Part 5 线程概念及实现

5.2 线程和进程的比较

01 调度的基本单位

- 在传统的OS中，拥有资源、独立调度和分派的基本单位都是进程；
- 在引入线程的OS中，线程作为调度和分派的基本单位，进程作为资源拥有的基本单位；
- 在同一进程中，线程的切换不会引起进程切换，在由一个进程中的线程切换到另一个进程中的线程时，将会引起进程切换。

02 并发性

- 在引入线程的操作系统中，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，也可并发执行。

5.2 线程和进程的比较

03 拥有资源

- 进程是系统中拥有资源的一个基本单位，它可以拥有资源。
- 线程本身不拥有系统资源，仅有一点保证独立运行的资源。
- 允许多个线程共享其隶属进程所拥有的资源。

04 独立性

- 同一进程中的不同线程之间的独立性要比不同进程之间的独立性低得多。

5.2 线程和进程的比较

05 系统开销

- 在创建或撤消进程时，OS所付出的开销将显著大于创建或撤消线程时的开销。
- 线程切换的代价远低于进程切换的代价。
- 同一进程中的多个线程之间的同步和通信也比进程的简单。

06 支持多处理机系统

5.3 线程状态

线程状态

- 执行态、就绪态、阻塞态
- 线程状态转换与进程状态转换一样

线程控制块（thread control block，TCB）

- 线程标识符、一组寄存器、线程运行状态、优先级、线程专有存储区、信号屏蔽、堆栈指针

5.4 线程实现

Linux线程——C

```
#include<stdio.h>
#include<pthread.h>
void task1(void);
void task2(void);
int sharedi=0;
int main(void) {
    pthread_t thrd1, thrd2;     int ret;
    ret = pthread_create(&thrd1, NULL, (void *)task1,
    NULL);
    ret = pthread_create(&thrd2, NULL, (void *)task2,
    NULL);
    pthread_join(thrd1, NULL);
    pthread_join(thrd2, NULL);
    printf("sharedi = %d\n", sharedi);
}
```

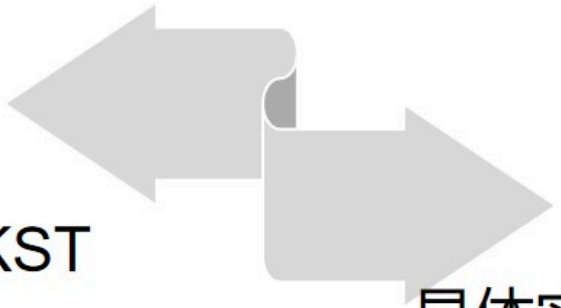

```
void task1(void)
{   long i,tmp;
    for(i=0; i<1000000; i++) {
        tmp = sharedi;
        tmp = tmp + 1;
        sharedi = tmp;
}}

void task2(void)
{   long i,tmp;
    for(i=0; i<1000000; i++) {
            tmp = sharedi;
            tmp = tmp + 1;
            sharedi = tmp;
    }}
```

