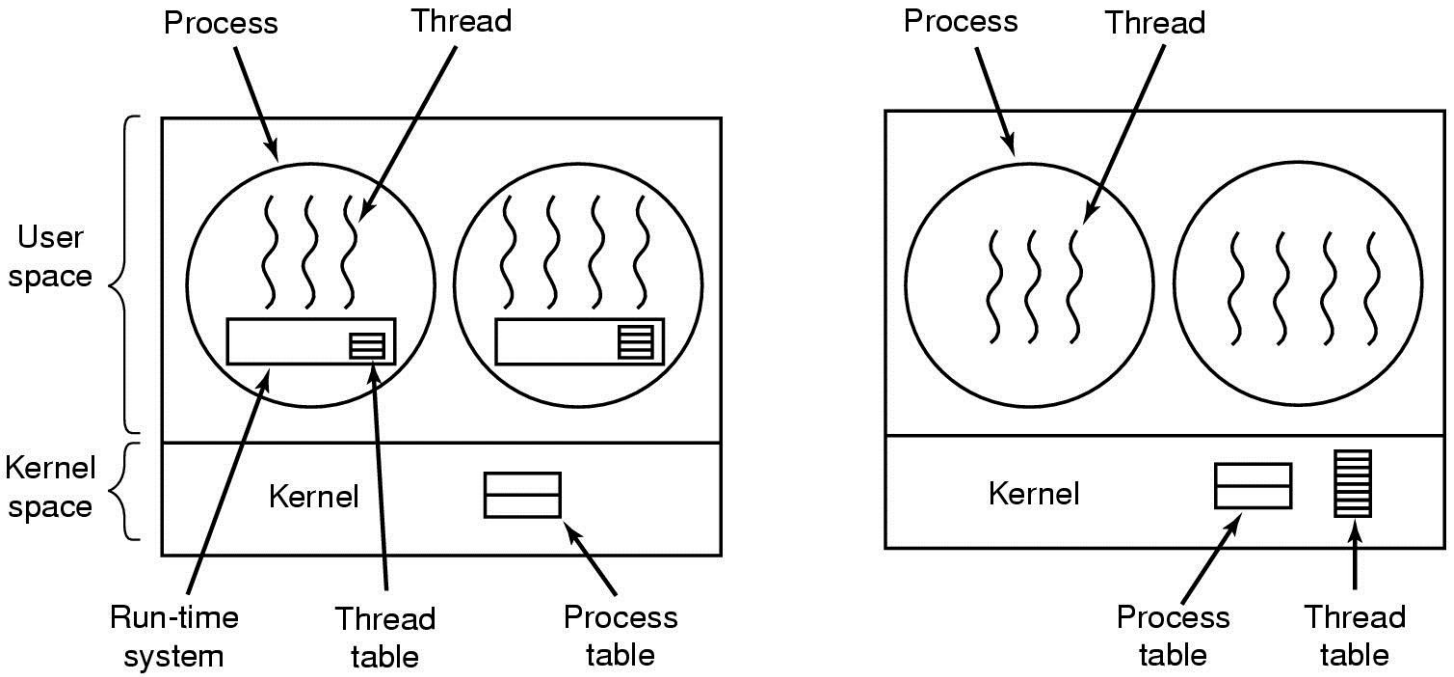
Implementing Threads in User Space



(a) A user-level threads package.

(b) A threads package managed by the kernel.

