

Chapter 4 线程





内容

1. 什么是线程？
2. 多线程模型
3. 线程库



1、概述



线程引入的原因

线程概念

线程和进程

线程结构

线程优点

Windows线程

Linux线程



■ 引入原因

■ 性能

- 进程是重量级的，负载了程序运行的所有内容，进程操作开销大
- Unix的轻型进程(fork)

■ 应用

- 进程代码有并行执行的需求

■ 硬件

- 多核处理器已经是主流硬件
- 加速进程的运行





线程

- 线程（轻型进程light weight process, LWP ）
- 进程内的一个代码片段可以被创建为一个线程
- 线程状态：就绪、运行、等待等
- 线程操作：创建、撤销、等待、唤醒
- 进程依旧是资源分配的基本单位
- 线程自己不拥有完整的系统资源，通过进程申请资源





■ 传统进程：

- 可称为重型进程(**heavy weight process**)
- 等价于只有一个线程的任务，主线程
- 单线程模型





■ 操作系统引入线程后, 资源分配的基本单位是()

- A. 线程
- B. 进程
- C. 超线程
- D. 以上都不是





线程和进程

代码

进程包含
线程

线程是进
程中的一
段代码

资源

进程是资
源分配基
本单位

线程不拥
有资源,
共享进程
的资源

调度

同一进程
中的线程
切换不会
引起进程
切换

线程是基
本调度单
位

切换

进程:重
量级,上
下文切换
代价大

线程:轻
量级,代
价小

生命 期

进程撤销
会导致它
的所有线
程被撤销

线程撤销
不会影响
进程





线程结构

- 代码和数据：来自进程
- 各类资源：来自进程
- 线程控制块(Thread Control Block, TCB)
 - 线程ID
 - 线程计数器PC
 - 寄存器集
 - 栈空间





Windows 2000 Process and Thread

Windows 2000进程和线程

Microsoft Spy++ - [进程 2]

Spy(S) 目录树(T) 搜索(E) 视图(V) 窗口(W) 帮助(H)

进程 00000000 IDLE
进程 00000008 SYSTEM
进程 00000090 SMSS
 线程 00000060 SMSS
 线程 0000008C SMSS
 线程 00000094 SMSS
 线程 00000098 SMSS
 线程 0000009C SMSS
 线程 000000A0 SMSS
进程 000000A8 CSRSS
进程 000000BC WINLOGON
进程 000000D8 SERVICES
进程 000000E4 LSASS
进程 00000178 SVCHOST
 线程 00000174 SVCHOST
 线程 0000017C SVCHOST
 线程 00000184 SVCHOST
 线程 00000188 SVCHOST
 线程 000001DC SVCHOST
 线程 0000024C SVCHOST
 线程 000002B4 SVCHOST
 线程 000002EC SVCHOST
 线程 000003E4 SVCHOST
 线程 00000500 SVCHOST
进程 00000190 CCEVTMGR
进程 000001F0 SPOOLSV
进程 00000210 SVCHOST
进程 00000224 MDM

若要获取帮助，请按 F1

线程属性

常规

模块名: SVCHOST

线程 ID:	00000500	当前优先级:	10
进程 ID:	00000178	基本优先级:	8
线程状态:	等待	起始地址:	77E69824
等待原因:	执行延迟	用户 PC:	2012778846
CPU 时间:	0:00:00.000	上下文开关:	86
用户时间:	0:00:00.000		
特权时间:	0:00:00.000		
运行时间:	23:26:28.283		