5.4 线程实现

实现方式：

- 内核支持线程KST
- 用户级线程ULT
- 组合方式

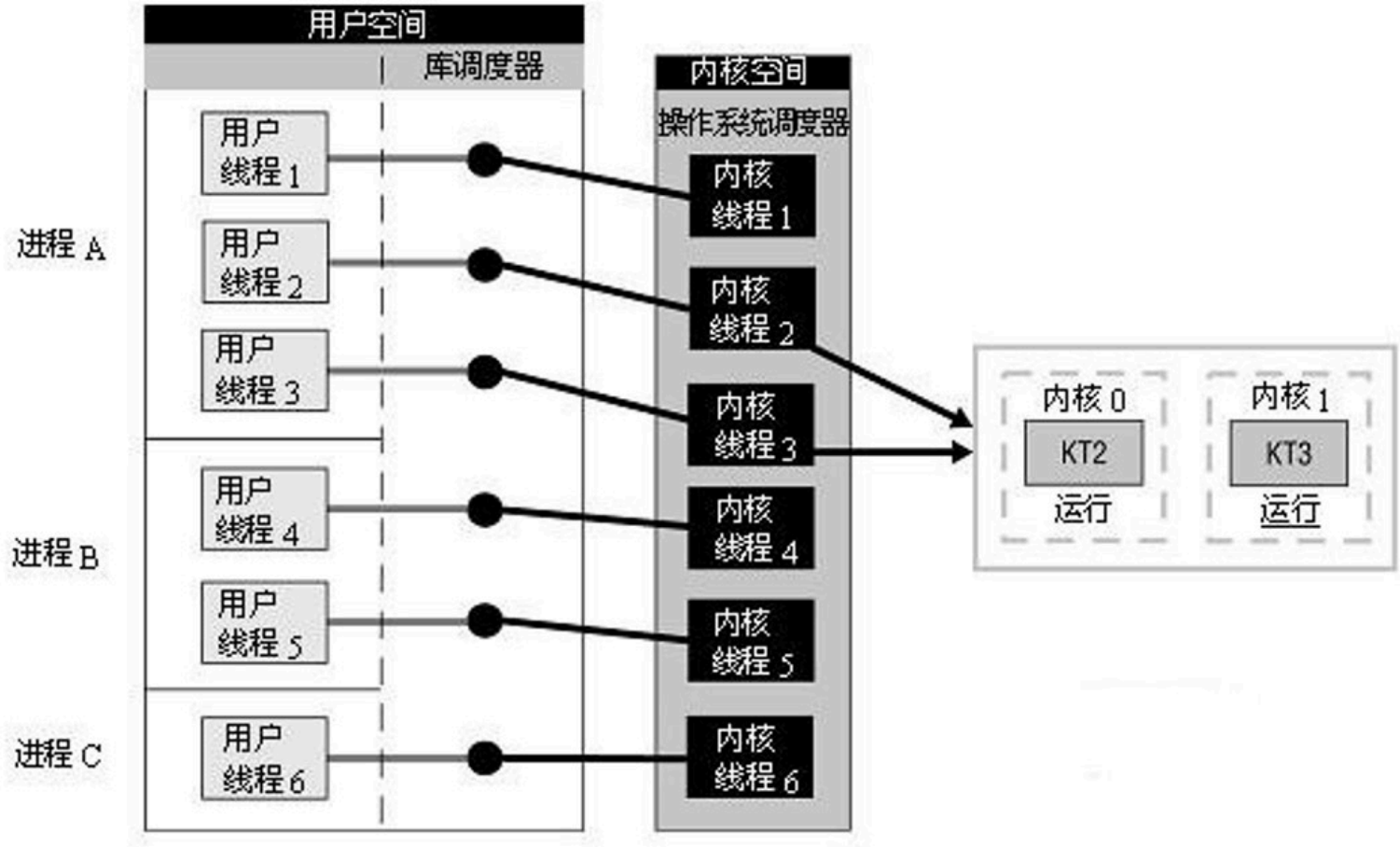


具体实现：

- 内核支持线程的实现（利用系统调用）
- 用户级线程的实现（借助中间系统）

5.4 线程实现

内核支持线程KST



