<https://blog.csdn.net/thanksgining/article/details/41890293>

多任务的概念人们已经非常熟悉了，它是指用户可以在同一时间内运行多个应用程序。Linux就是一种支持多任务的操作系统，它支持多进程、多线程等多任务处理和任务之间的多种通信机制。

**Linux下多任务机制的介绍**

多任务处理是指用户在同一时间内运行多个应用程序，每个应用程序被称做一个任务。Linux就是一个支持多任务的操作系统，它比单任务系统的功能增强了许多。当多任务操作系统使用某种任务调度策略允许两个或更多进程并发共享一个处理器时，事实上处理器在某一时刻只会给一个任务提供服务。由于任务调度机制保证了不同的任务之间的切换速度十分迅速，因此给人多个任务同时运行的错觉。多任务系统中有3个功能单位：任务、进程和线程，下面分别进程介绍。

**1、任务**

任务是一个逻辑概念，指由一个软件完成的活动，或者是一系列共同达到某一个目的的操作。通常一个任务是一个程序的一次运行，一个任务包含一个或多个完成独立功能的子任务，这个独立的子任务就是进程或线程。例如，一个杀毒软件的一次运行是一个任务，目的是从各种病毒的侵害中保护计算机系统，这个任务包含多个独立功能的子任务（进程或线程），包含实时监控功能、定时查杀功能、防火墙功能及用户交互功能等。**个人理解：就好比假设一个应用程序中由一个或多个可执行文件共同执行组成，那么此应用程序的一次执行就是一个任务，而这些可执行文件的执行就是一个进程的执行，而可执行文件是由一个线程或多个线程构成的，当只有一个线程构成了这个进程，则此时进程和线程就是一样的概念（可执行文件的一次运行）。**

**2、进程**

**进程的基本概念**

进程是一个具有独立功能的程序在某个数据集上的一次动态执行过程，它是系统进行资源分配和调度的基本单位（**个人理解：系统好比是一个大型的任务，由多个进程(可执行文件)构成，而资源分配和资源调度分别都是一个进程，所以进程是系统进行资源分配和调度的基本单位**）。一次任务的运行可以并发激活多个进程，这些进程相互合作来完成该任务的一个最终的目标。

进程具有并发性、动态性、交互性、独立性和异步性等主要特性

* 并发性：指的是系统中多个进程可以同时并发执行，相互之间不受干扰。
* 动态性：指的是进程都有完整的生命周期，而且在进程的生命周期内，进程的状态是不断变化的。另外，进程具有动态的地址空间（包括代码、数据和进程控制块）。
* 交互性：指的是进程在执行过程中可能会与其他进程发生直接和间接的交互操作，如进程同步和进程互斥等，需要为此添加一定的进程处理机制。
* 独立性：指的是进程是一个相对完整的资源分配和调度的基本单位，各个进程的地址空间是相互独立的，只有采用某些特定的通信机制才能实现进程间的通信。
* 异步性：指的是每个进程都按照各自独立的、不可预知的速度向前执行。

进程和程序是有本质的区别：程序是静态的一段代码，是一些保存在非易失性存储器的指令的有序集合，没有任何执行的概念；而进程是一个动态的概念，它是程序执行的过程，包括动态创建、调度和消亡的整个过程，它是程序执行和资源管理的最小单位。

Linux系统中包括以下几种类型的过程：

* 交互式过程：这类进程进程与用户进程交互，因此要花很多时间等待用户的交互操作（键盘和鼠标操作等）。当接收到用户的交互操作后，这类进程应该很快被允许，而且相应时间的变化也应该很小，否则用户就会觉得系统反应迟钝或者不太稳定。典型的交互式进程有shell命令进程、文本编辑器和图形应用程序运行等。
* 批处理进程：这类进程不必与用户进行交互，因此进程在后台运行。因为这类进程通常不必很快地相应，因此往往受到调度器的“慢待”。典型的批处理进程有编译器的编译操作、数据库搜索引擎等。
* 实时进程：这类进程通常对调度响应时间有很高的要求，一般不会被低优先级的进程阻塞。它们不仅要求很短的响应时间，而且更重要的是响应时间的变化应该很小。典型的实时进程有视频和音频的应用程序、实时数据采集系统程序等。