关闭 刷新 帮助

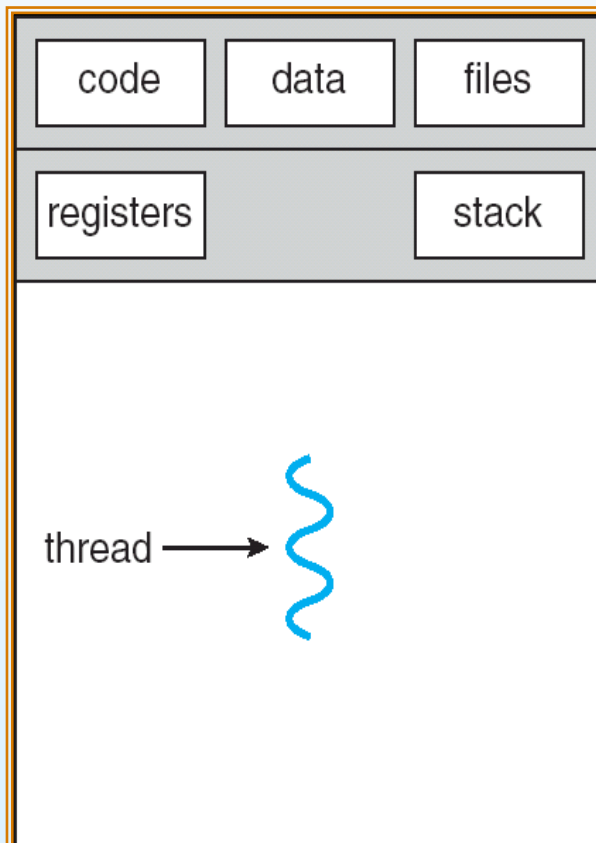




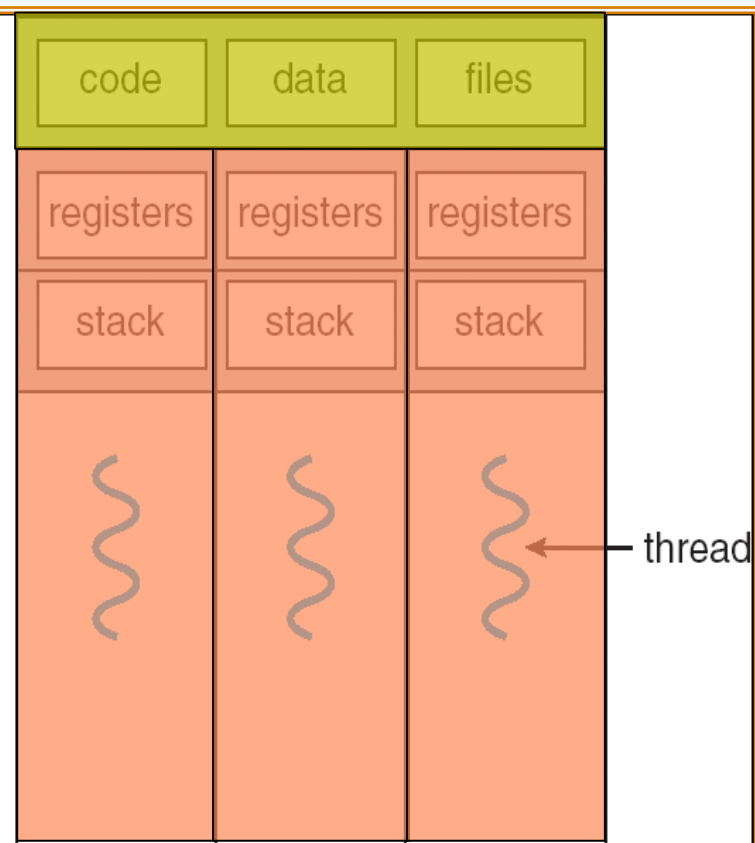
单个线程和多线程进程

单线程

- 一个进程只有一个线程
- 早期操作系统



single-threaded process



multithreaded process

多线程

- 一个进程有多个线程
- 支持线程的操作系统





线程优点

■ 响应度高

- 线程创建开销小
- 例子：Web服务器

■ 资源共享

- 进程中的线程可以共享进程资源

■ 经济性

- 线程创建、上下文切换比进程快
- Solaris：创建线程比进程快30倍，线程切换比进程切换快5倍

■ MP体系结构的运用