5.4 线程实现

在内核空间实现

优点：

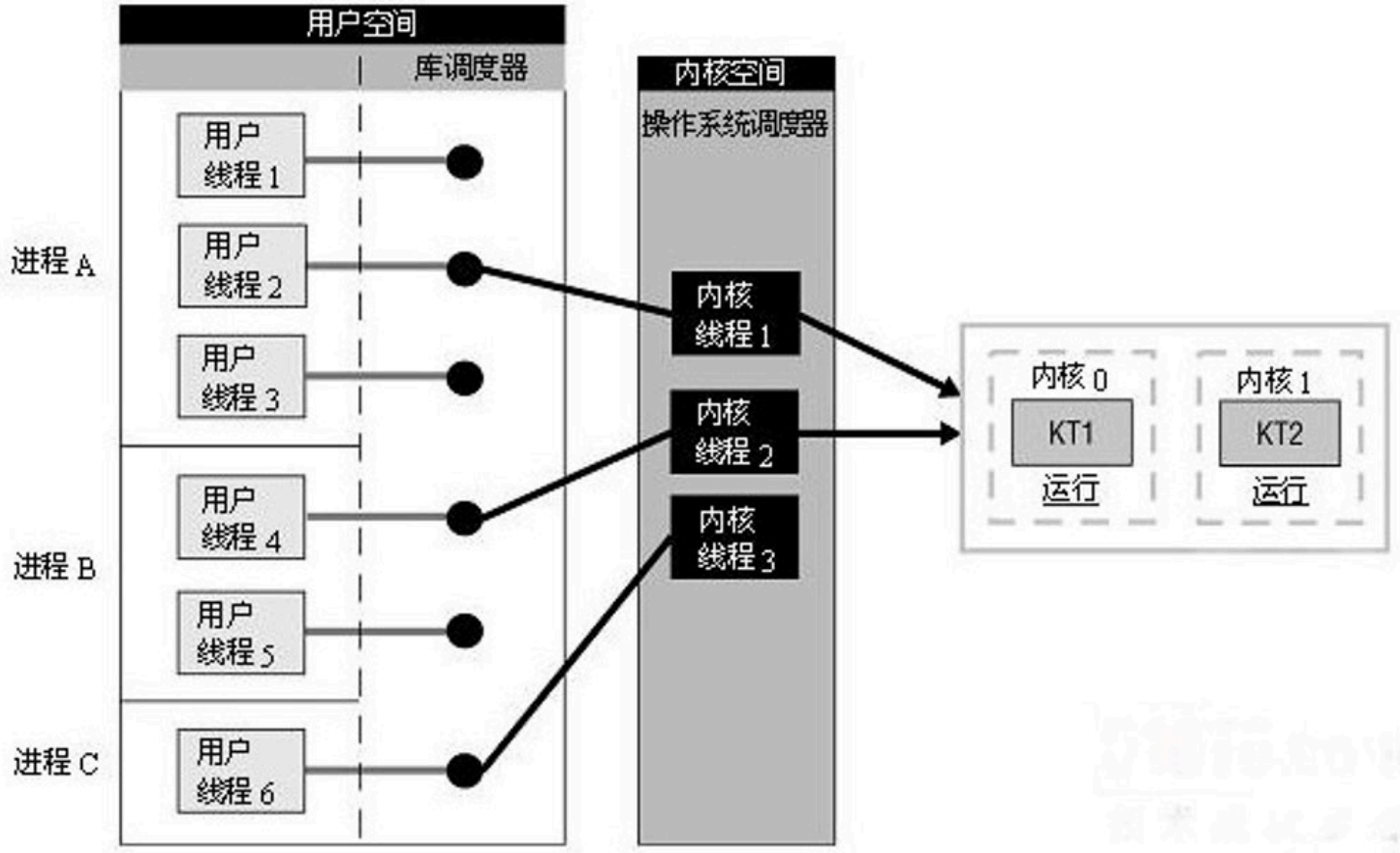
- 在多处理机系统中，内核可同时调度同一进程的多个线程
- 如一个线程阻塞了，内核可调度其他线程(同一或其他进程)。
- 线程的切换比较快，开销小。
- 内核本身可采用多线程技术，提高执行速度和效率。

