用户级线程、核心级线程、混合方式

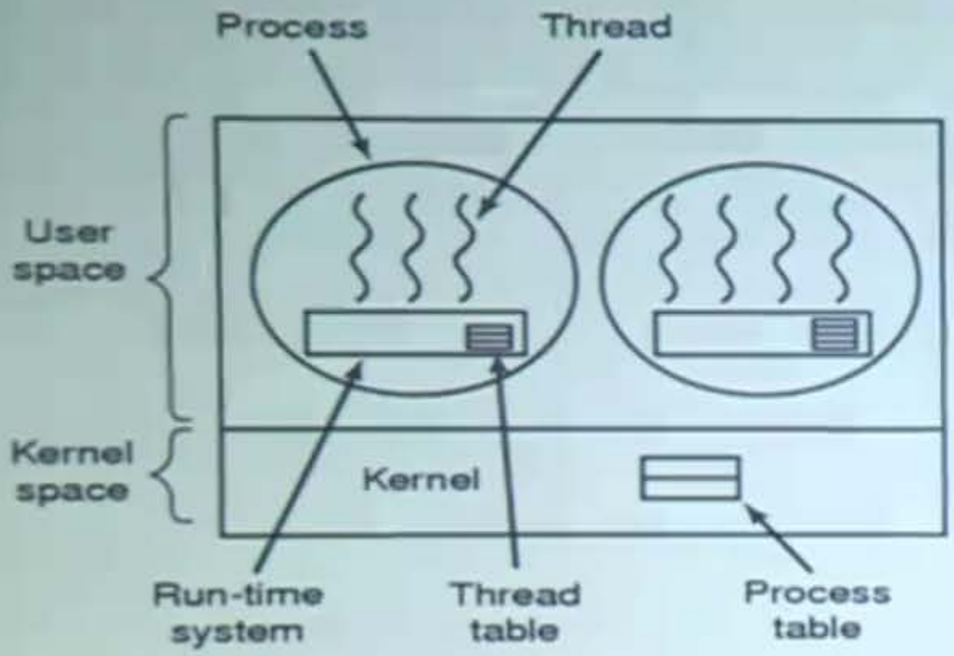
线程机制的实现

线程的实现

- 用户级线程
- 核心级线程
- 混合—两者结合方法

因此当有一个新的机制提出来的时候， 不同的操作系统对这个机制的支持是不一样的。

1.用户级线程（USER LEVEL THREAD）



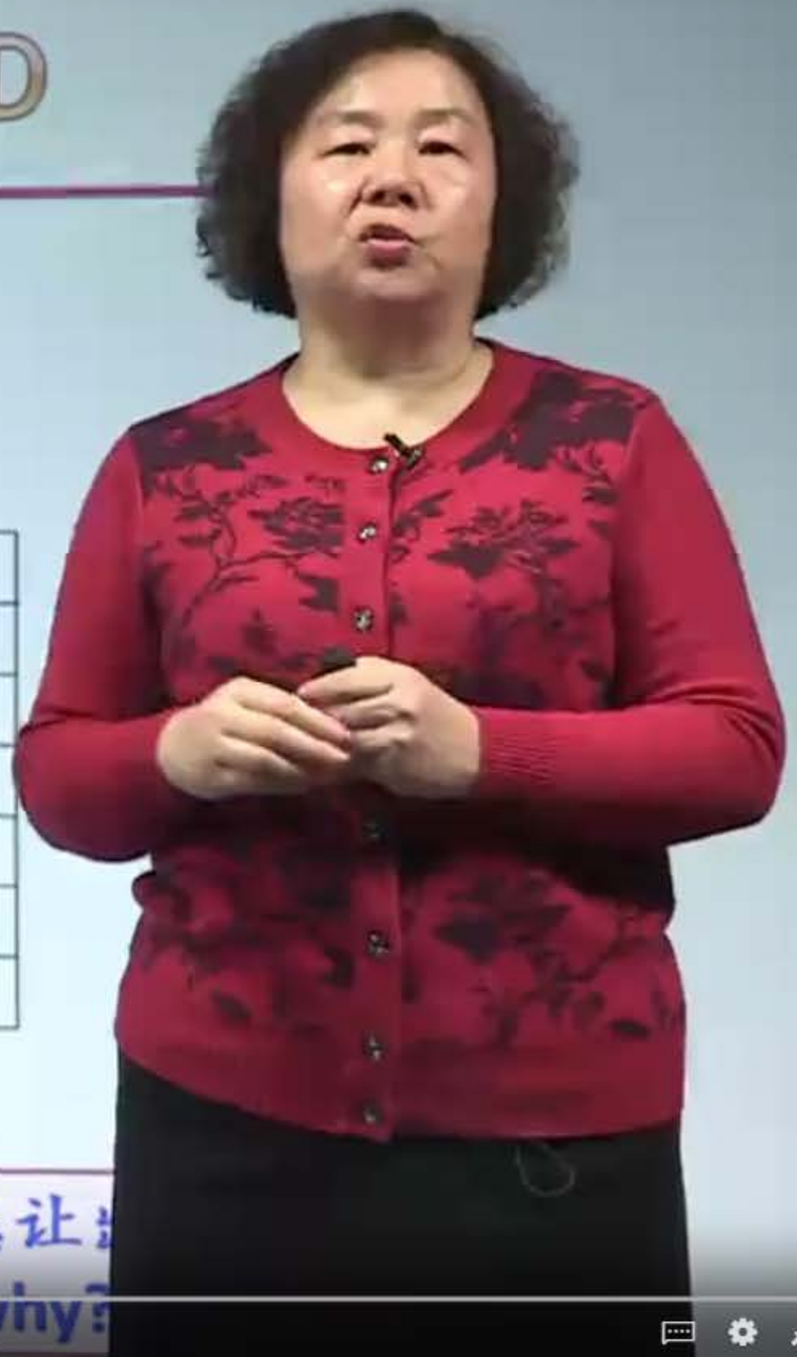
- 在用户空间建立线程库：提供一组管理线程的过程
- 运行时系统：完成线程的管理工作（操作、线程表）
- 内核管理的还是进程，不知道线程的存在
- 线程切换不需要内核态特权
- 例子：**UNIX**