- 一个进程中的线程在不同处理器上并行运行，缩短了运行时间





Web服务器



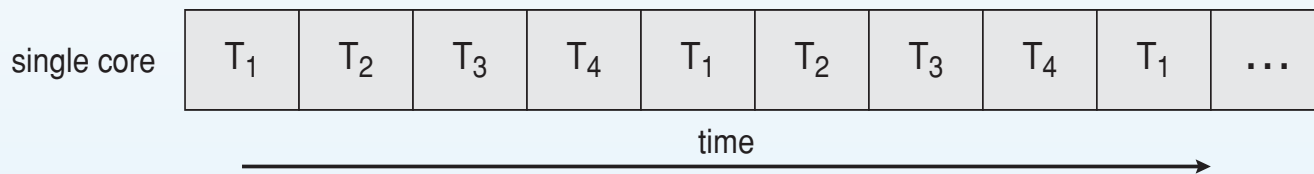
`<html>.......</html>`



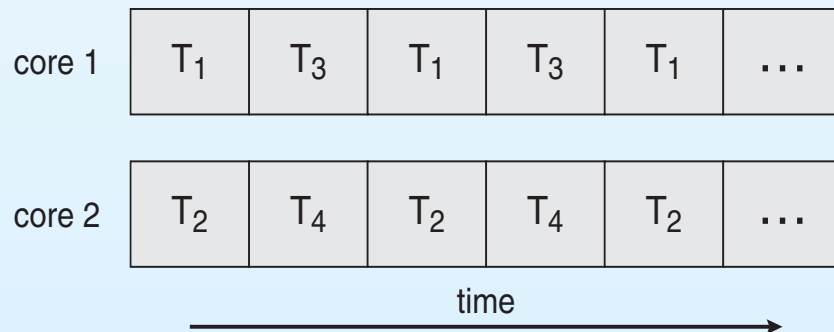


并发（Concurrency）和并行（Parallelism）

■ 单核系统并发：



■ 多核系统并行：





Windows 线程

■ 操作系统支持线程

■ 线程主要数据结构:

● 执行线程块ETHREAD

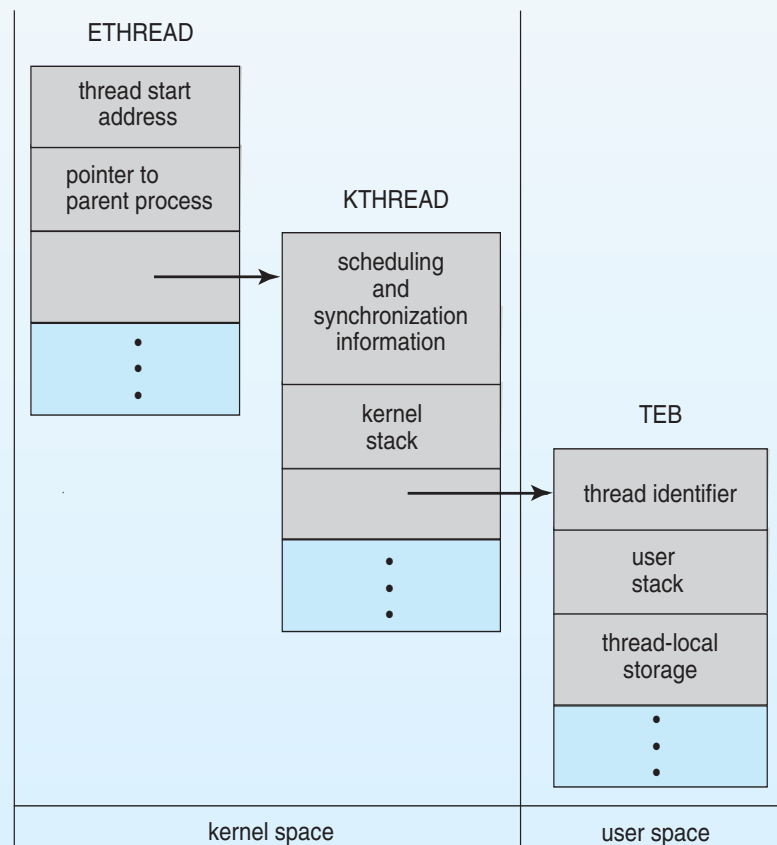
- ▶ Executive thread block
- ▶ 对应程序地址
- ▶ 指向KTHREAD指针
- ▶ 内核空间

