西安交通大学
软件学院

操作系统原理

Operating System Principle

田丽华

4-1 线程概念

The introduction of threads

线程的引入

进程具有二个基本属性：

- 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。
- 这二个基本属性使进程成为并发执行的基本单位
- 在一些早期的OS中，比如大多数UNIX系统、Linux等，进程同时具有这二个属性。

由于进程是一个资源的拥有者，因而在进程创建、撤销、调度切换时，系统需要付出较大的时空开销。

进程的数目不宜过多，进程切换频率不宜过高，限制了并发程度。

The introduction of threads

线程的引入

操作系统的设计目标

- 提高并发度
- 减小系统开销

将进程的两个基本属性分开

对于拥有资源的基本单位，不对其进行频繁切换，对于调度的基本单位，不作为拥有资源的单位，“轻装上阵”

引入线程以小的开销来提高进程内的并发程度。

The introduction of threads

线程的引入

进程：

资源分配单位（存储器、文件）和CPU调度（分派）单位。又称为"任务(task)"

线程：

作为CPU调度单位，而进程只作为其他资源分配单位。

- 只拥有必不可少的资源，如：线程状态、程序计数器、寄存器上下文和栈
- 同样具有就绪、阻塞和执行三种基本状态
- 与同属一个进程的其它线程共享进程拥有的全部资源
- 可并发执行

The introduction of threads

线程的引入

线程的优点：

减小并发执行的时间和空间开销（线程的创建、退出和调度），因此容许在系统中建立更多的线程来提高并发程度。

- 线程的创建时间比进程短；
- 线程的终止时间比进程短；
- 同进程内的线程切换时间比进程短；
- 由于同进程内线程间共享内存和文件资源，可直接进行不通过内核的通信；

Threads
线程

- A *thread* (or *lightweight process*) is a basic unit of CPU utilization; it consists of:

  线程（轻型进程）是CPU运用的一个基本单元，包括
  - program counter 程序计数器
  - register set 寄存器集
  - stack space 栈空间
- A thread shares with its peer threads its:

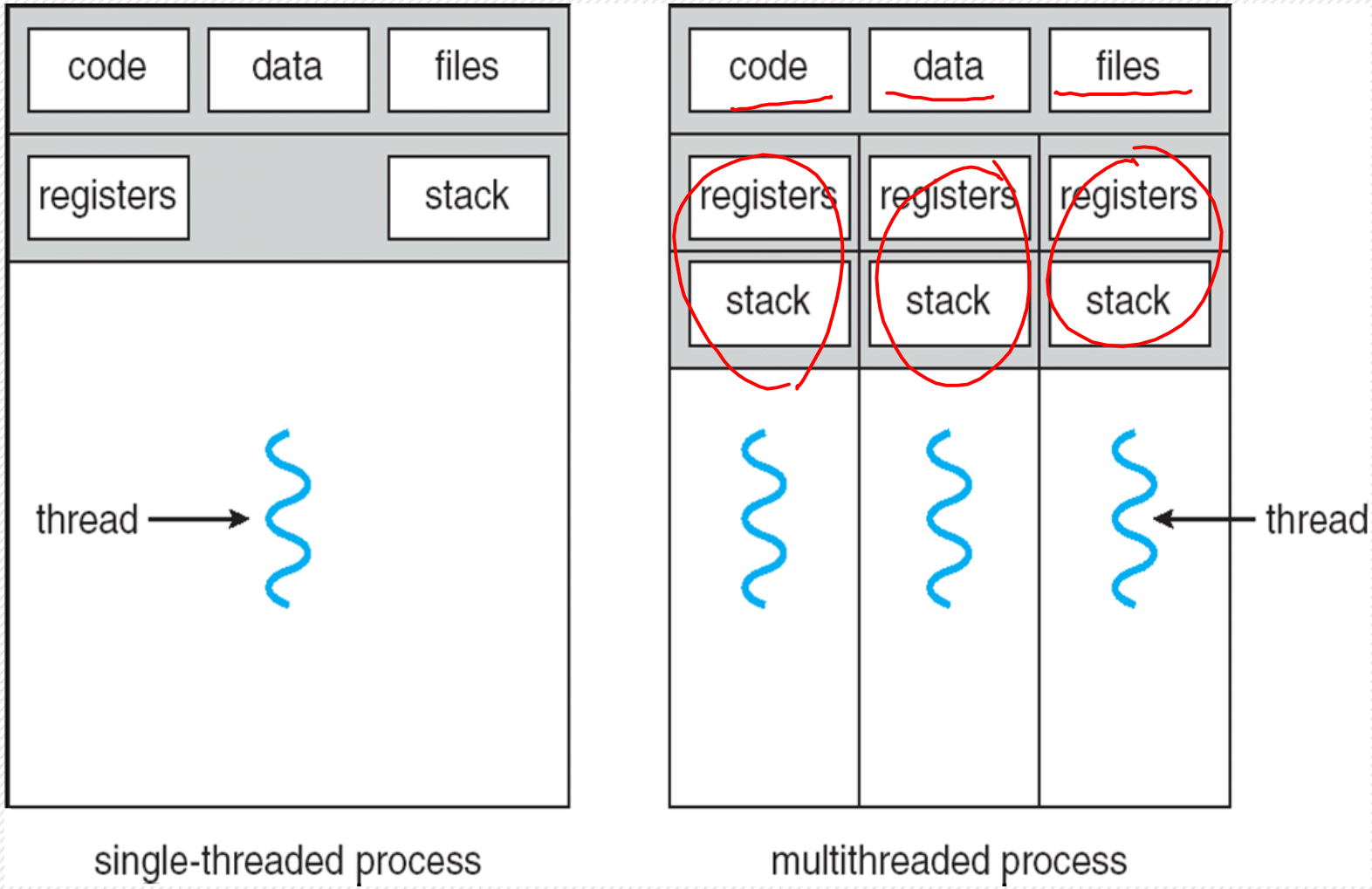
  一个线程与它的对等线程共享：
  - code section 代码段
  - data section 数据段
  - operating-system resources 操作系统资源

