线程

1. 多线程编程优点：响应度高、资源共享、经济、多处理器体系结构的利用.

2. 提供线程支持：用户线程：受内核支持，不受内核管理

内核线程：由操作系统直接支持管理.

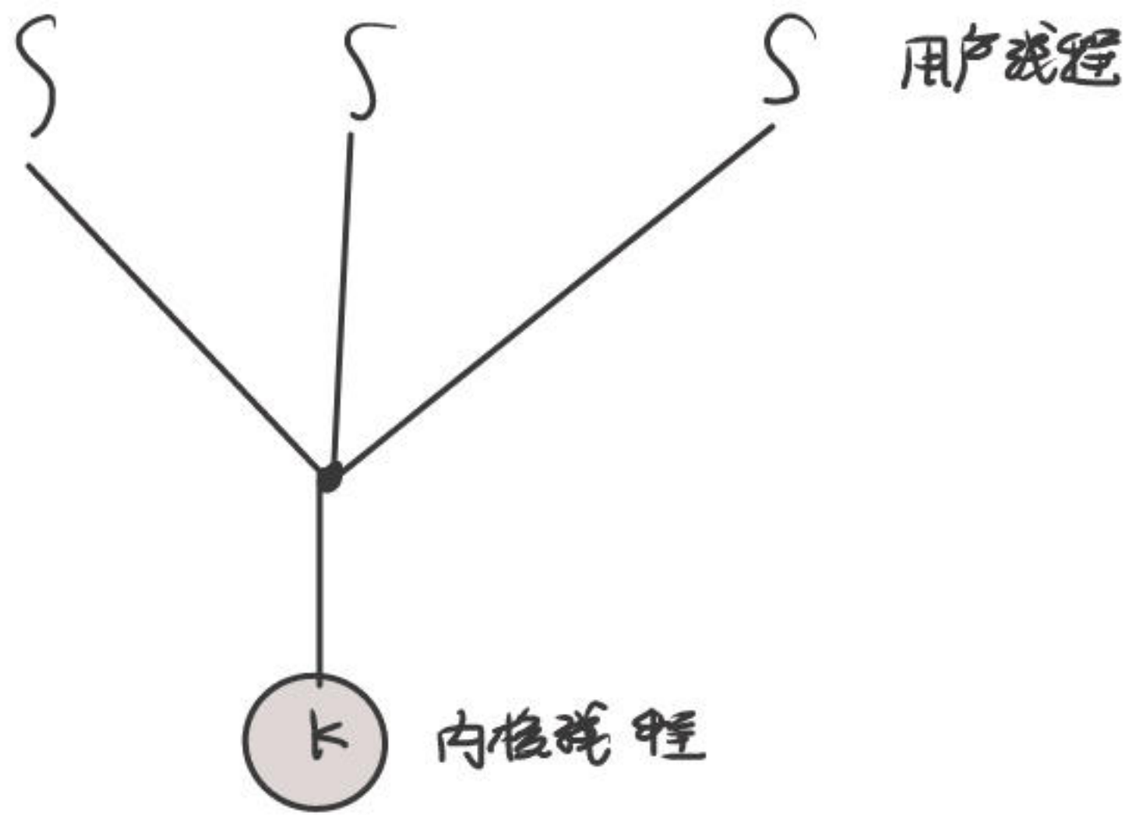
3. 用户线程与内核线程的关系

(1) 多对一模型：许多用户级线程映射到一个内核线程

效率高

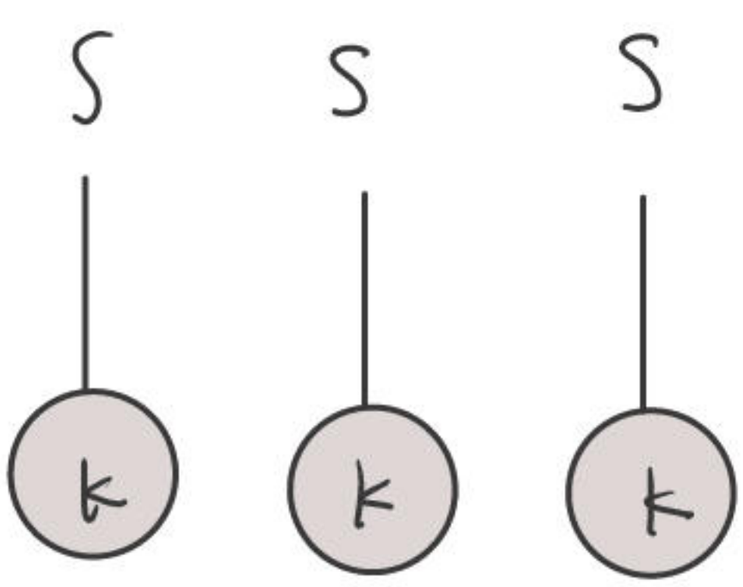
若一个线程阻塞系统调用，整个进程会阻塞

任一时刻只一个线程可访问内核

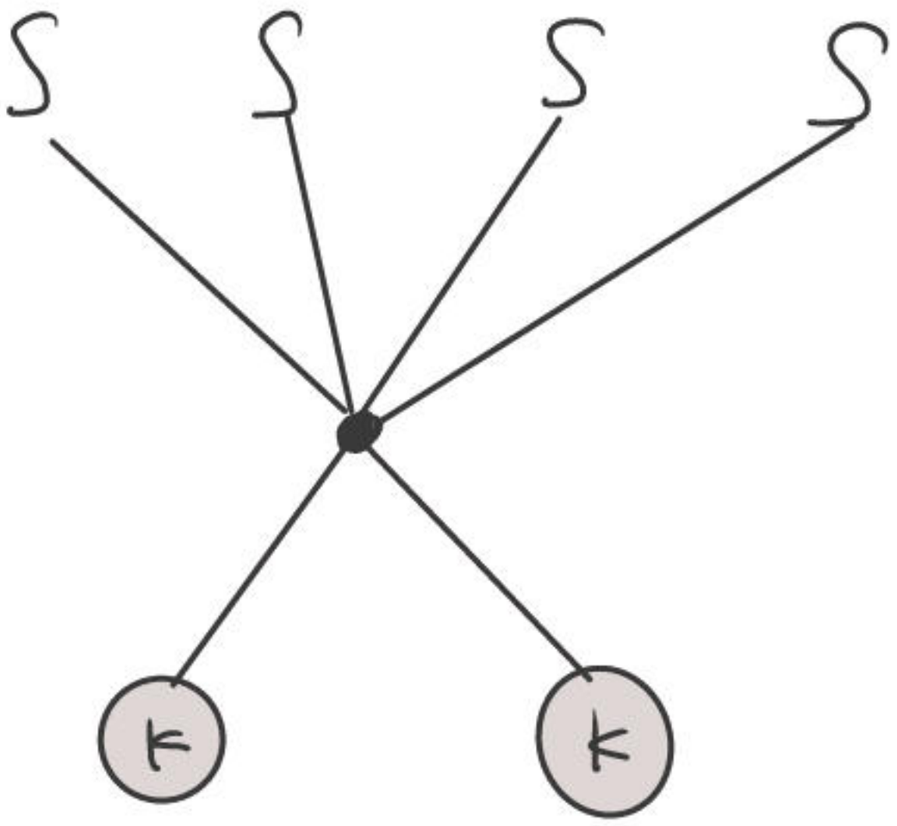


(2) 一对一模型，每个用户线程映射到一个内核线程

创建线程开销大



(3) 多对多模型：　许多用户线程到内核数量或更少数量
内核线程上。



4. 线程库：　为程序员提供创建和管理线程的API
- 提供一个没有内核支持的库
- 执行一个由操作系统直接支持的内核级的库
  - POSIX Pthread
  - Win32
  - Java

5. fork()：创建新进程，复制所有线程或只复制调用了fork()的线程
exec()：其参数所指定的程序会替换整个进程
线程取消：线程完成前终止线程
- 异步取消：立即终止
- 延迟取消：不断检查是否应终止

6. 线程池：　在进程开始时创建一定数量线程
一旦线程完成工作，它便返回池中等待工作

CPU 调度

1. 发生环境：
   (1) 运行态 → 等待态
   (2) 运行态 → 就绪态
   (3) 等待态 → 就绪态
   (4) 终止时

   若调度只发生在(1)(4)，则为非抢占
   否则为抢占

2. 调度准则
   CPU使用率：尽可能忙
   吞吐量：一个时间单元内完成的进程数量
   周转时间：进程提交～进程完成
   等待时间：进程在就绪队列中的等待时间
   响应时间：从提交请求到产生第一响应的时间.

3. 先到先服务调度 FCFS
   利用FIFO队列
   护航效果

4. 最短作业优先调度 SJF
   赋予具有最短CPU区间的进程.
   计算区间算法

5. 优先级调度
   每一个进程与优先级关联