**Linux下的进程结构**

进程不但包括程序的指令和数据，而且包括程序计数器和处理器的所有寄存器以及存储临时数据的进程堆栈，因此正在执行的进程包括处理器当前的一切活动。

因为Linux是一个多进程的操作系统，所以其他的进程必须等到系统将处理器使用权分配各自己之后才能运行。当正在运行的进程等待其他的系统资源时，Linux内核将取得处理器的控制权，并将处理器分配给其他正在等待的进程，它按照内核中的调度算法决定处理器分配给哪个进程。

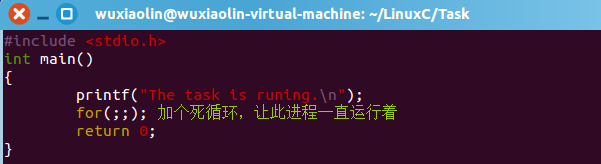
内核将所有进程存放在双向循环链表（进程链表）中，其中链表的头是init\_task描述符。链表的每一项都是类型为task\_struct，称为进程描述符的结构，该结构包含了与一个进程相关的所有信息，定义在<include/linux/sched.h>文件中。task\_struct内核结构比较大，它能完整地描述一个进程，如进程的状态、进程的基本信息、进程标识符、内存相关信息、父进程相关信息、与进程相关的终端信息、当前工作目录、打开的文件信息、所接收到的信号信息等。

下面详细讲解task\_struct结构中最为重要的两个域：state（进程状态）和pid（进程标识符，即进程号）。

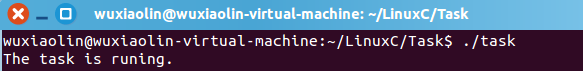
**1）进程状态，Linux中的进程有以下几种状态**

* 运行状态（TASK\_RUNNING）：进程当前正在运行，或者正在运行队列中等待调度。

创建一个task.c文件，task.c文件内容如下：



保存后，输入gcc task.c -o task编译生成二进制代码task，输入./task运行task进程



打开另一个终端，输入ps -aux查看进程状态：（ps -axjf 可查看进程有哪些子进程，ps -e 也 可以查到进程的状态，但只显示进程的PID、TTY、TIME和CMD）

ps工具标识进程的5中状态码：

D 不可中断 uninterruptible sleep (usually IO)

R 运行 runnable (on run queue)

S 中断 sleeping

T 停止 traced or stopped

Z 僵尸 a defunct ("zombie") process

注：其它状态还包括W(无驻留页)，<(高优先级进程)，N(低优先级进程)，L(内存锁页)