    such as open files and signals

  collectively know as a *task*. 总体作为一个任务
- A traditional or *heavyweight* process is equal to a task with one thread

  传统的或重型进程等价于只有一个线程的任务

Single and Multithreaded Processes



A comparison between a process and a thread

进程和线程的比较

- 并发性：在引入线程的OS中，不仅进程之间可以并发执行，而且在一个进程中的多个线程之间亦可并发执行，因而使OS具有更好的并发性，从而能更有效地使用系统资源和提高系统吞吐量。
- 拥有资源：进程是拥有资源的独立单位
- 系统开销：在创建或撤消进程时，系统都要为之分配或回收资源，如内存空间、I/O设备等。因此，OS所付出的开销将明显地大于在创建或撤消线程时的开销。

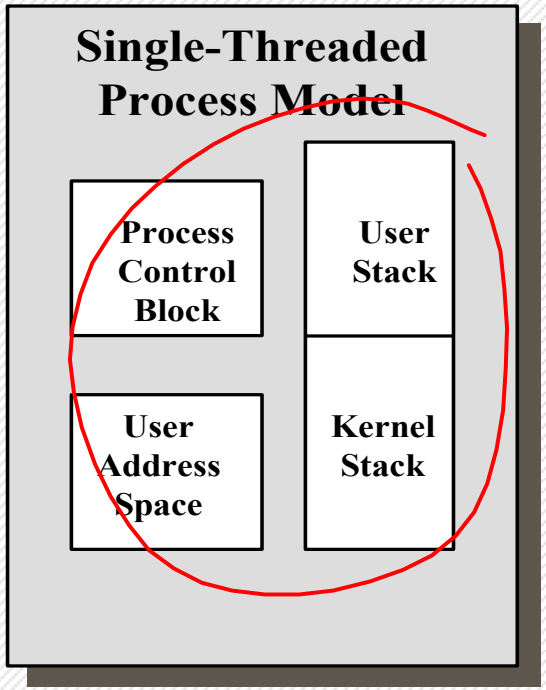
A comparison between a process and a thread