● 核心线程块KTHREAD

- ▶ Kernel thread block
- ▶ 线程调度和同步信息
- ▶ 内核空间

● 线程环境块TEB

- ▶ Thread environment block
- ▶ 用户空间的数据结构, 线程本地存储





Linux 线程

- Linux 2.2版引入线程
- 通过 **clone()** 系统调用创建线程
 - 类似fork
 - Linux只支持克隆线程
- 用术语“任务”表示进程和线程，一般不用“线程”
- 用户线程：PThreads





Clone

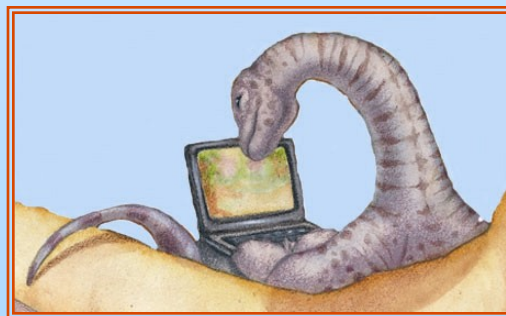
- 创建进程或者线程
- Clone可有选择性继承父进程的资源
 - 不和父进程共享

int clone(int (*fn)(void *), void *child_stack, int flags, void *arg)

- fn为函数指针，此指针指向一个函数体，即想要创建进程的静态程序；
- child_stack为给子进程分配系统堆栈的指针
- flags为要复制资源的标志，描述你需要从父进程继承那些资源



2、多线程模型



用户线程
内核线程
多对一模型
一对一模型
多对多模型

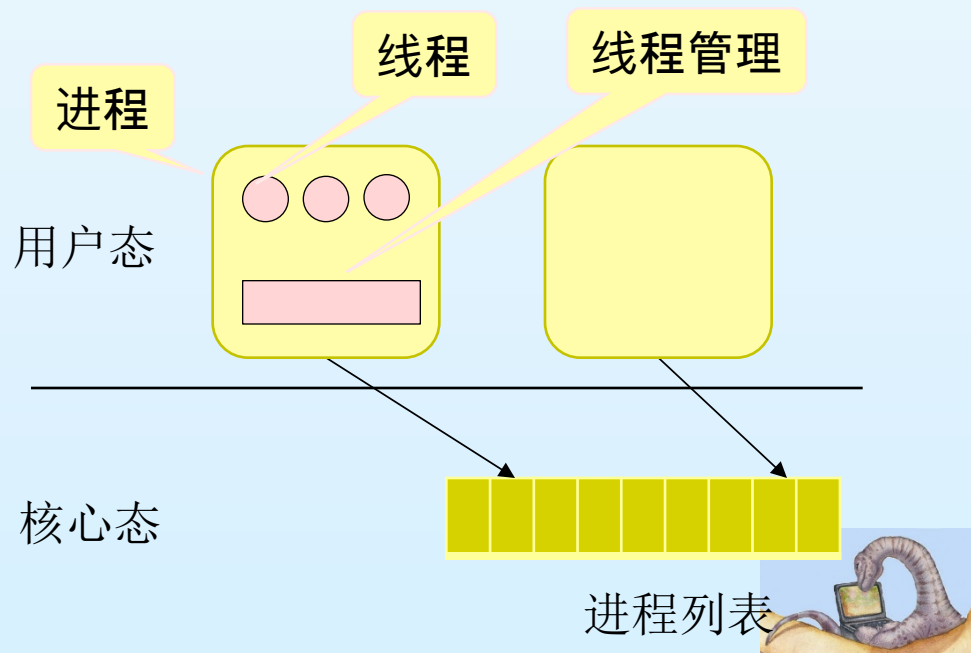


用户线程

- 用户线程：由用户线程库进行管理的线程
 - 内核看不到用户线程
 - 用户线程的创建和调度在用户空间中，不需要内核的干预
 - 应用于传统的只支持进程的操作系统

