ch5 线程

引入原因

引入思想

进程是**资源分配**的独立单位，**调度**的基本单位。 -> 引入线程后，线程是CPU调度的单位

引入思想：将进程资源分配和调度分开

引入益处

1. 开销

   启动一个新进程必须分配独立地址空间，建立众多的数据表来维护它的代码段、堆栈段，这是一种很“**昂贵**”的多任务工作方式 。

   运行于一个进程中的多个线程，彼此之间使用**相同的**地址空间，**共享**大部分数据，启动一个线程所花费的空间远远小于启动一个进程所花费的空间。

   线程间彼此**切换**所需的时间也远远小于进程间切换所需要的时间。

2. 通信

不同进程具有独立的数据空间，要进行数据的传递只能通过**通信方式**进行，这种方式不仅费时，而且很不方便。

由于同一进程下的线程之间**共享数据空间**，所以一个线程的数据可以直接为其他线程所用，这不仅快捷，而且方便。

总之：

- 创建一个新线程花费时间和资源少
- 两个线程的切换花费时间少
- **同一进程内的线程共享内存和文件**，因此它们之间相互通信无须调用内核
- 适合多处理机系统（实现真正意义上的并行）

线程概念

线程定义

- 线程---轻量级进程
  - 进程中的一个运行实体
  - 是一个**CPU调度单位**
  - **资源的拥有者是进程**
  - **可由内核控制，也可由用户控制**
- 线程也描述为：
  - 进程的执行体、一个执行单元、进程内的一个可调度实体等。
- 超线程
- 多线程
- 可以用单个CPU模拟出线程级并行

线程组成

不拥有系统资源，存取进程的资源

只包含程序计数器、寄存器和一组栈

TCB 线程控制块

不运行时保存上下文

线程状态

创建

阻塞

执行

就绪

完成

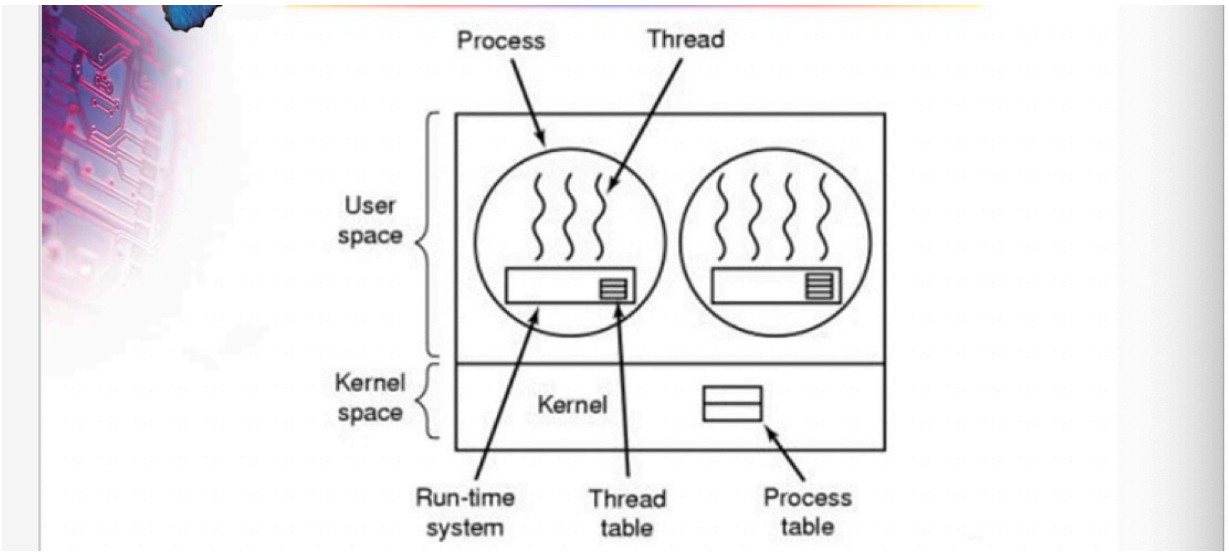
线程模式

- 单进程、单线程(如MS-DOS)
- 单进程、多线程(如VxWorks)
- 多进程、一个进程一个线程(如UNIX)
- 多进程、一个进程多个线程(如Windows NT等)

实现机制

各自的优缺点

用户级线程(User-Level Thread)