在用户空间中实现线程

内核对线程一无所知

运行时系统：管理线程的过程的集合

thread_create, thread_exit, thread_wait, thread_yield

每个进程有专用的线程表，跟踪进程中的线程

（与内核中的进程表类似，记录线程PC、堆栈指针、寄存器、状态）

线程表由运行时系统管理

线程切换的例程可参考教材

允许每个进程有自己定制的调度算法

在用户空间中实现线程

问题：如何实现阻塞调用，而不影响其他线程？（read读？）

1.系统调用改成非阻塞（失去用户级线程的优势）

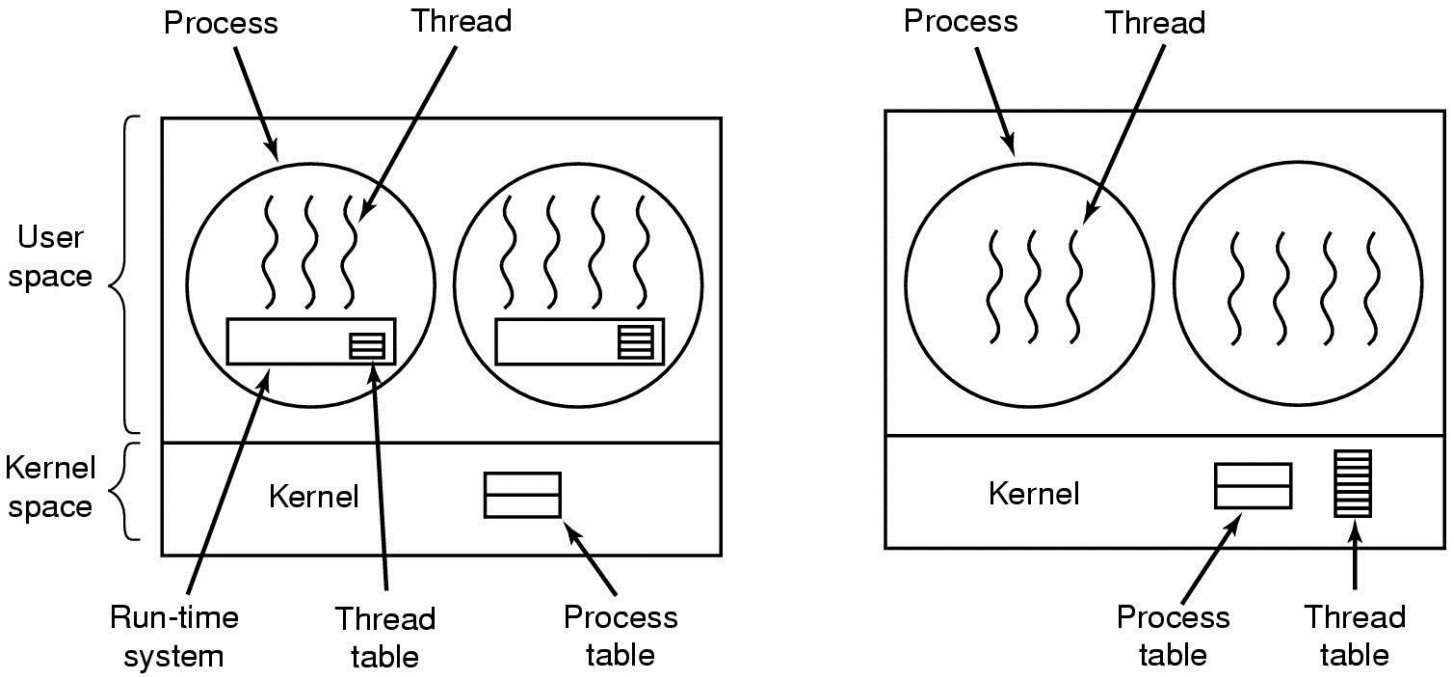
2.如果某个调用会阻塞，提前通知（select，例程见教材）

（包装器，jacket或wrapper）

问题：页面失效？（见教材）

问题：时钟中断？（见教材）

Implementing Threads in kernel Space



(a) A user-level threads package.

(b) A threads package managed by the kernel.