每列对应关系：

USER：进程所有者

PID：进程ID

%CPU：占用CPU的使用率

%MEM：占用内存的使用率

VSZ：占用虚拟内存大小

 RSS：占用内存大小

TTY：终端次要装置号码

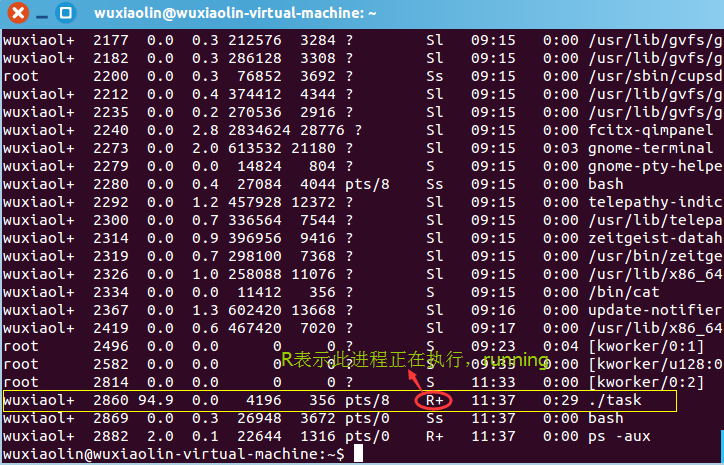
STAT：进程状态

START：进程启动时间

TIME：进程消耗cup时间

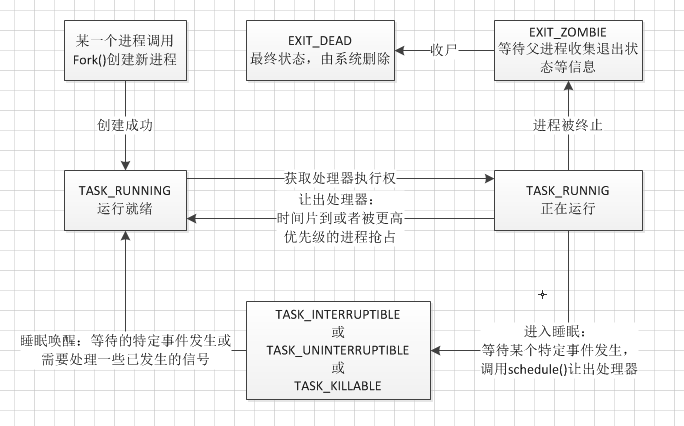
COMMAND：命令的名称和参数

https://img-blog.csdn.net/20141212115337640



* 可中断的阻塞状态（TASK\_INTERRUPTIBLE）：进程处于阻塞(睡眠)状态，正在等待某些事件发生或能够占用某些资源。处在这种状态下的进程可以被信号中断。接收到信号或被显式的唤醒呼叫（如调用wake\_up系列宏：wake\_up、wake\_up\_interruptible等）唤醒之后，进程转变为TASK\_RUNNING状态。
* 不可中断的阻塞状态（TASK\_UNINTERRUPTIBLE）：此进程状态类似于可中断的阻塞状态（TASK\_INTERRUPTILBE），只是它不会处理信号，把信号传递到这种状态下的进程不能改变它的状态。在一些特定的情况下（进程必须等待，直到某些不可被中断的事件发生），这种状态是很有用的。只有在它所等待的事件发生时，进程被显式的唤醒呼叫唤醒。
* 可终止的阻塞状态（TASK\_KILLABLE）：Linux内核2.6.25引入了一种新的进程状态，名为TASK\_KILLABLE。该状态的运行机制类似于TASK\_UNINTERRUPTILBE，只不过在该状态下的进程可以响应致命信号。它可以替代有效但可能无法终止的不可中断的阻塞状态，以及易于唤醒安全性欠佳的可中断的阻塞状态。
* 暂停状态（TASK\_STOPPED）：进程的执行被暂停，当进程收到SIGTOP、SIGTSTP、SIGTTIN、SIGTTOU等信号时，就会进入暂停状态。
* 跟踪状态（TASK\_TRACED）：进程的执行被调试器暂停。当一个进程被另一个进程监控是（如调试器使用ptrace()系统调用监控测试程序），任何信号都可以把这个进程置于跟踪状态。
* 僵尸状态（EXIT\_ZOMBIE）：进程运行结束，父进程尚未使用wait函数族（如使用waitpid()函数）等系统调用来“收尸”，即等待父进程销毁它。处于该状态下的进程“实体”已经放弃了几乎所有的内存空间，没有任何可执行代码，也不能调度，仅仅在进程列表保留一个位置，记载该进程的退出状态等信息供其他进程收集。
* 僵尸撤销状态（EXIT\_DEAD）：这是最终状态，父进程调用wait函数族“收尸”后，进程彻底有系统删除。

它们之间的转换关系如图所示：



内核可以使用set\_task\_state和set\_current\_state宏来改变指定进程的状态和当前执行进程的状态。

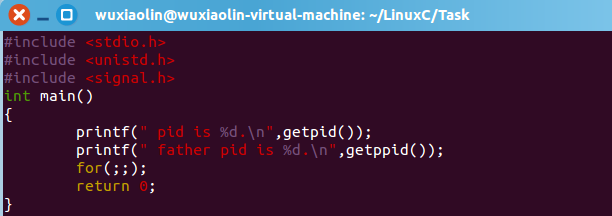
**2）进程标识符**

Linux内核通过唯一的进程标识符PID来标识每个进程。PID存放进程描述符的pid字段中，新创建的PID通常是前一个进程的PID加1，不过PID的值有上限（最大值 = PID\_MAX\_DEFAULT - 1，通常为32767），我们可以在终端输入 vim /proc/sys/kernel/pid\_max 来确定该系统的进程数上限。

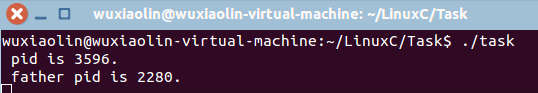
当系统启动后，内核通常作为一个进程的代表。一个指向task\_struct的宏current用来记录正在运行的进程。current经常作为进程描述符结构指针的形式出现在内核代码中，例如，current->pid表示处理器正在执行进程的PID。当系统需要查看所有的进程时，则调用for\_each\_process()宏，这将比系统搜索数组的速度要快得多。