例子：POSIX线程库——PTHREAD

- POSIX(Portable Operating System Interface)
- 多线程编程接口，以线程库方式提供给用户

Thread call	Description
Pthread_create	Create a new thread
Pthread_exit	Terminate the calling thread
Pthread_join	Wait for a specific thread to exit
Pthread_yield	Release the CPU to let another thread run
Pthread_attr_init	Create and initialize a thread's attribute structure
Pthread_attr_destroy	Remove a thread's attribute structure

线程是自愿让出
CPU的，why?



用户级线程小结

Jacketing/
wrapper

优点：

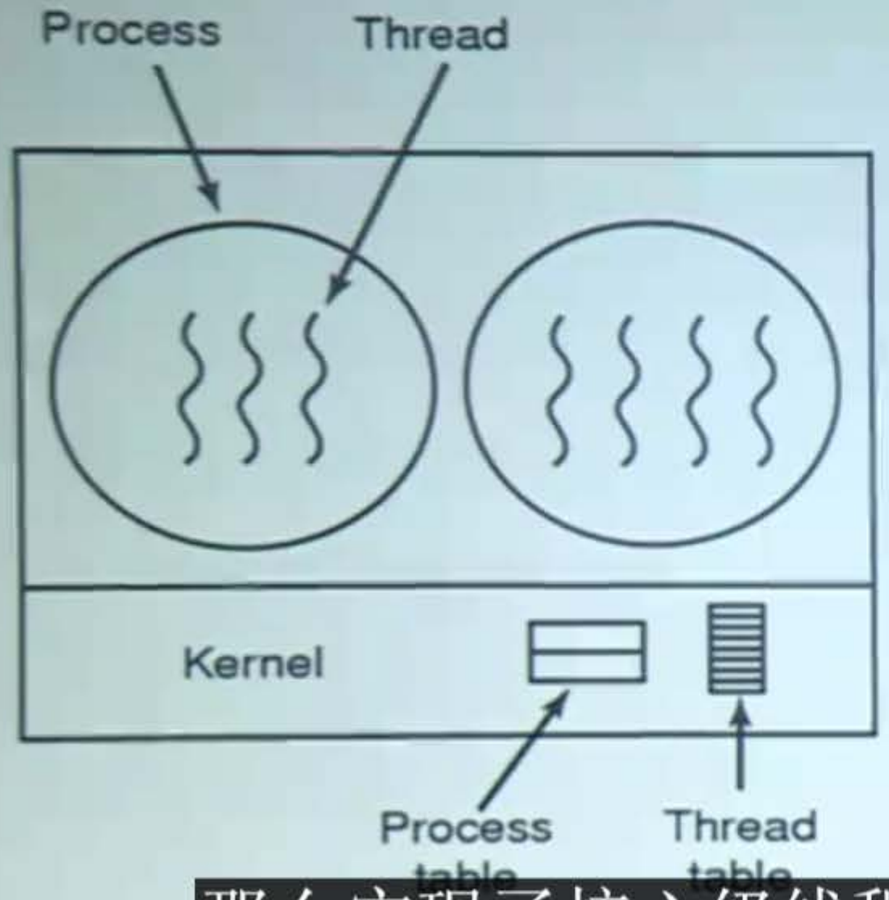
- 线程切换快
- 调度算法是应用程序特定的
- 用户级线程可运行在任何操作系统上（只需要实现线程库）

