**linux用户栈和内核栈解析**

2017年10月23日 16:58:34 [傲世阿龍](https://me.csdn.net/u010383937) 阅读数：210更多

个人分类： [Linux](https://blog.csdn.net/u010383937/article/category/6891567) [linux\_driver](https://blog.csdn.net/u010383937/article/category/7136688)

**进程是程序的一次执行过程。用剧本和演出来类比，程序相当于剧本，而进程则相当于剧本的一次演出，舞台、**

**灯光则相当于进程的运行环境。**

**进程的堆栈**

**每个进程都有自己的堆栈，内核在创建一个新的进程时，在创建进程控制块task\_struct的同时，也为进程创建**

**自己堆栈。一个进程 有2个堆栈，用户堆栈和系统堆栈；用户堆栈的空间指向用户地址空间，内核堆栈的空间**

**指向内核地址空间。当进程在用户态运行时，CPU堆栈指针寄存器指向的 用户堆栈地址，使用用户堆栈，当进**

**程运行在内核态时，CPU堆栈指针寄存器指向的是内核栈空间地址，使用的是内核栈；**

**进程用户栈和内核栈之间的切换**

**当进程由于中断或系统调用从用户态转换到内核态时，进程所使用的栈也要从用户栈切换到内核栈。系统调用**

**实质就是通过指令产生中断，称为软中断。进程因为中断（软中断或硬件产生中断），使得CPU切换到特权工**

**作模式，此时进程陷入内核态，进程进入内核态后，首先把用户态的堆栈地址保存在内核堆栈中，然后设置堆**

**栈指针寄存器的地址为内核栈地址，这样就完成了用户栈向内核栈的切换。**

**当进程从内核态切换到用户态时，最后把保存在内核栈中的用户栈地址恢复到CPU栈指针寄存器即可，这样就**

**完成了内核栈向用户栈的切换。**

**这里要理解一下内核堆栈。前面我们讲到，进程从用户态进入内核态时，需要在内核栈中保存用户栈的地址。**

**那么进入内核态时，从哪里获得内核栈的栈指针呢？**

**要解决这个问题，先要理解从用户态刚切换到内核态以后，进程的内核栈总是空的。这点很好理解，当进程在**

**用户空间运行时，使用的是用户 栈；当进程在内核态运行时，内核栈中保存进程在内核态运行的相关信息，但**

**是当进程完成了内核态的运行，重新回到用户态时，此时内核栈中保存的信息全部恢 复，也就是说，进程在内**

**核态中的代码执行完成回到用户态时，内核栈是空的。**

**理解了从用户态刚切换到内核态以后，进程的内核栈总是空的，那刚才这个问题就很好理解了，因为内核栈是**

**空的，那当进程从用户态切换到内核态后，把内核栈的栈顶地址设置给CPU的栈指针寄存器就可以了。**

**X86 Linux内核栈定义如下（可能现在的版本有所改变，但不妨碍我们对内核栈的理解）：**

**在／include/linux/sched.h中定义了如下一个联合结构：**

**union task\_union {**

**struct task\_struct task;**

**unsigned long stack[2408];**

**};**

**从这个结构可以看出，内核栈占8kb的内存区。实际上，进程的task\_struct结构所占的内存是由内核动态分配**

**的，更确切地说，内核根本不给task\_struct分配内存，而仅仅给内核栈分配8K的内存，并把其中的一部分给task\_struct使用。**

**这样内核栈的起始地址就是union task\_union变量的地址+8K 字节的长度。例如：我们动态分配一个union task\_union类型的变量如下：**

**unsigned char \*gtaskkernelstack**

**gtaskkernelstack = kmalloc(sizeof(union task\_union));**

**那么该进程每次进入内核态时，内核栈的起始地址均为：（unsigned char \*）gtaskkernelstack + 8096**

**知乎某大神回答:**

作者：唐浩然  
链接：https://www.zhihu.com/question/43699081/answer/124798606  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

为何使用用户 内核态堆栈：  
1，多是安全原因 ，进入系统调用需要 保存用户 态运行时的 寄存器信息，（包括控制寄存器信息 EFLAGS, 以及 cs/ip,)这些 保存在低权限 的用户态堆栈，实在不安全。  
2： 任务切换必须发生在内核态，在任务切换时，也需要使用用户的内核态堆栈，来保存 用户 态时的 寄存器信息（包换用户态堆栈指针）。 比如，最后将 新任务的 内核态堆栈中 的 保存的上次切换时保存 的用户 态寄存器 弹出，然后恢复新任务用户 态执行。  
3： 还有 注意到一个特点， 每当用户 程序 从用户 态进入内核态时， 用户程序的内核态 堆栈总是空的， 什么内容也没有，或许 这可能也是从安全层面的考虑？

以下是原答案   
权限检查的原因  
拿x86，2.6内核举例:  
1.内核的分页管理机制中，对页表及页全局目录的描述中，有分系统页表/页全局目录和用户页表/全局目录（即u/s位），为0时只有内核可以访问（cpu特权级为0，或者说cs寄存器的权限标志位为0，这个权限也适用于对“段”的访问，因为被访问的“段”也有权限级别，intel设计了四个段级别，由两个bit来表示，但linux只使用了00和11，即0和3特权级，表示内核态和用户态）  
2.系统调用实际上是调用内核提供给用户的函数接口（其产生就是软中断的产生，此时内核会保存用户态时的寄存器内容到内核堆栈，调用执行完在将保存的寄存器回复出去给用户），而函数在内核态执行，即这些函数所在内存的页面以及运行过程使用的堆栈地址占用的页面都落在在 页表 的u/s位标志为0的页上，内核的栈指针可以访问到这些页，用户态的进程对这些页面没有权限读写（页没有标志‘’执行‘权限’），用户态的堆栈指针指不到这些地址。  
***3. 再者，执行内核函数（系统调用）进入内核态，cs寄存器的cpl发生了切换，相应的ds也必须含有内核数据段的段选择符，ss段也同样要有内核数据段的段选择符。***  
4. 4.做个假设，如果执行内核线程，强行不执行各种权限的检查，强行使用用户态堆栈，内核所使用的线性地址和用户使用的线性地址方法有差异，内核的线性地址是固定的（内存也就一个工作内核），而所有用户进程可使用的线性地址都一样（但被映射到不同的物理地址），当内核访问用户空间时还需要做页面映射，将用户的堆栈相关的物理页再映射到内核页表？而每个用户的系统调用都做一次页面映射，这个性能开销应该不小吧，应该不如直接保存用户寄存器数据，系统调用完了再恢复来的快。