在Linux中获得当前进程的进程号(PID)和父进程号(PPID)的系统调用函数分别为getpid()和getppid()。

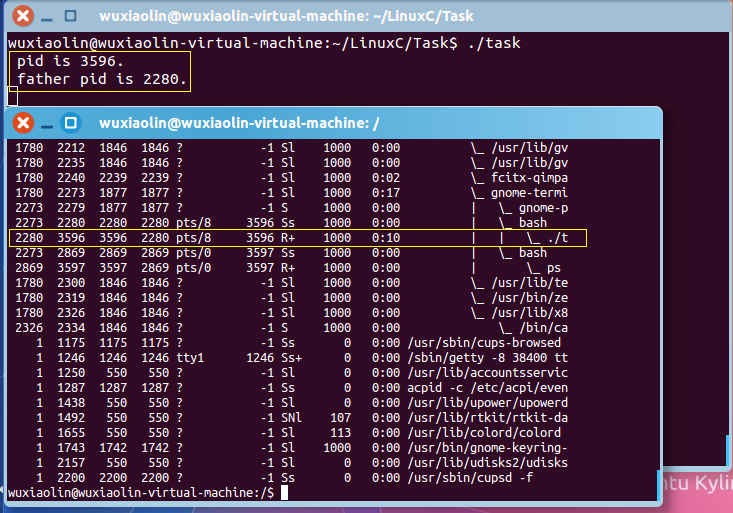
测试代码：



测试结果：



输入 ps -axjf 命令查看所有进程与父进程



我们在次输入ps -aux命令查看所有进程，可以得知父进程为bash

https://img-blog.csdn.net/20141212153222281

**进程的创建、执行和终止**

**1）进程的创建和执行**

许多操作系统提供的都是产生进程的机制，也就是说，首先在新的地址空间里创建进程、读入可执行文件、最后在开始执行。Linux中进程的穿件很特别，它把上述步骤分解到两个单独的函数中去执行：fork()和exec函数族。首先fork()函数通过复制当前进程创建一个子进程，子进程与父进程的区别在于不同的PID、PPID和某些资源及统计量。exec函数族负责读取可执行文件并将其载入地址空间开始运行。

要注意的是，Linux中的fork()函数使用的是写时复制页的技术，也就是内核在创建进程时，其资源并没有被复制过来，资源的复制仅仅只有在需要写入数据时才发生，在此之前只是以只读的方式共享数据。写时复制技术可以使Linux拥有快速执行的能力，因此这个优化是非常重要的。

**2）进程的终止**

进程终结也需要做很多繁琐的收尾工作，系统必须保证回收进程所占的资源，并通知父进程。Linux首先把终止的进程设置为僵尸状态，这时，进程无法投入运行，它的存在只为父进程提供信息，申请死亡。父进程得到信息后，开始调用wait函数族，最后终止子进程，子进程占用的所有资源被全部释放。

**进程的内存结构**

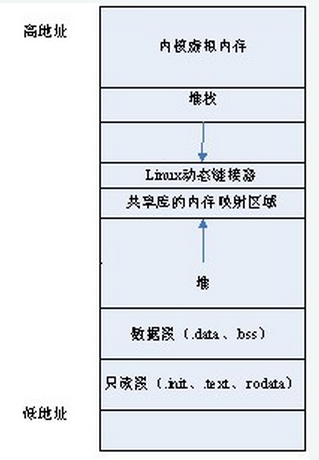
Linux操作系统采用虚拟内存管理技术，使得每个进程都有各自互不干涉的进程地址空间。该地址空间是大小为4GB的线性虚拟空间(当然是指32位系统)，用户所看到和接触到的都是该虚拟地址，无法看到实际的物理内存地址。利用这种虚拟地址不但能起到保护操作系统的效果（用户不能直接访问物理内存），而且更重要的是，用户程序可以使用比实际物理内存更大的地址空间。

我们可以通过命令getconf LONG\_BIT 来查询当前自己的系统是多少位的？

https://img-blog.csdn.net/20141212161632792

我的安装的UbuntKylin是64位的，即实际内存最大可能达到2^64 = 128GB。（2^10 = 1kb，2^30 = 1GB）。

4GB的进程地址空间会被分出两部分：用户空间与内核空间。用户地址空间是从0~3GB（0xC0000000），内存地址空间占据3GB~4GB。用户进程通常情况下只能访问用户控件的虚拟地址，不能访问内核空间的虚拟地址。只有用户进程使用系统调用（代表用户进程在内核执行）时可以访问内核空间的虚拟空间。**每当进程切换时，用户空间就会跟着变化；而内核空间有内核负责映射，它并不会跟着进程改变而改变，是固定的。**内核空间地址有自己对应的页表，用户进程各自用不同的页表。每个进程用户空间都是完全独立、互不相干的。进程的虚拟内存地址空间如图所示：