在内核空间中实现线程

内核了解和管理线程，不需要运行时系统和线程表

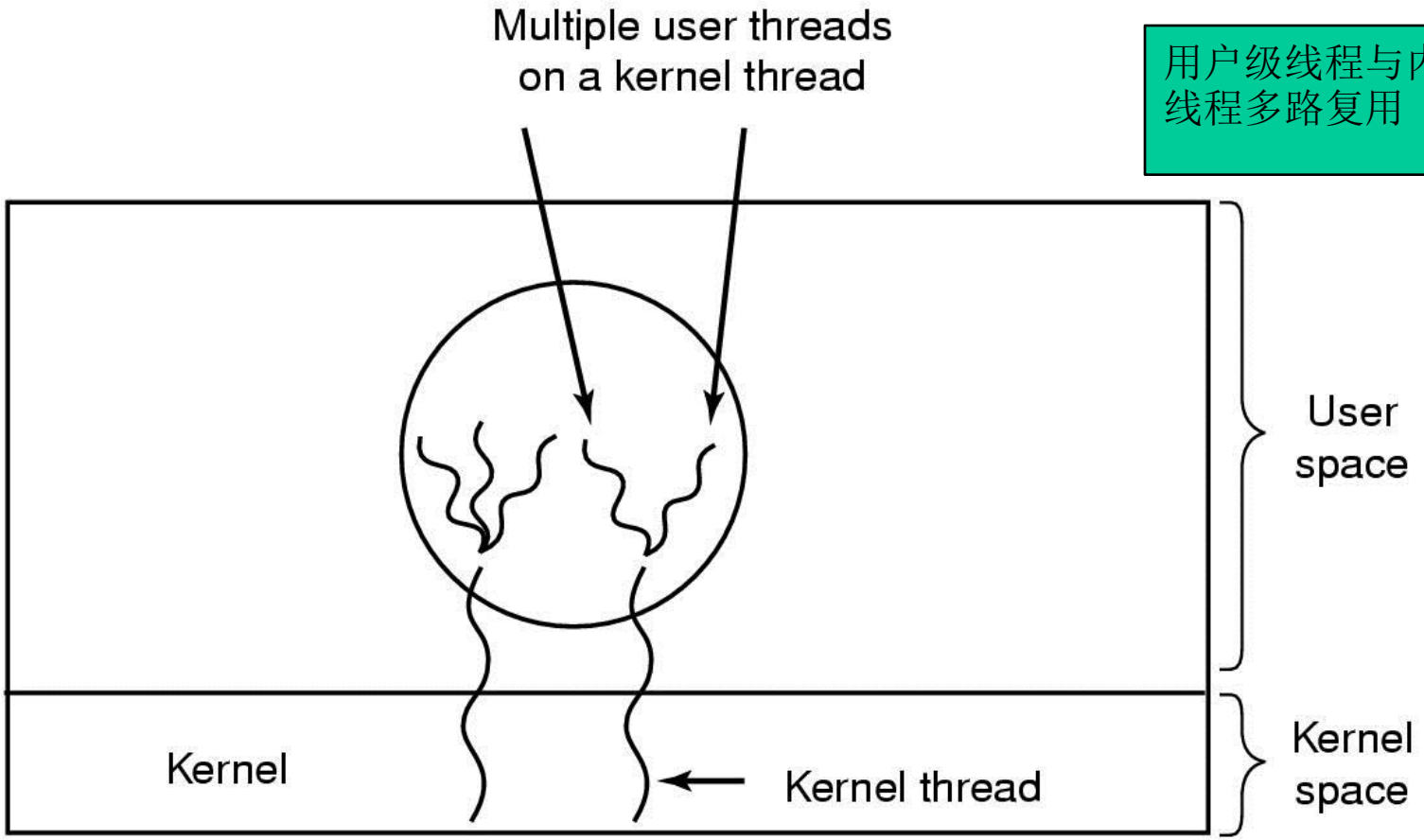
内核中有用来记录系统中所有线程的线程表

内核同时维护进程表，以便跟踪进程的状态

阻塞线程的调用都以系统调用，代价大（见教材）

内核线程不需要非阻塞调用，页面失效时，可以运行其它线程以此等待磁盘

Hybrid Implementations



Multiplexing user-level threads onto kernel-level threads.

Scheduler Activations

- Goal – mimic functionality of kernel threads
  - gain performance of user space threads
- Avoids unnecessary user/kernel transitions
- Kernel assigns virtual processors to each process
  - lets runtime system allocate threads to processors
- Problem:
  Fundamental reliance on kernel (lower layer) calling procedures in user space (higher layer)