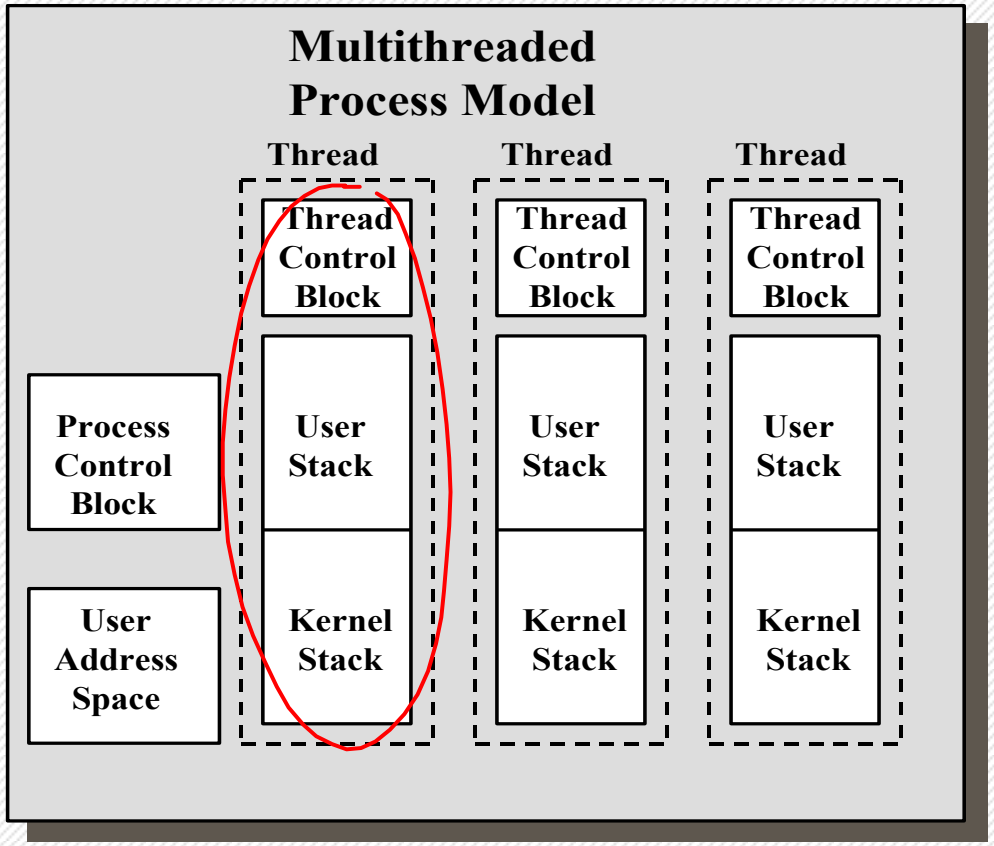
进程和线程的比较

- 地址空间和其他资源（如打开文件）：进程间相互独立，同一进程的各线程间共享——某进程内的线程在其他进程不可见
- 通信：进程间通信IPC，线程间可以直接读写进程数据段（如全局变量）来进行通信——需要进程同步和互斥手段的辅助，以保证数据的一致性
- 调度：线程上下文切换比进程上下文切换要快得多；

A comparison between a process and a thread

进程和线程的比较





线程切换和进程切换

Benefits
益处

Responsiveness

响应度高：一个多线程的应用在执行中，即使其中的某个线程阻塞，其他的线程还可继续执行，从而提高响应速度

Resource Sharing

资源共享：同一进程的多个线程共享该进程的内存等资源

Economy

经济性：创建和切换线程的开销要低于进程。比如，Solaris中进程创建时间是线程创建的30倍，进程切换时间是线程切换的5倍。

Utilization of MP Architectures

MP体系结构的运用：多线程更适用于多处理机结构。

西安交通大学
软件学院

操作系统原理

Operating System Principle

田丽华

4-2 用户线程和内核线程

Threads
线程

- Kernel-supported threads内核支持的线程 (Mach and OS/2).

- User-level threads; supported above the kernel, via a set of library calls at the user level (Project Andrew from CMU). 用户级线程；在内核之上，通过用户级的库调用

Kernel Threads
内核线程

Supported by the Kernel
由内核支持，在内核空间执行线程创建、调度和管理

Thread is unit of CPU scheduling

Examples例子

- Windows XP/2000
- Solaris
- Digital UNIX

kernel-level thread
内核线程

依赖于OS核心，由内核进行创建、撤销和切换

Windows NT和OS/2支持内核线程；

- 内核维护进程和线程的上下文信息；
- 线程切换由内核完成；
- 一个线程发起系统调用而阻塞，不会影响其他线程的运行。
- 时间片分配给线程，所以多线程的进程获得更多CPU时间。

User Threads
用户线程

- Thread Management Done by User-Level Threads Library

  由用户级线程库进行管理的线程

- 线程库提供对线程创建\调度和管理的支持，无需内核支持。

- Process is still unit of CPU scheduling from OS kernel perspective

- Examples例子

- POSIX Pthreads
- Mach C-threads
- Solaris threads

user-level thread
用户线程

不依赖于OS核心，应用进程利用线程库提供创建、同步、调度和管理线程的函数来控制用户线程。调度由应用软件内部进行，通常采用非抢先式和更简单的规则，也无需用户态/核心态切换，所以速度特别快。

