进程可以通过系统调用来创建新的进程，则它本身称为父进程，新进程称为子进程。子进程还能再创建它的子进程，因此于是便形成了进程树。通常操作系统以一个初始进程（如 init）为根形成进程树。

进程创建的一些细节实现可能并不唯一。如进程创建时需要一定的资源（如内存，I/O），它可能从操作系统那里直接获取，也可能从父进程获取。另一方面，子进程创建时，父进程可能与其并发执行，也可能等待子进程执行完后才继续执行。下面以 UNIX 系统为例，说明进程创建的基本过程。

在 UNIX 中，通过系统调用 `fork()`来创建子进程，它将是父进程的一份完全拷贝，包括用户内存空间和 PCB 中的信息，同时进程 PCB 被加入到任务链表中。接下来，子进程的 PID 将被修改为一个与父进程不同的值，进程时间被置为 0，同时它与父进程将相互记录为父子关系。由于 PCB 程序计数器也被完全复制，则子进程将与父进程继续执行相同的代码。另外，一个明显的差别是父进程的 `fork()`将返回子进程的 PID，而子进程的 `fork()`将返回 0，在代码中可以据此区分它们。

对于子进程，在创建完成后，可以不执行其他代码，与父进程保持相同。然而更有意义的做法是通过系统调用 exec()来载入指定路径下的程序，这样进程将执行一个全新的程序，除了保留 PID、程序运行时间和父子关系，包括内存和寄存器中的信息几乎都被丢弃，代码段被重置为载入程序的代码。

对于父进程，在调用 fork()后，如果没有什么事情可做，可以调用 wait()来将自身挂起，等待子进程执行直到结束。

**（3）终止进程。**当进程完成执行后，通过调用 exit()请求系统删除自身，所有的内存信息都将被释放。然而，进程将保留一个空的 PCB 在系统进程链表中，成为“僵尸”进程，同时向父进程发送一个 SIGCHLD 信号。这样做的一个原因是，子进程结束时，父进程可能由于有其他任务而还没有调用 wait()或及时响应。直到 wait()调用后，僵尸子进程才真正从任务链表中被删除。

另外，进程也可能通过系统调用来终止其他进程，通常只能对子进程实施这种操作。当一个进程被终止时，在一些系统中，子进程也被一并结束，另一些系统中，子进程将以 init 为父进程。

## 4、线程的概念和优点

进程帮助人们将不同的工作分离开来，各自运行。更具体地看，一项工作往往需要同时处理多个任务（如服务器同时响应多个客户，浏览器同时获取数据和渲染页面），但我们并不因此创建多个进程，而是在一个进程下创建多个线程（Thread）来实现。这样做的主要原因或者说线程相对于进程的优点有以下几点：进程的创建和切换很耗费时间和资源，而线程之间可以共享进程的代码、数据和文件，大大减小了创建和切换的开销，同时更易于相互之间的通信。另一方面，线程能充分利用处理器的多个核心，真正实现并行，当一个线程出现阻塞时，其他线程并不因此受到影响。

当然，线程也存在一些问题，如怎样分配任务，使每个线程的工作相对平衡？如何保证数据的同步和一致性？

## 5、多线程模型

线程可以分为用户线程和内核线程（由系统直接管理），它们之间有三种关系

（1）多对一模型：既多个用户线程对应一个内核线程，这种做法效率较高，但如果有一个线程执行了阻塞系统调用，那么整个进程都会被阻塞。

（2）一对一模型：既一个用户线程对应一个内核线程，这种做法允许多个线程并行运行在多处理器上，缺点是大多数系统的内核线程数量是有限的。

（3）多对多模型：这种做法多路复用了内核线程，允许任意多的用户线程，当一个线程执行了阻塞调用时，内核可以调度另一个线程来执行。

## 6、多线程相关问题

线程库（Thread library）是为开发人员提供的创建和管理线程的 API，目前主要有三种线程库：POSIX Pthread、Win32 和 Java。

多线程中 fork()调用有两种，一种复制所有线程，一种只复制调用了 fork()的线程，而 exec()与进程中的相同。

线程池（Thread pool）是一种在进程开始时创建一定数量的线程放入池中的机制，当有任务请求时，它唤醒一个线程来处理，完成任务后，线程返回池中再等待工作。线程池解决了服务器中接受大量请求时各自频繁地创建和删除线程带来的巨大耗费。

参考文献:

[1] Abraham Silberschatz. 操作系统概念. 高等教育出版社, 2010.1.