- 例子

- POSIX *Pthreads*
- Win32 *threads*
- Java *threads*





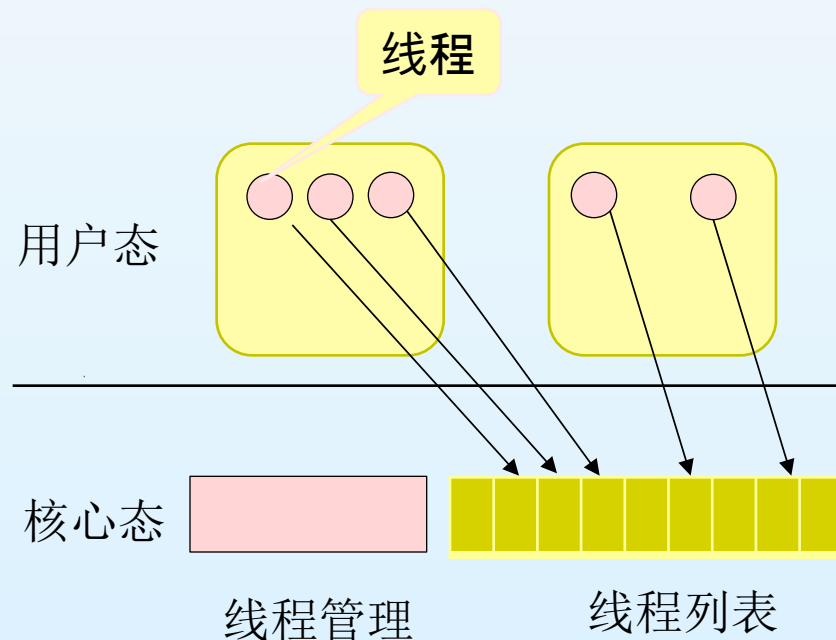
内核线程

■ 内核线程：内核进行管理的线程

- 需要内核支持
- 由内核完成线程调度
- 由内核进行创建和撤销

■ 例子

- Windows XP/2000
- Solaris
- Linux
- Tru64 UNIX
- Mac OS X





多线程模型

- 多对一模型
- 一对一模型
- 多对多模型