通过 **线程库** 管理

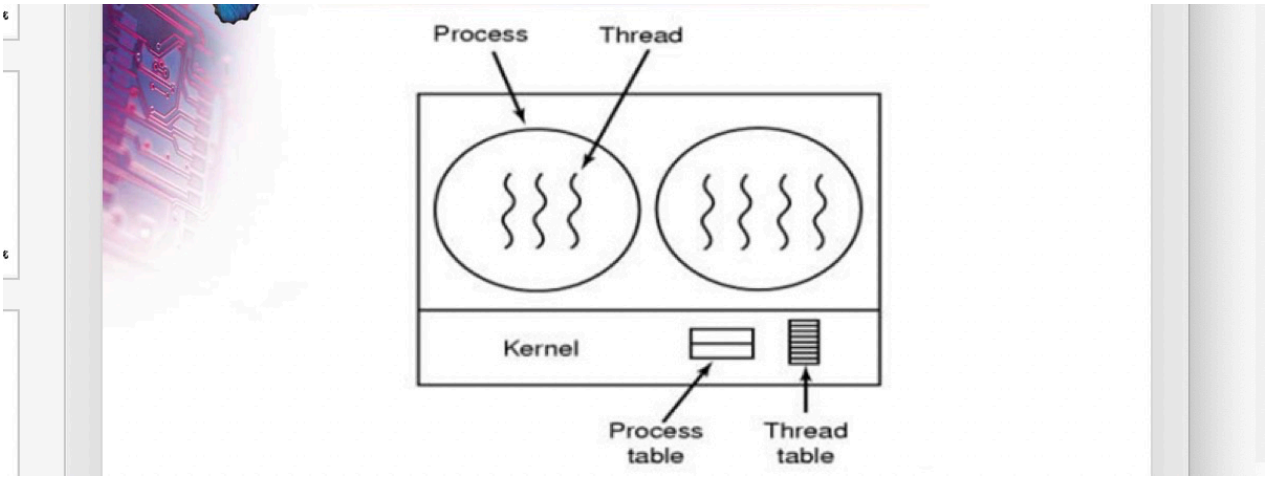
由应用程序完成所有线程的管理

内核不知道线程的存在，只管理进程

线程切换不需要核心态特权

线程调用 系统调用 时，进程阻塞，但线程仍运行，两者的状态独立

内核级线程(Kernel-Level Thread)



线程管理由核心完成

没有线程库，但对核心线程工具提供API

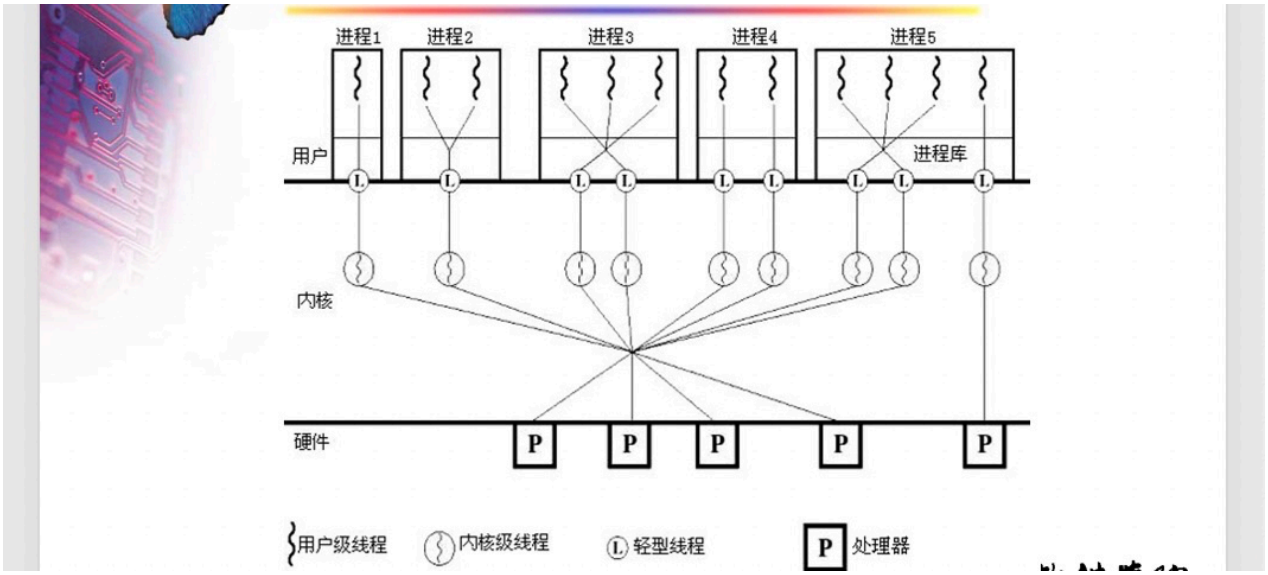
提高了并发行，但速度下降

两者结合方法

多对一：多个用户级线程对应一个内核级线程

一对一

多对多（混合式线程）



若干多线程问题

（应该不考吧...）

系统调用fork和exec

取消

信号处理

线程池

举例