- 用户线程的维护由应用进程完成；
- 内核不了解用户线程的存在；
- 用户线程切换不需要内核特权；
- **DRAWBACKS:**
  - 如果内核是单线程的,那么一个用户线程发起系统调用而阻塞，则整个进程阻塞。
  - 时间片分配给进程，多线程则每个线程就慢。

User threads and kernel threads

用户线程与内核线程

调度方式：

内核线程的调度和切换与进程的调度和切换十分相似，用户线程的调度不需OS的支持。

调度单位：

用户线程的调度以进程为单位进行，在采用时间片轮转调度算法时，每个进程分配相同的时间片。对内核级线程，每个线程分配时间片。

西安交通大学
软件学院

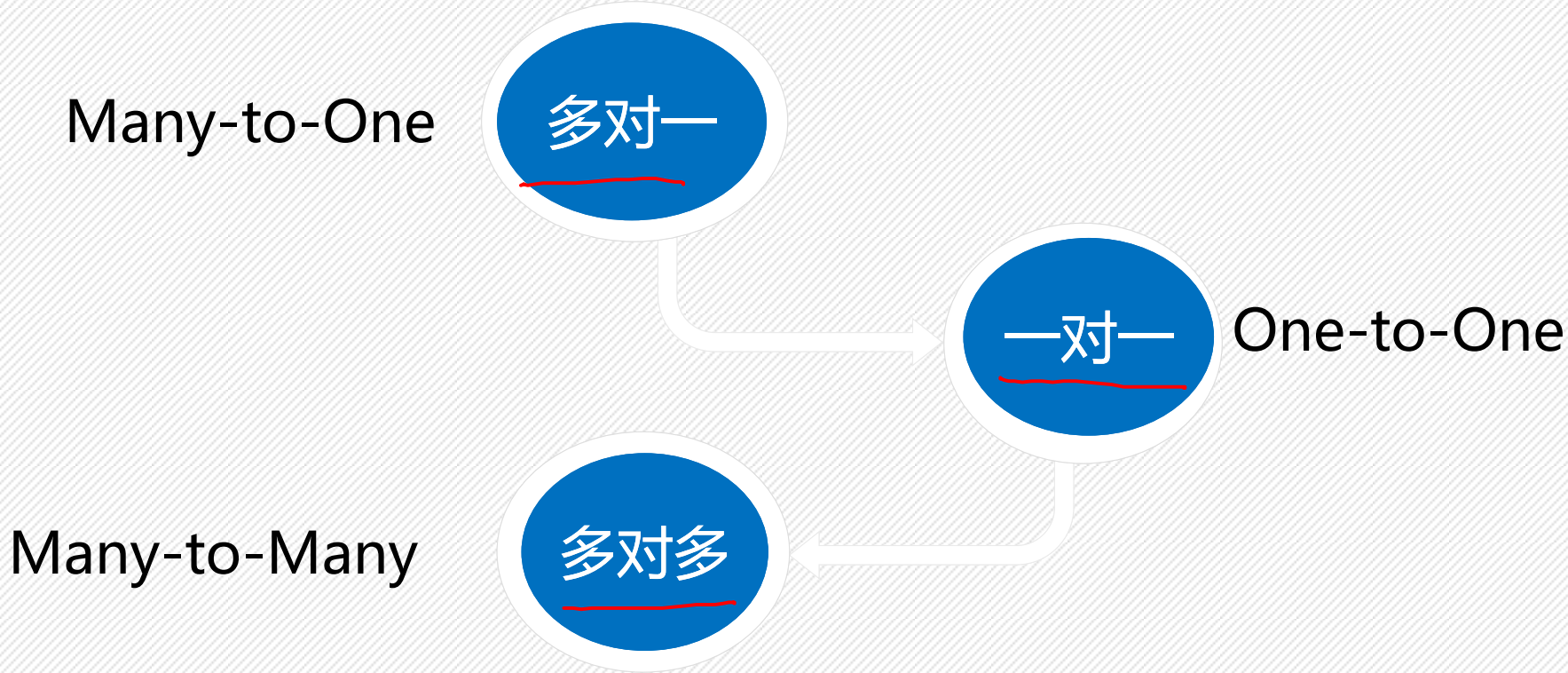
操作系统原理

Operating System Principle

田丽华

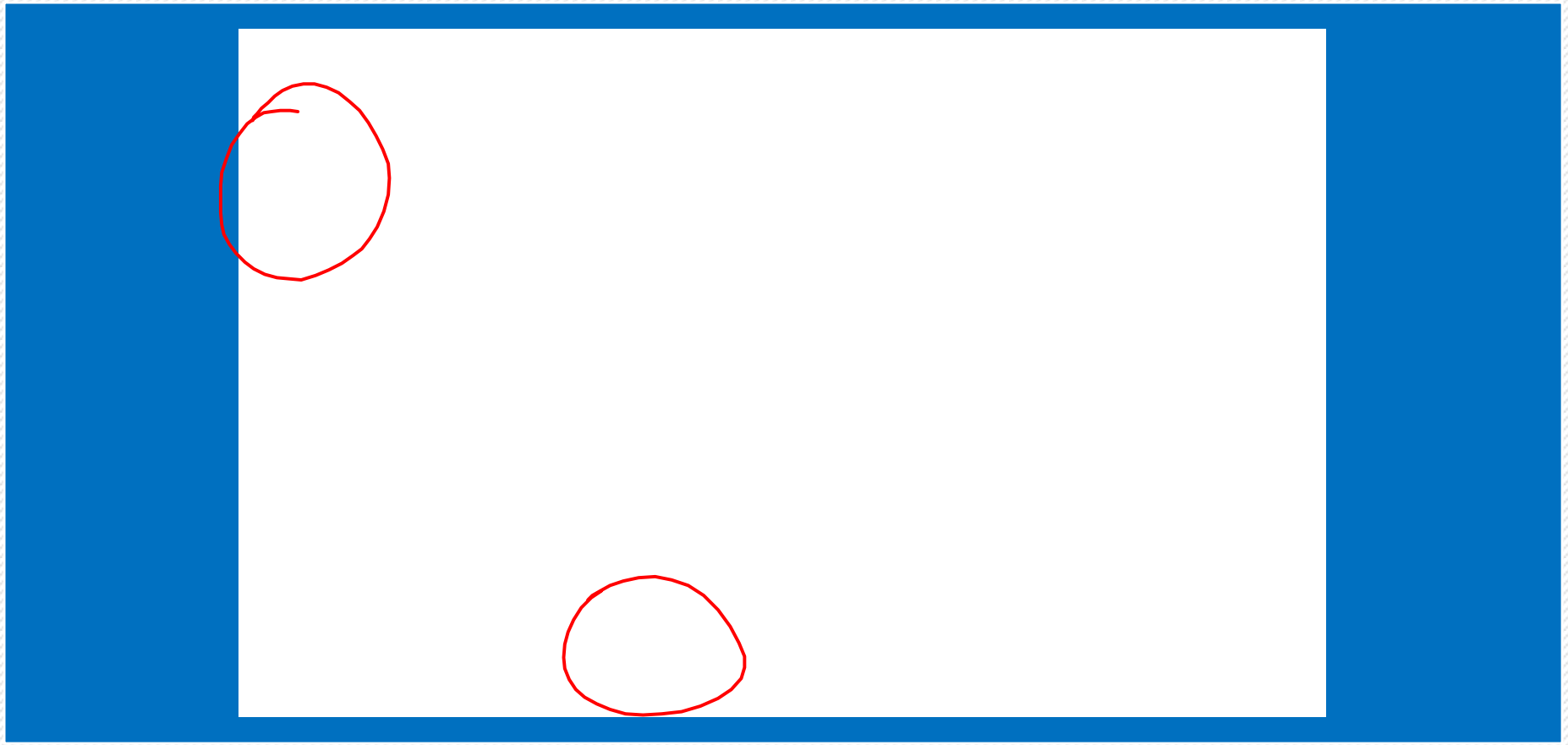
4-3 线程模型

Multithreading Models
多线程模型



Many-to-one Model

多对一模型



Many-to-One
多对一

Many User-Level Threads Mapped to Single Kernel Thread.
多个用户级线程映像进单个内核线程

Used on Systems That Do Not Support Kernel Threads.
用于不支持内核线程的系统中

任一时刻只能有一个线程可以访问内核(并发性低)

The entire process will block if a thread makes a blocking system call.
一个用户线程发起系统调用而阻塞，则整个进程阻塞。

One-to-one Model

一对一模型

One-to-One

一对一

Each User-Level Thread Maps to Kernel Thread.