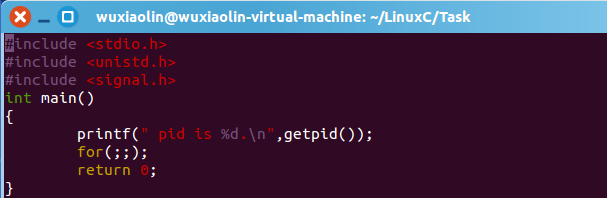


其中用户空间包括以下几个功能区域：

* 只读段：包含程序代码（.init和.exit）和只读数据(.rodata)
* 数据段：存放的是全局变量和静态变量。其中可读可写数据段（.data）存放已经初始化的全局变量和静态变量，BSS数据段(.bss)存放未初始化的全局变量和静态变量
* 堆：由系统自动分配释放，存放函数的参数值、局部变量的值、返回地址等
* 堆栈：存放动态分配的数据，一般由程序员动态分配和释放。若程序员不释放，程序结束时可能由操作系统回收。
* 共享库的内存映射区域：这是Linux动态连接器和其他共享库代码的映射区域。

由于在Linux系统中每一个进程都会有/proc文件系统下与之对应的一个目录(如将init进程的相关信息在/proc/1 目录下的文件中描述，1表示init进程的进程号)，因此通过proc文件系统可以查看某个进程的地址空间的映射情况。

测试代码：



运行此程序：

https://img-blog.csdn.net/20141212171532781

输入 size task

https://img-blog.csdn.net/20141212170518655

text：存放的是代码     data：存放的是初始化过的全局变量或静态变量   bss：存放的是未初始化的全局变量或静态变量

输入命令 cat /proc/3834/maps   其中3834是task的PID



**3、线程**

前面已经提到，进程是系统中程序执行和资源分配的基本单位。每个进程都拥有自己的数据段、代码段和堆栈段，这就造成了进程进程切换等操作时需要较复杂的上下文切换等动作。为了进一步减少处理机制的空转时间，支持多处理器及减少上下文切换开销，进程在演化中出现了另一个概念——线程。它是进程内独立的一条运行路线，是处理器调用的最小单元，也可以成为轻量级进程。线程可以对进程的内存空间和资源进程访问，并与同一个进程中的其他线程共享。因此，线程上下文切换的开销比创建进程小得多。

一个进程可以拥有多个线程，每个线程必须有一个父进程。线程不拥有系统资源，它只具有运行所必需的一些数据，如堆栈、寄存器与线程控制块（TCB），线程与其父进程的其他线程共享该进程所拥有的全部资源。要注意的是，由线程共享了进程的资源和地址空间，因此，任何线程对系统资源的操作都会给其他线程带来影响。由此可知，多线程中的同步是非常重要的问题。在多线程系统中，进程与线程的关系如图所示：



在Linux系统中，线程可以分为以下3种：

**用户级线程**

用户级线程主要解决的是上下文切换的问题，它的调度算法和调度过程全部由用户自己选择决定，在运行时不需要特定的内核支持。在这里，操作系统往往会提供一个用户空间的线程库，该线程库提供了线程的创建、调度和撤销等功能，而内核仍然仅对进程进行管理。**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函数，那么该进程好吧该进程中的其他所有线程也同时被阻塞**。这种用户级线程的主要缺点是在一个进程的多个线程的调度中无法发挥多处理器的优势。

**轻量级进程**

轻量级进程是内核支持的用户线程，是内核线程的一种抽象对象。每个线程拥有一个或多个轻量级进程，而每个轻量级进程分别被绑定在一个内核线程上。

**内核线程**

内核线程允许不同进程中的线程按照同一相对优先调度方法进行调度，这样就可以发挥多处理器的并发优势。现在大多数系统都采用用户级线程与核心级线程并存的方法。一个用户级线程可以对应一个或几个核心级线程，也就是“一对一”或“多对一”模型。这样既可以满足多处理系统的需要，也可以最大限度地减少调度开销。

使用线程机制大大加快了上下文切换速度，而节省了很多资源。但是因为在用户态和内核态均要实现调度管理，所有会增加实现的复杂度和引起优先级翻转的可能性。同时，一个多线程程序的同步设计与调试也会增加程序实现的难道。