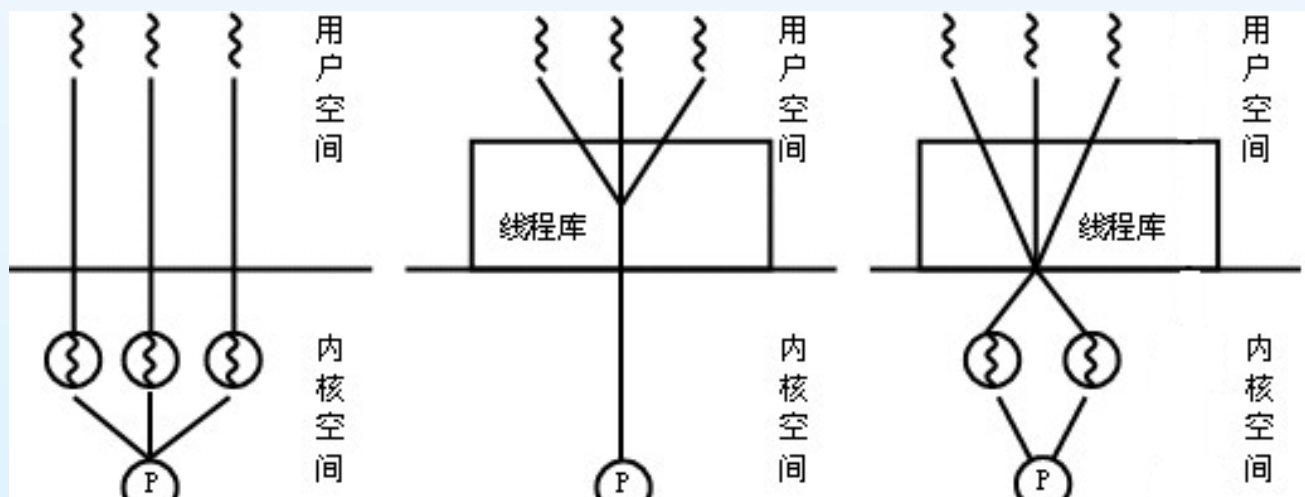


多线程模型

多对一模型

一对一模型

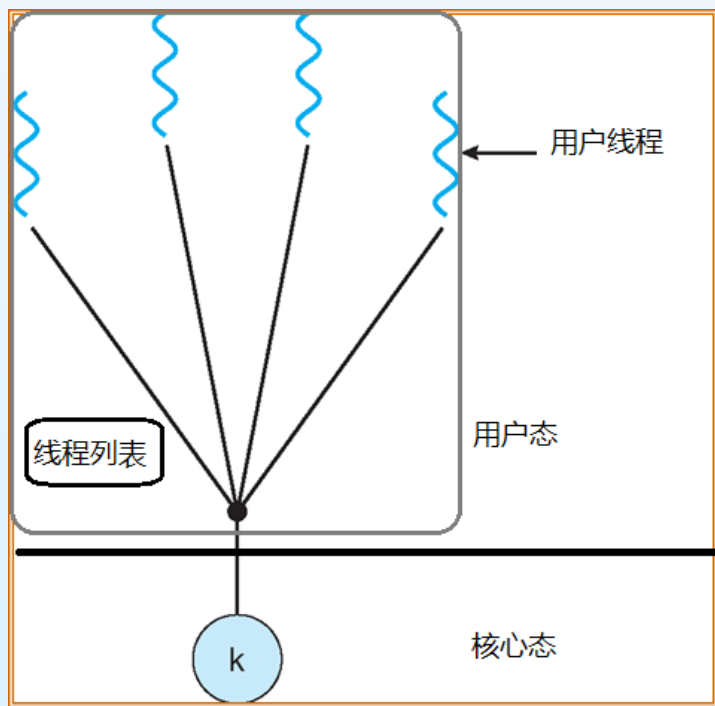
多对多模型





多对一模型

- 不支持内核线程的操作系统
 - 内核只有进程
- 内核只看到一个进程
 - 多个线程不能并行运行在多个处理器上（缺点）
- 进程中的用户线程由进程自己管理
 - 进程内线程切换不会导致进程切换（优点）
 - 一个线程的系统调用会导致整个进程阻塞
- 例子：
 - Solaris Green Threads
 - GNU Portable Threads





