内核线程经常被称为内核“守护进程”， 轻量级进程，也被称为线程

引入线程的目的：

提高系统的资源利用率，减少因频繁的进程调度和切换带来的大量系统开销。提高系统 的并发粒度。

线程的特点：

1、线程共享同一个进程的许多资源，

2、线程的切换，系统开销小，

3、cpu执行的最小单位

4、线程是进程内部的东西

内核线程处理任务类型：

l  周期性地将修改的内存页与页来源块设备同步。

l  如果内存页很少使用，则写入交换区。

l  管理延时动作

l  实现文件系统的事务日志

内核线程主要有两种类型：

1． 线程启动后一直等待，直至内核请求线程执行某一特定操作。

2． 线程启动后按周期性间隔运行，检测特定资源的使用，在用量超出或低于预置的限 制时采取行动。

**线程和进程的差别**

在现代操作系统中，进程支持多线程。进程是资源管理及分配的最小单元；而线程是程序执行的最小单元。一个进程的组成实体可以分为两大部分：线程集和资源集。进程中的线程是动态的对象，代表了进程指令的执行过程。资源，包括地址空间、打开的文件、用户信息等等，由进程内的线程共享。线程有自己的私有数据：程序计数器，栈空间以及寄存器。

多线程的必要性：

单线程：现实中有很多需要并发处理的任务，如数据库的服务器端、网络服务器、大容量计算等。传统的UNIX进程是单线程的，单线程意味着程序必须是顺序执行，不能并发，即在一个时刻只能运行在一个处理器上，因此不能充分利用多处理器框架的计算机。

多进程：如果采用多进程的问题：

 fork一个子进程的消耗是很大的，fork是一个昂贵的系统调用。

各个进程拥有自己独立的地址空间，进程间的协作需要复杂的IPC技术，如消息传递和共享内存等。

线程推广了进程的概念，使一个进程可以包含多个活动（或者说执行序列等等）。多线程的优点和缺点实际上是对立统一的。使用线程的优点在于：

改进程序的实时响应能力；

更有效的使用多处理器，真正的并行(parallelism)；

改进程序结构，具备多个控制流；

通讯方便，由于共享进程的代码和全局数据；

减少对系统资源的使用。对属于同一个进程的线程之间进行调度切换时不需要调用系统调用，因此将减少额外的消耗，往往一个进程可以启动上千个线程也没有什么问题。

缺点在于：

由于各线程共享进程的地址空间，因此可能会导致竞争，因此对某一块有多个线程要访问的数据需要一些同步技术。

2      线程的分类

2.1     内核线程

Linux内核可以看作一个服务进程(管理软硬件资源，响应用户进程的种种合理以及不合理的请求)。内核需要多个执行流并行，为了防止可能的阻塞，多线程化是必要的。内核线程就是内核的分身，一个分身可以处理一件特定事情。Linux内核使用内核线程来将内核分成几个功能模块，像kswapd、kflushd等，这在处理异步事件如异步IO时特别有用。

内核线程的使用是廉价的，唯一使用的资源就是内核栈和上下文切换时保存寄存器的空间。支持多线程的内核叫做多线程内核(Multi-Threads kernel )。内核线程的调度由内核负责，一个内核线程处于阻塞状态时不影响其他的内核线程，因为其是调度的基本单位。这与用户线程是不一样的。

2.2     轻量级进程

轻量级进程(LWP)是一种由内核支持的用户线程。它是基于内核线程的高级抽象，因此只有先支持内核线程，才能有LWP。每一个进程有一个或多个LWPs，每个LWP由一个内核线程支持。这种模型实际上就是恐龙书上所提到的一对一线程模型。在这种实现的操作系统中，LWP就是用户线程。

由于每个LWP都与一个特定的内核线程关联，因此每个LWP都是一个独立的线程调度单元。即使有一个LWP在系统调用中阻塞，也不会影响整个进程的执行。

轻量级进程具有局限性。首先，大多数LWP的操作，如建立、析构以及同步，都需要进行系统调用。系统调用的代价相对较高：需要在user mode和kernel mode中切换。其次，每个LWP都需要有一个内核线程支持，因此LWP要消耗内核资源（内核线程的栈空间）。因此一个系统不能支持大量的LWP。

注：

LWP的术语是借自于SVR4/MP和Solaris 2.x。将之称为轻量级进程的原因可能是：在内核线程的支持下，LWP是独立的调度单元，就像普通的进程一样。所以LWP的最大特点还是每个LWP都有一个内核线程支持。