每个用户级线程映像进内核线程

Allowing another thread to run when a thread makes a blocking system call

提供了更好的并发性,一个用户线程发起系统调用而阻塞时允许另一个线程运行

每创建一个用户级线程需创建一个相应的内核线程,带来了额外开销,所以许多系统限制应用中的线程数目

Many-to-many Model
多对多模型

多对一模型的缺点

不能实现真正的并发

一对一模型的缺点

需限制应用中的线程数目

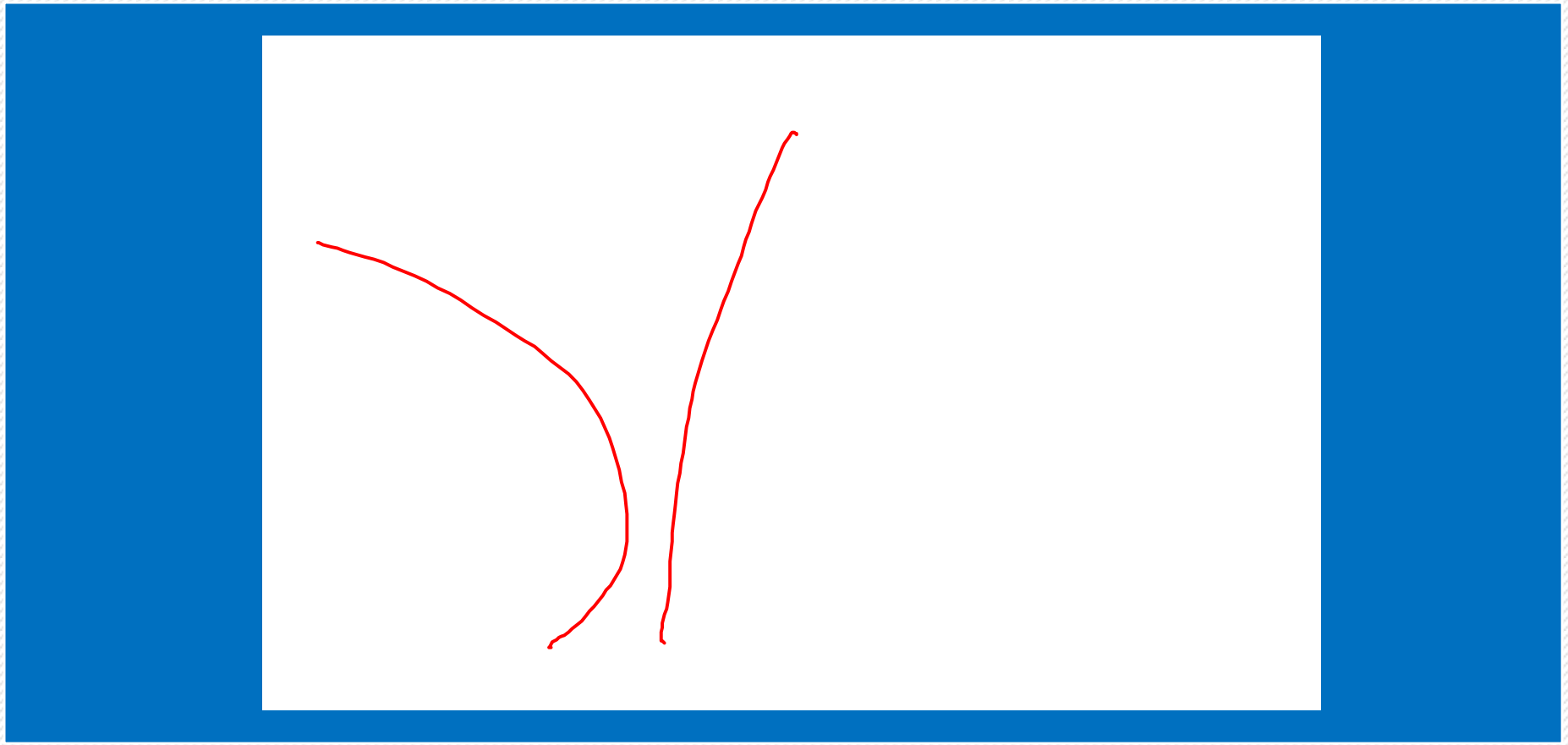
Many-to-many model multiplexes many user-level threads to a smaller or equal number of kernel threads.

多对多模型

不限制应用的线程数、多个线程可以并发

Many-to-many Model

多对多模型



Two-level Model
两级模型

Similar to M:M, except that it allows a user thread to be bound to kernel thread