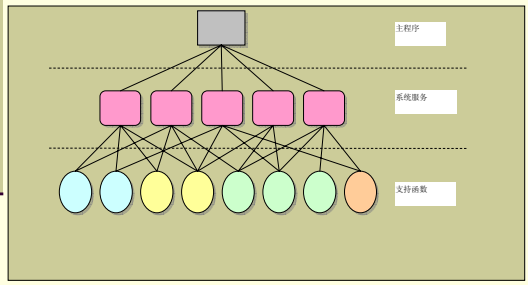
Linux内核体系结构

同济大学
赵炯 (gohigh@sh163.net)
2004.05.10

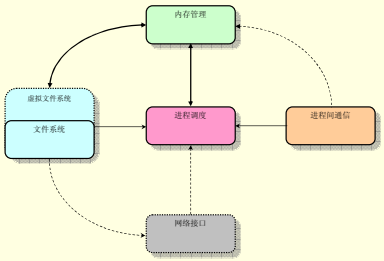
Linux内核体系结构

- Linux内核模式
- Linux内核系统
- 中断机制
- 系统定时
- Linux进程控制
- Linux内核对内存的使用方法
- Linux系统中堆栈的使用方法

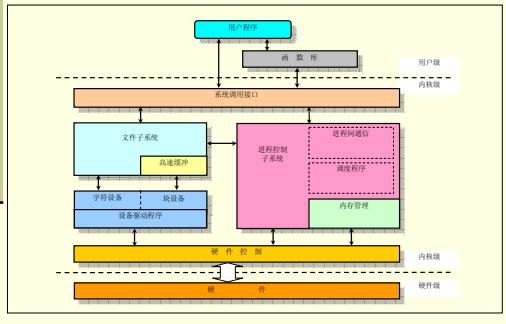
Linux内核模式



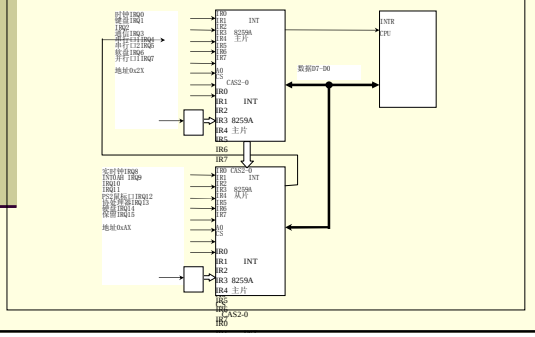
Linux内核系统



Linux内核系统



中断机制



系统定时(1)

PC机的可编程定时芯片Intel 8253被设置成每隔10毫秒就发出一个时钟中断（IRQ0）信号。这个时间节拍就是系统运行的脉搏，我们称之为1个系统滴答。因此每经过1个滴答就会调用一次时钟中断处理程序（timer_interrupt）。该处理程序主要用来通过jiffies变量来累计自系统启动以来经过的时钟滴答数。每当发生一次时钟中断该值就增1。然后从被中断程序的段选择符中取得当前特权级CPL作为参数调用do_timer()函数。

系统定时（2）

do_timer()函数则根据特权级对当前进程运行时间作累计。如果CPL=0，则表示进程是运行在内核态时被中断，因此把进程的内核运行时间统计值stime增1，否则把进程用户态运行时间统计值增1。如果程序添加过定时器，则对定时器链表进行处理。若某个定时器时间到（递减后等于0），则调用该定时器的处理函数。然后对当前进程运行时间进行处理，把当前进程运行时间片减1。如果此时当前进程时间片并还大于0，表示其时间片还没有用完，于是就退出do_timer()继续运行当前进程。

系统定时（3）

如果此时进程时间片已经递减为0，表示该进程已经用完了此次使用CPU的时间片，于是程序就会根据被中断程序的级别来确定进一步处理的方法。若被中断的当前进程是工作在用户态的（特权级别大于0），则do_timer()就会调用调度程序schedule()切换到其它进程去运行。如果被中断的当前进程工作在内核态，也即在内核程序中运行时被中断，则do_timer()会立刻退出。因此这样的处理方式决定了Linux系统在内核态运行时不会被调度程序切换。内核态程序是不可抢占的，但当处于用户态程序中运行时则是可以被抢占的。

Linux进程控制

程序是一个可执行的文件，而进程（process）是一个执行中的程序实例。利用分时技术，在Linux操作系统上同时可以运行多个进程。分时技术的基本原理是把CPU的运行时间划分成一个个规定长度的时间片，让每个进程在一个时间片内运行。当进程的时间片用完时系统就利用调度程序切换到另一个进程去运行。因此实际上对于具有单个CPU的机器来说某一时刻只能运行一个进程。但由于每个进程运行的时间片很短（例如15个系统滴答=150毫秒），所以表面看来好象所有进程在同时运行着。

Linux进程控制 （1）

除了第一个进程是“手工”建立以外，其余的都是进程使用系统调用fork创建的新进程，被创建的进程称为子进程（child process），创建者，则称为父进程（parent process）。内核程序使用进程标识号（process ID，pid）来标识每个进程。进程由可执行的指令代码、数据和堆栈区组成。进程中的代码和数据部分分别对应一个执行文件中的代码段、数据段。每个进程只能执行自己的代码和访问自己的数据及堆栈区。进程之间相互之间的通信需要通过系统调用来进行。对于只有一个CPU的系统，在某一时刻只能有一个进程正在运行。内核通过调度程序分时调度各个进程运行。