调度程序激活机制

对用户级线程提供内核级别的支持

1.用户线程保持其高效

2.调度核心仍在用户空间

目标：如果某个线程由于等待另一个线程的工作而阻塞，此时无需请求内核，进而减少内核-用户转换的开销。

用户空间的运行时系统可以阻塞同步的线程而调度另一个新线程。

Pop-Up Threads

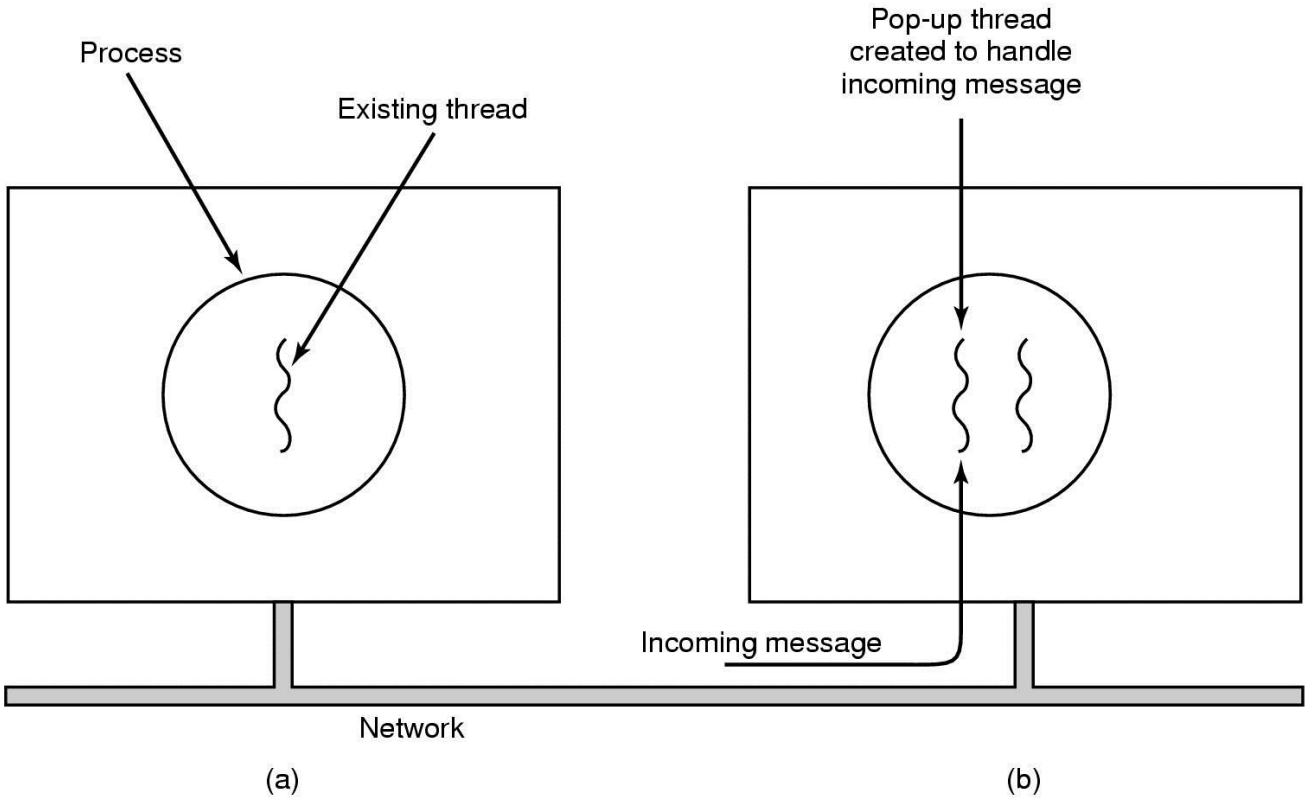


Figure 2-18. Creation of a new thread when a message arrives.

(a) Before the message arrives.

(b) After the message arrives.

Making Single-Threaded Code Multithreaded (1)