缺点：

- 对用户线程切换，开销较大。

5.4 线程实现

用户级线程ULT



5.4 线程实现

在用户空间实现

优点：

- 线程切换不需要转换到内核空间。
- 调度算法可以是进程专用的。
- 线程的实现与OS平台无关。

缺点：

- 系统调用的阻塞问题。
- 多线程应用不能利用多处理机进行多重处理的优点。

5.4 线程实现

混合方式——多对一模型

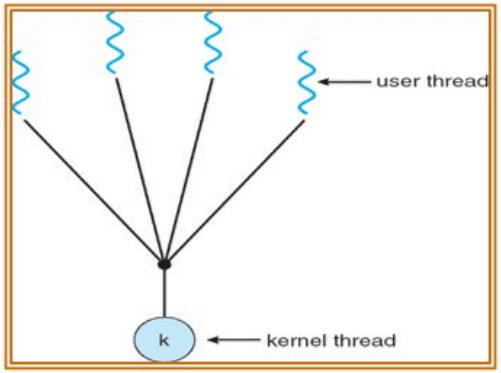
01 多个用户级线程映射到一个内核线程。

02 多个线程不能并行运行在多个处理器上。

03 线程管理在用户态执行，因此是高效的，但一个线程的阻塞系统调用会导致整个进程的阻塞 。

04 用于不支持内核线程的系统中。

05 例子:
- Solaris Green Threads
- GNU Portable Threads



5.4 线程实现

混合方式——一对一模型

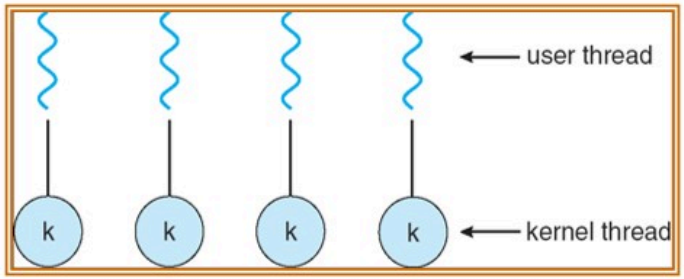
01 每个用户级线程映射到一个内核线程。

02 比多对一模型有更好的并发性。

03 允许多个线程并行运行在多个处理器上。

04 创建一个ULT需要创建一个KLT，效率较差。

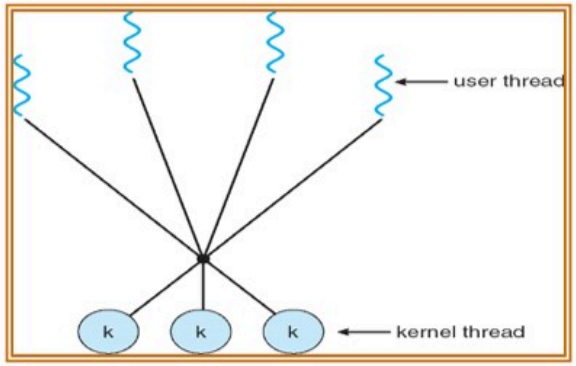
05 例子：
- Windows 95/98/NT/XP/2000
- Linux
- Solaris 9 and later
- OS/2



5.4 线程实现

混合方式——多对多模型

- 多个用户级线程映射为相等或小于数目的内核线程。
- 允许操作系统创建足够多的KLT。
- 例子：
  - Solaris 9 以前的版本；
  - 带有ThreadFiber开发包的Windows NT/2000 。



Part 5 线程概念及实现

进程线程对比

对比维度	多进程	多线程	总结
数据共享、同步	数据共享复杂，需要用IPC；数据是分开的，同步简单	因为共享进程数据，数据共享简单，但也是因为这个原因导致同步复杂	各有优势
内存、CPU	占用内存多，切换复杂，CPU利用率低	占用内存少，切换简单，CPU利用率高	线程占优
创建销毁、切换	创建销毁、切换复杂，速度慢	创建销毁、切换简单，速度很快	线程占优
编程、调试	编程简单，调试简单	编程复杂，调试复杂	进程占优
可靠性	进程间不会互相影响	一个线程挂掉将导致整个进程挂掉	进程占优
分布式	适应于多核、多机分布式；如果一台机器不够，扩展到多台机器比较简单	适应于多核分布式	进程占优

Part 5 线程概念及实现

进程线程对比

1.需要频繁创建销毁的优先用线程

2.需要进行大量计算的优先使用线程。所谓大量计算，会耗费很多CPU，切换频繁，优先选择线程。

3.强相关处理用线程，弱相关处理用进程

4.可能要扩展到多机分布的用进程，多核分布用线程。

单选题 1分

关于线程的理解，描述正确的是（ ）。

A 一个进程只能创建一个线程

B 线程是资源分配的单位

C 线程之间的通信必须使用调用系统函数完成

D 线程有自己的CPU执行信息，可以独立执行

单选题 1分

关于进程和线程正确的是（ ）。

A 线程是资源分配的单位，进程是资源调度的单位

B 不管系统中是否有线程，进程都是拥有资源的独立单位

C 有了线程，进程仍然是资源分配和调度的单位

D 一个进程只能创建一个线程

单选题 1分

下列关于进程和线程说法不正确的是？（ ）

A 线程可以称为轻量级的进程

B 一个进程可以有多个线程

C 进程之间可以共享资源，比如文件描述符等

D 线程是CPU调度的最小单位

单选题 1分

关于内核线程和用户线程，描述不正确的是( )

A 在多机系统中，调度可以为一个进程中的多个内核线程分配多个CPU

B 当进程中的一个用户线程被阻塞时，整个进程并不用等待

C 采用轮转调度算法，进程中设置内核线程和用户线程的效果完全不同

D 当内核线程阻塞时，CPU将会调度同一进程中的其他内核线程执行

多选题 2分

关于多线程和多线程编程，以下哪些说法正确的（）

A 多进程之间的数据共享比多线程编程复杂

B 多线程的创建，切换，销毁速度快于多进程

C 对于大量的计算优先使用多进程

D 多线程没有内存隔离，单个线程崩溃会导致整个应用程序的退出

知识回顾

Q1：进程是什么?

Q2：进程在内存中的存储映像包括什么?

Q3：进程如何切换?

Q4：进程之间如何通信?

Q5：线程是什么？为什么引入线程?

Q6：线程和进程有何关系?