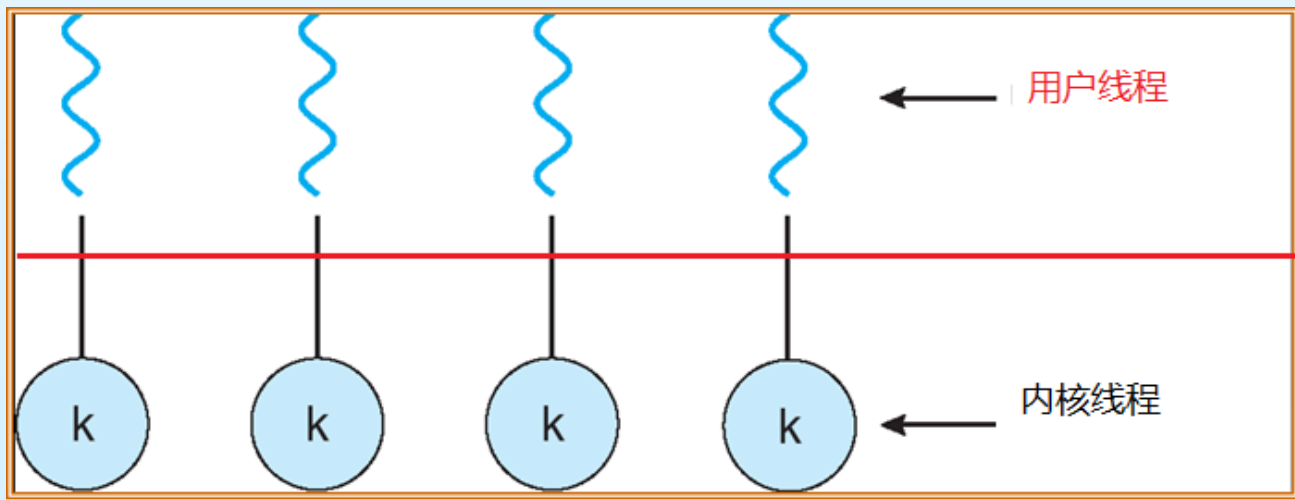
- 判断：多对一模型中内核只能看到一个进程，看不到进程中的线程。





一对一模型

- 用于支持线程的操作系统
 - 用户线程映射到内核线程
 - 操作系统管理这些线程
- 并发性好：多个线程可并行运行在多个处理器上
- 内核开销大
- 例子
 - Windows
 - OS/2