缺点：

- 内核只将处理器分配给进程，同一进程中的两个线程不能同时运行于两个处理器上
- 大多数系统调用是阻塞的，因此，由于内核阻塞进程，故进程中所有线程也被阻塞

线程调用了一个阻塞的系统调用，而被，使得整个进程被阻塞。

2.核心级线程（KERNEL LEVEL THREAD）

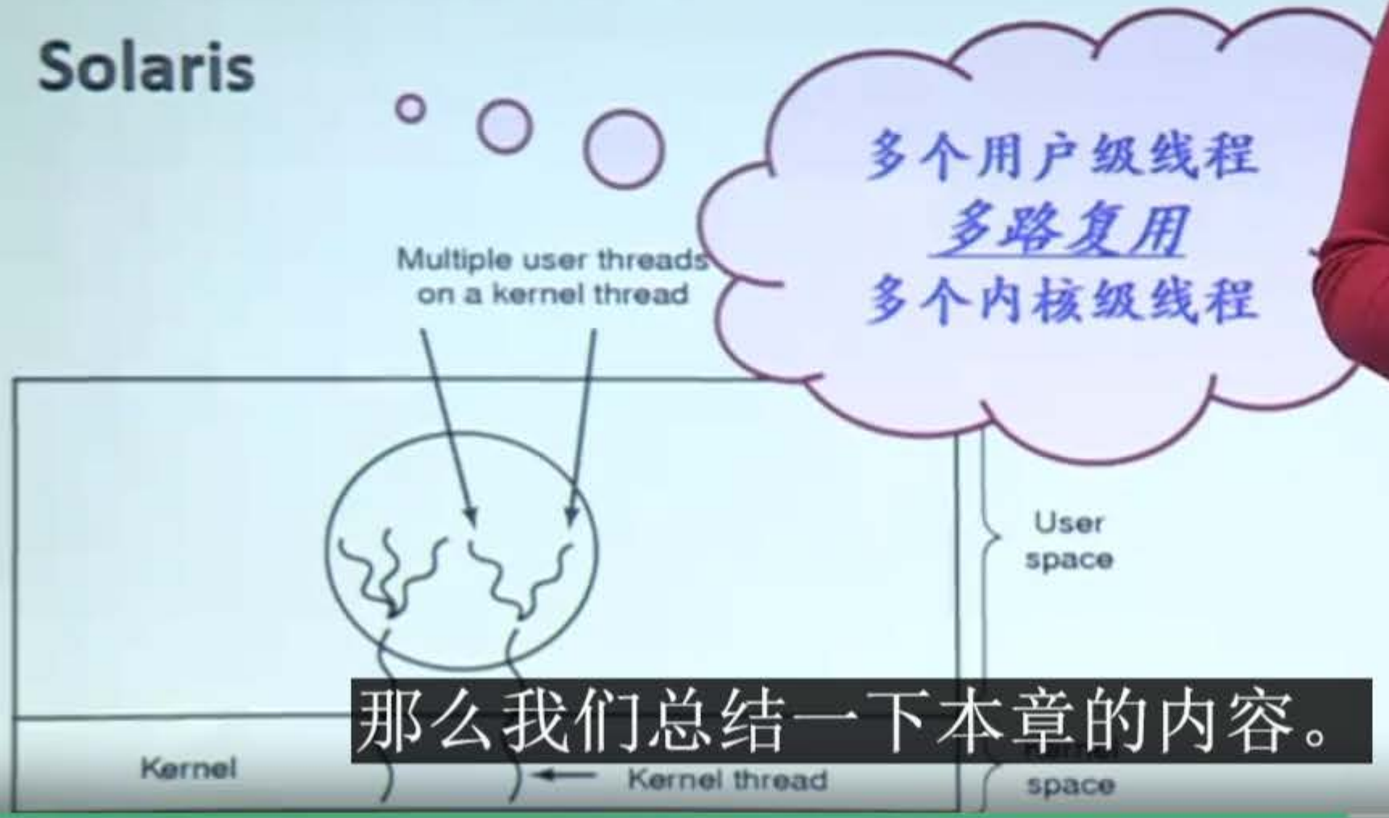


- 内核管理所有线程管理，并向应用程序提供API接口
- 内核维护进程和线程上下文
- 线程的切换需要内核支持
- 以线程为基础进行调度
- 例子：Windows