2.3     用户线程

LWP虽然本质上属于用户线程，但LWP线程库是建立在内核之上的，LWP的许多操作都要进行系统调用，因此效率不高。而这里的用户线程指的是指不需要内核支持而完全建立在用户空间的线程库，用户线程的建立、同步、销毁及调度完全在用户空间完成，不需要内核的参与帮助。因此这种线程的操作是极其快速的且低消耗的。

上图是最初的一个多对一用户线程模型，从中可以看出，进程中包含线程，用户线程在用户空间中实现，内核并没有直接对用户线程进程调度，内核的调度对象和传统进程一样，还是进程本身，内核并不知道用户线程的存在。

由于Linux内核没有轻量级进程（线程）的概念，因此不能独立的对用户线程进行调度，而是由一个线程运行库来组织线程的调度，其主要工作在于在各个线程的栈之间调度。如果一个进程中的某一个线程调用了一个阻塞的系统调用，整个进程就会被调度程序切换为等待状态，其他线程得不到运行的机会。因此UNIX使用了异步I/O机制。这种机制主要的缺点在于在一个进程中的多个线程的调度中无法发挥多处理器的优势（如上述的阻塞情况）。

3      初识内核线程

内核线程（thread）或叫守护进程(daemon)，在操作系统中占据相当大的比例，当Linux操作系统启动以后，尤其是X window也启动以后，你可以用”ps -ef”命令查看系统中的进程，这时会发现很多以”d”结尾的进程名，确切说名称显示里面加 "[]"的，这些进程就是内核线程。系统的启动是从硬件->内核->用户态进程的，pid的分配是一个往前的循环的过程，所以随系统启动的内核线程的pid往往很小。

?      PID TTY STAT TIME COMMAND

?      1 ? S 0:01 init [3]

?      3 ? SN 0:00 [ksoftirqd/0]

?      5 ? SN 0:00 [ksoftirqd/1]

?      6 ? S< 0:00 [events/0]

?      7 ? S< 0:00 [events/1]

?      8 ? S< 0:00 [khelper]

?      9 ? S< 0:00 [kblockd/0]

?      10 ? S< 0:00 [kblockd/1]

?      11 ? S 0:00 [khubd]

?      35 ? S 0:42 [pdflush]

?      36 ? S 0:02 [pdflush]

?      38 ? S< 0:00 [aio/0]

?      39 ? S< 0:00 [aio/1]

?      37 ? S 0:19 [kswapd0]

?      112 ? S 0:00 [kseriod]

?      177 ? S 0:00 [scsi\_eh\_0]

?      178 ? S 0:00 [ahc\_dv\_0]

?      188 ? S 0:00 [scsi\_eh\_1]

?      189 ? S 0:00 [ahc\_dv\_1]

?      196 ? S 2:31 [kjournald]

?      1277 ? S 0:00 [kjournald]

?      1745 ? Ss 0:02 syslogd -m 0

?      1749 ? Ss 0:00 klogd -x

?      1958 ? Ss 0:13 /usr/sbin/sshd

?      2060 ? Ss 0:00 crond

?      2135 tty2 Ss+ 0:00 /sbin/mingetty tty2

?      2136 tty3 Ss+ 0:00 /sbin/mingetty tty3

?      2137 tty4 Ss+ 0:00 /sbin/mingetty tty4

?      2138 tty5 Ss+ 0:00 /sbin/mingetty tty5

?      2139 tty6 Ss+ 0:00 /sbin/mingetty tty6

?      23564 ? S 0:00 bash

?      25605 ? Ss 0:00 sshd: peter [priv]

?      25607 ? S 0:00 sshd: peter@pts/2

3.1     内核线程

?      events 处理内核事件 很多软硬件事件(比如断电，文件变更)被转换为events，并分发给对相应事件感兴趣的线程进行响应

?      ksoftirqd 处理软中断 硬件中断处理往往需要关中断，而这个时间不能太长，否则会丢失新的中断。所以中断处理的很大一部分工作移出，转给任劳任怨的ksoftirqd在中断之外进行处理。比如一个网络包，从网卡里面取出这个过程可能需要关中断，但是TCP/IP协议处理就不必关中断了

?      kblockd 管理磁盘块读写

?      kjournald Ext3文件系统的日志管理 通常每个 \_已mount\_ 的 Ext3分区会有一个 kjournald看管，各分区的日志是独立

?      pdflush dirty内存页面的回写 太多dirty的页面意味着风险，比如故障时候的内容丢失，以及对突发的大量物理内存请求的响应(大量回写会导致糟糕的响应时间)

?      kswapd 内存回收 确保系统空闲物理内存的数量在一个合适的范围

?      aio 代替用户进程管理io 用以支持用户态的AIO