   产生无穷阻塞或饥饿
   解决：老化：提高优先级

6. 轮转法调度 RR
   没有进程被分配超过一个时间片的CPU时间

7. 多级队列调度：

多个独立队列，每个队列调度算法不同

进程永久分配到一个队列

8 多级反馈队列调度：

允许进程在队列间移动

9. 多处理器调度：

非对称多处理：一个处理器处理所有调度及活动

其它处理器执行用户代码

对称多处理：每个处理器自我调度

进程同步

1. 临界区：当一个进程进入临界区，没有其它进程被允许在临界区内执行

进入区

临界区

退出区

剩余区

2. 临界区问题解答要求：

互斥、前进、有限等待

非抢占内核不会导致临界区问题

3. Peterson 算法,

```
do {   flag[i] = TRUE
       turn = j
       while flag[j] && turn==j ) ;
       临界区
       flag[i] = FALSE
       剩余区
} while (TRUE)
```

4. 硬件同步：锁

```
boolean TestAndSet (boolean * taryet) {
    boolean ru = * target
    * target = TRUE
    return ru
}

void swap ....
```

5. 信号量

```
P: wait (s) {
       while (s<=0) ;
       S--
   }
V: signal(s) {
       S++
   }
```

do {

wait(mutex)

临界区

signal(mutex)

剩余区

}

6-管程：提供一组由程序员定义、在管程内互斥的操作

确保一次只有一个进程在管程内活动

死锁

1. 所申请的资源被其它等待进程占有.

2. 必要条件：互斥. 占有并等待. 非抢占. 循环等待

3. 处理方法：使用协议以预防或避免死锁

允许系统进入死锁状态，然后检测它并加以恢复

忽视这个问题.

死锁预防：确保至少一个条件不发生

死锁避免：操作系统事先得到有关进程申请与使用资源的信息