那么实现了核心级线程机制的典型的操作系统就是 Windows。

3.混合模型

- 线程创建在用户空间完成
- 线程调度等在核心态完成
- 例子：Solaris



那么我们总结一下本章的内容。

本章重点小结(1/2)

进程

- 并发性 任何进程都可以与其他进程一起向前推进
- 动态性 进程是正在执行程序的实例
  - ✓ 进程是动态产生，动态消亡的
  - ✓ 进程在其生命周期内，在三种基本状态之间转换
- 独立性 进程是资源分配的一个独立单位
  例如：各进程的地址空间相互独立
- 交互性 指进程在执行过程中可能与其他进程产生直接或间接的关系
- 异步性 每个进程都以其相对独立的、不可预知的速度向前推进
- 进程映像 程序 + 数据 + 栈(用户栈、内核栈) + PCB

一些数据结构，通过这些我们来界定这是一个进程的相关信息。

本章重点小结(2/2)

- 线程
  - 多线程应用场景
  - 线程基本概念、属性
  - 线程实现机制

可再入程序（可重入）：
可被多个进程同时调用的程序，具有下列性质
它是纯代码的，即在执行过程中自身不改变；
调用它的进程应该提供数据区

本周呢要求大家去阅读第二章的相关内容，

本周要求

- 重点阅读教材

  第2章相关内容：2.1、2.2(除2.2.8-2.2.10外)

- 重点概念

进程　进程状态及状态转换　进程控制
进程控制块(PCB)　进程地址空间　进程上下文环
线程　线程属性　用户级线程　核心级线程
Pthreads　可再入程序　原语　Web服务器

那么作为重点的概念我们列举在这里，希望大家能够掌握这些重要的概念。

0:00 下面我们来看一下，在一个操作系统当中， 如何来支持线程机制的实现。 通常有三种方式， 用户级线程、 核心级线程和混合方式。 那么由于在线程的概念提出之前，操作系统已经运行了很多年，那么进程的概念已经 用了很长时间。 因此当有一个新的机制提出来的时候， 不同的操作系统对这个机制的支持是不一样的。 那我们先看第一种实现： 用户级线程。 用户级线程的话呢， 顾名思义，是在用户空间建了一个线程库， 这个线程库里提供了一系列的针对线程的操作。 这些线程的管理 是通过一个 Run-time System 运行时系统来管理的。 那么它完成的就是这些线程的创建、 还有线程数据结构的一些管理工作。 我们来看一下，在图中 有一个 Run-time System，它呢，是管理这些线程 这里头有线程的数据结构、 线程表。 这是 用户级线程的一个实现。 那么对于内核而言， 由于你的线程的实现是在用户空间，所以操作系统 内核并不知道线程的存在，也就是说 它的管理还是以进程为单位来管理，它没有感知 线程的存在。 我们从图中可以看到，线程的数据结构是由 Run-time System 来管理的。 内核儿只看到了进程的数据结构， 因此线程的切换，从一个线程换到另外一个线程不需要 操作系统内核的干预，也不需要进入内核来做这件事情，比较快。 那么 UNIX 内的操作系统实际上呢通常采用这种方式来支持线程。 我们来看一下它支持线程的时候呢，是遵循了 POSIX 规范。 也就是 POSIX 规范当中确定了多线程的这种编程的接口。 以什么样的方式呈现给用户，它对线程库进行了相应的规范。 那么这个规范呢实际上就是 PTHREAD 线程库。 这个线程库按照规范要提供若干个函数来支持线程、 创建线程、 撤销线程，等待某个线程的结束。 在这堆操作当中，我们重点介绍一下 yield 这样一个函数，那么这个函数表示这个线程自愿让出 CPU。 因为我们知道一个进程的若干线程实际上是相互配合来完成一项任务的。 所以这线程之间是可以协商由谁上 CPU，所以 一个线程如果占 CPU 时间太长，那么别的线程得不到机会，就需要这个线程 高尚一点，让出 CPU，它就调用 yield 让出 CPU。 如果它不让出 CPU，其实其它线程是没法 上 CPU 的，因为对于线程而言，它 感知不到时钟中断，因为整个时钟段是对进程而言的。 那么我们总结一下用户级线程。 它的优点是切换很快，速度快。 调度算法是应用程序特定的，可以按照应用程序的需求来 切换不同的线程来执行。 用户线程可以运行在任何 操作系统之上，那么只需要在这个操作系统之上来支持相应的线程库。 但是它的缺点也很明显，因为内核是把 CPU 分配给进程，而不是线程。 所以，同一个进程如果有多个 线程，实际上即便是有多个 CPU，它也不能够运行在不同的 CPU 上。 这就是它的