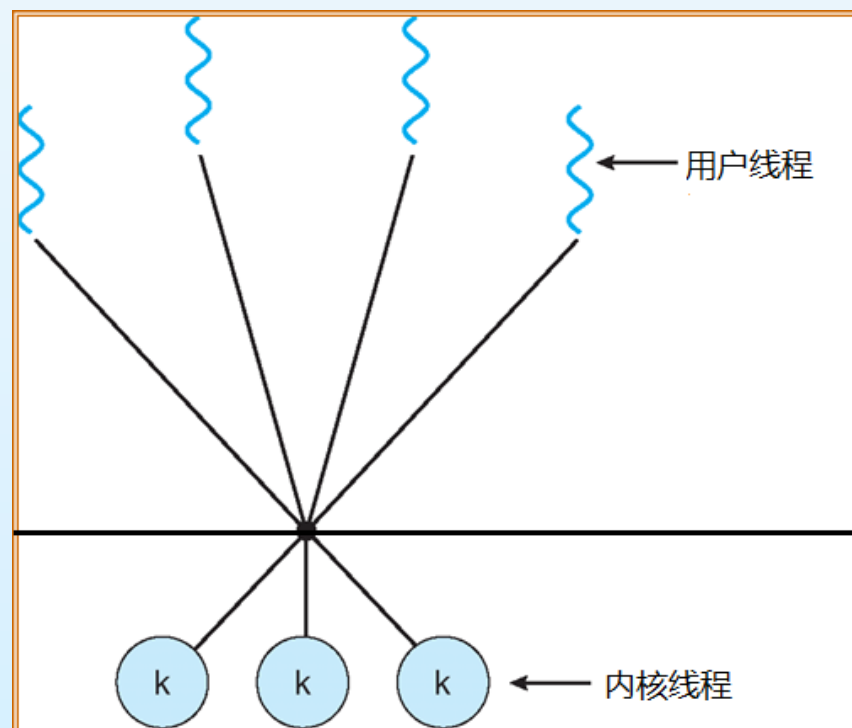
多对多模型

■ 多个用户线程映射为相等或更小数目的内核线程

- 并发性和效率兼顾
- 增加复杂度

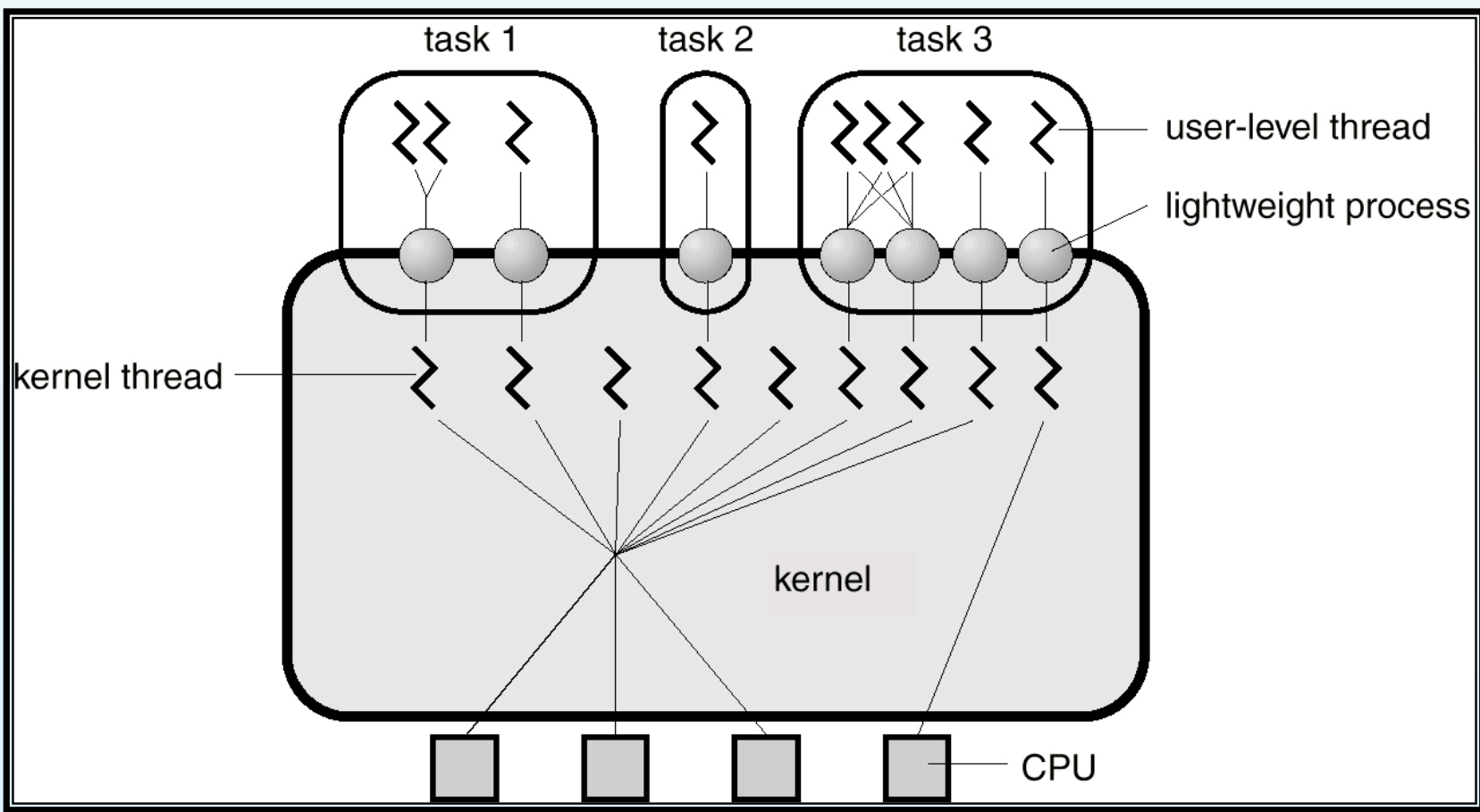
■ 例子

- Solaris 9 以前的版本





Solaris 2 多对多模型





两级模型

- 类似于 **M:M**, 只不过它允许一个用户线程绑定到内核线程
- 例子
 - IRIX
 - HP-UX
 - Tru64 UNIX
 - Solaris 8 and earlier