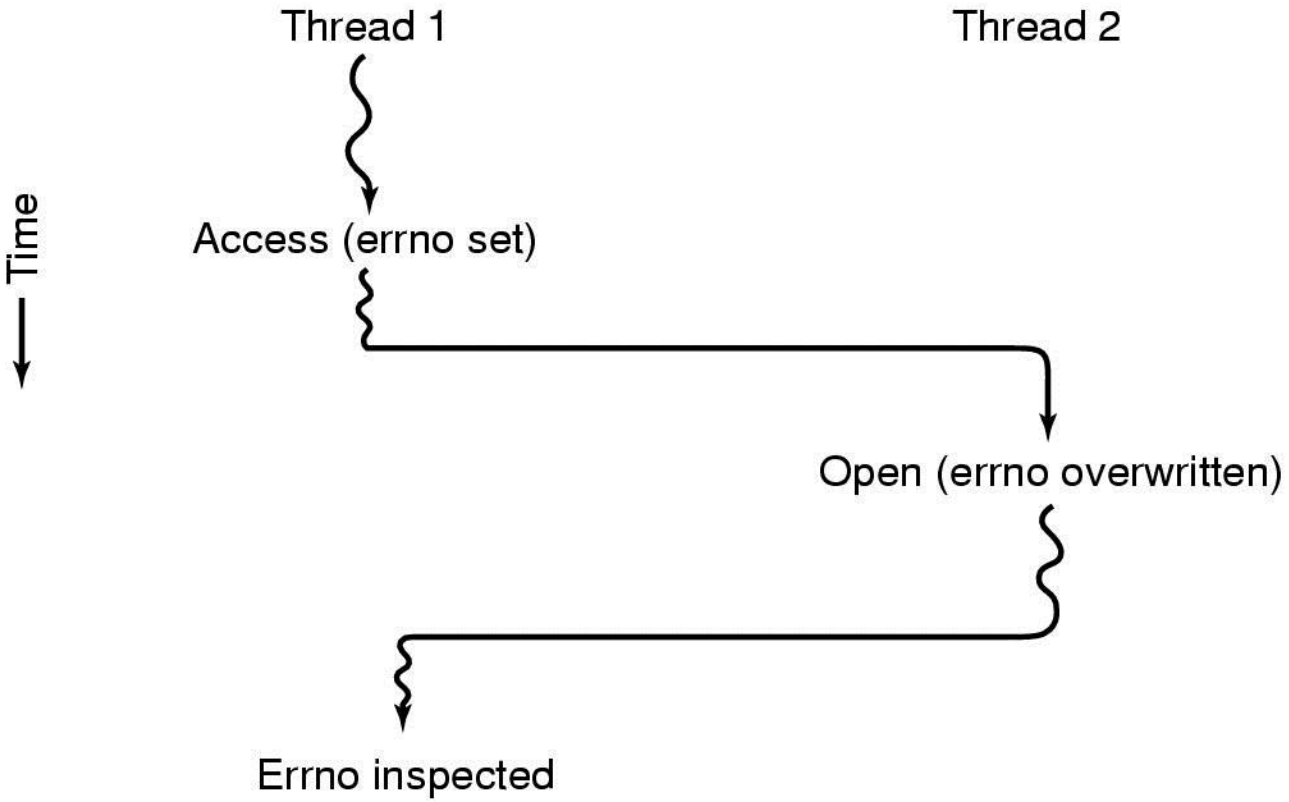


Figure 2-19. Conflicts between threads over the use of a global variable.

Making Single-Threaded Code Multithreaded (2)

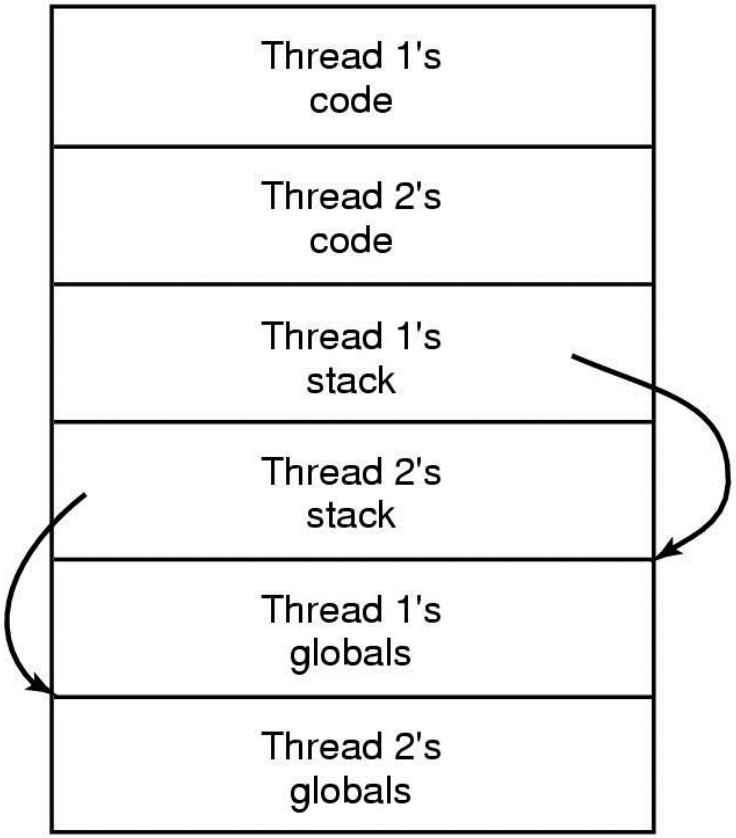


Figure 2-20. Threads can have private global variables.