一个缺点。 那么第二个缺点呢，因为在这个过程当中， 线程会调用一些系统调用。 而这个系统调用如果是 阻塞的系统调用的话，实际上对于内核而言，是把整个进程阻塞起来。 因此，可能是某个线程调用了系统调用，对于 内核而言，它只看到了进程，所以它把这个进程阻塞起来。 那么 其它的线程得不到机会去运行。 怎么样来改变这样一个 状况呢？可以有不同的方法。 虽然都不很完美，但是可以尝试一下。 比如说 我把系统调用，阻塞系统调用该成一个非阻塞的行不行啊？ 或者是用 Jacketing/ wrapper 的这种技术 在系统调用之外包装一层，封装一层。 在调用系统调用之前，先判断一下 调用这个系统调用会不会导致线程阻塞，如果导致线程阻塞， 那么就赶紧地换其它线程，这样的话，就不会因为某个 线程调用了一个阻塞的系统调用，而被，使得整个进程被阻塞。 第二类实现 线程机制的方法呢我们称之为核心级线程。 当然这个方案就是彻底地改造了操作系统。 也就是内核管理所有的线程。 通过 API 的接口向用户提供一些 API 的函数，由用户可以创建线程。 所以内核既维护了进程的数据结构，也维护 了进程里头的各个线程的数据结构，我们从图中可以看到，内核里头既管了线程表， 也管了进程表。 线程的切换 就需要内核干预了，因此要进入内核来完成切换的过程。 调度也是以线程为单位来进行的。 那么实现了核心级线程机制的典型的操作系统就是 Windows。 另外一种叫混合模型。 所谓混合模型就是 线程的创建呢是在用户空间用线程库来完成的。 但是内核儿呢，也要管理线程，也就是说调度是由内核来完成的。 那么这个采用这种混合模型实现线程机制的呢是 Solaris 操作系统。 那么用户空间的线程和内核的这个关系是什么呢？实际上呢是 用户线程通过了一个多路复用来 复用多个内核级线程，也就是 核外的用户空间的线程通过 一个机制和核内的一个内核线程对应起来。 那么调度内核这个线程上 CPU 其实就是调度 这个核外的这个线程上 CPU。 这是 Solaris 的一种实现。 那么我们总结一下本章的内容。 本章有两部分内容，一个是进程， 进程呢具有以下几个特性。 并发性，任何进程 都可以与其他进程一起向前推进，并发执行。 动态性，进程是正在执行的程序的实例。 它是动态产生、动态消亡，有生命周期，而且在它的生命周期期间它的状态 还会不断地变化。 进程呢 又具有独立性，因为它是资源分配的一个独立单位。 每个进程的地址空间是相互独立的，互不干扰的。 进程呢也具有交互性，也就是进程 不同的进程可能会产生直接的或间接的一些关系。 这个呢是我们后面的一个主题。 进程 是异步的，也就是每个进程都以其相

对独立的、 不可预知的速度向前推进。 那么进程从静态的角度来看，那么我们可以得到一个进程的映像。 包括了代码、 数据、 堆栈的内容，当然这里头用户栈和内核栈都在里头了，还有相关的 一些数据结构，通过这些我们来界定这是一个进程的相关信息。 另外，第二个主题是线程。 那么关于线程呢我们首先要了解什么情况下去应用多线程。 线程作为一个进程中运行的一个实体， 有哪些是属于它自己的信息？有哪些 是共享同一个进程的其他的一些资源？ 在一个操作系统中如何来实现线程？ 如何来支持线程？有几种典型的方式。 最后，我们来看一下一个非常重要的概念。 叫可再入程序，或者叫可重入程序。 所谓可再入程序呢，指的是可以被多个 进程同时调用的程序，因此对这个程序有限制。 也就是它必须具有的性质是它是纯代码的， 在执行过程中这个代码不会改变。 那么如果有改变，就需要调用它的进程提供不同的数据区。 这些改变可以放在数据区，因为代码部分是不再改变的。 那么可再入程序实际上是我们大部分 进程和线程都必须是可再入程序才能去运行。 本周呢要求大家去阅读第二章的相关内容，2.1，2.2，那么2.2.8-2.2.10 这部分内容呢可以不去读。 那么作为重点的概念我们列举在这里，希望大家能够掌握这些重要的概念。 本讲的内容呢就介绍到这里， 谢谢大家！