Linux进程控制 （2）

Linux系统中，一个进程可以在内核态（kernel mode）或用户态（user mode）下执行，因此，Linux内核堆栈和用户堆栈是分开的。用户堆栈用于进程在用户态下临时保存调用函数的参数、局部变量等数据。内核堆栈则含有内核程序执行函数调用时的信息。

Linux进程控制 – 任务数据结构(1)

内核程序通过进程表对进程进行管理，每个进程在进程表中占有一项。在Linux系统中，进程表项是一个task_struct任务结构指针。任务数据结构定义在头文件include/linux/sched.h中。有些书上称其为进程控制块PCB（Process Control Block）或进程描述符PD（Processor Descriptor）。其中保存着用于控制和管理进程的所有信息。主要包括进程当前运行的状态信息、信号、进程号、父进程号、运行时间累计值、正在使用的文件和本任务的局部描述符以及任务状态段信息。

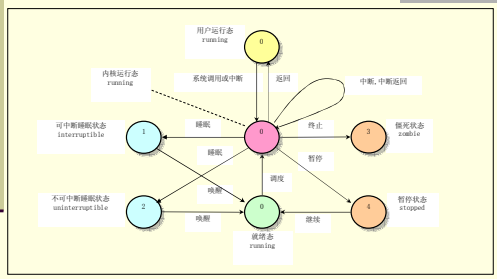
Linux进程控制 – 任务数据结构(2)

当一个进程在执行时，CPU的所有寄存器中的值、进程的状态以及堆栈中的内容被称为该进程的上下文。当内核需要切换（switch）至另一个进程时，它就需要保存当前进程的所有状态，也即保存当前进程的上下文，以便在再次执行该进程时，能够恢复到切换时的状态执行下去。在Linux中，当前进程上下文均保存在进程的任务数据结构中。在发生中断时，内核就在被中断进程的上下文中，在内核态下执行中断服务例程。但同时会保留所有需要用到的资源，以便中断服务结束时能恢复被中断进程的执行。

Linux进程控制 –进程运行状态(1)

一个进程在其生存期内，可处于一组不同的状态下，称为进程状态。进程状态保存在进程任务结构的state字段中。当进程正在等待系统中的资源而处于等待状态时，则称其处于睡眠等待状态。在Linux系统中，睡眠等待状态被分为可中断的和不可中断的等待状态。

Linux进程控制 –进程运行状态(2)



Linux进程控制 –进程运行状态(3)

◆运行状态（TASK_RUNNING）
当进程正在被CPU执行，或已经准备就绪随时可由调度程序执行，则称该进程为处于运行状态（running）。进程可以在内核态运行，也可以在用户态运行。当系统资源已经可用时，进程就被唤醒而进入准备运行状态，该状态称为就绪态。这些状态（图中中间一列）在内核中表示方法相同，都被成为处于TASK_RUNNING状态。
◆可中断睡眠状态（TASK_INTERRUPTIBLE）
当进程处于可中断等待状态时，系统不会调度该进行执行。当系统产生一个中断或者释放了进程正在等待的资源，或者进程收到一个信号，都可以唤醒进程转换到就绪状态（运行状态）。

Linux进程控制 –进程运行状态(4)

◆暂停状态（TASK_STOPPED）
当进程收到信号SIGSTOP、SIGTSTP、SIGTTIN或SIGTTOU时就会进入暂停状态。可向其发送SIGCONT信号让进程转换到可运行状态。
◆僵死状态（TASK_ZOMBIE）
当进程已停止运行，但其父进程还没有询问其状态时，则称该进程处于僵死状态。
◆不可中断睡眠状态（TASK_UNINTERRUPTIBLE）
与可中断睡眠状态类似。但处于该状态的进程只有被使用wake_up()函数明确唤醒时才能转换到可运行的就绪状态。

Linux进程控制 –进程运行状态(5)

当一个进程的运行时间片用完，系统就会使用调度程序强制切换到其它的进程去执行。另外，如果进程在内核态执行时需要等待系统的某个资源，此时该进程就会调用sleep_on()或sleep_on_interruptible()自愿地放弃CPU的使用权，而让调度程序去执行其它进程。进程则进入睡眠状态（TASK_UNINTERRUPTIBLE或TASK_INTERRUPTIBLE）。

只有当进程从“内核运行态”转移到“睡眠状态”时，内核才会进行进程切换操作。在内核态下运行的进程不能被其它进程抢占，而且一个进程不能改变另一个进程的状态。为了避免进程切换时造成内核数据错误，内核在执行临界区代码时会禁止一切中断。

Linux进程控制 – 进程初始化(1)

在boot/目录中引导程序把内核从磁盘上加载到内存中，并让系统进入保护模式下运行后，就开始执行系统初始化程序init/main.c。该程序首先确定如何分配使用系统物理内存，然后调用内核各部分的初始化函数分别对内存管理、中断处理、块设备和字符设备、进程管理以及硬盘和软盘硬件进行初始化处理。在完成了这些操作之后，系统各部分已经处于可运行状态。此后程序把自己“手工”移动到任务0（进程0）中运行，并使用fork()调用首次创建出进程1。在进程1中程序将继续进行应用环境的初始化并执行shell登录程序。而原进程0则会在系统空闲时被调度执行，此时任务0仅执行pause()系统调用，并又会调用调度函数。

Linux进程控制 – 进程初始化(2)

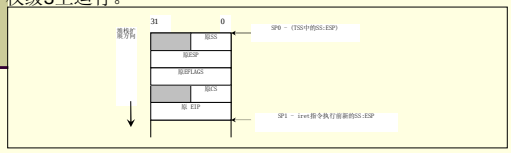
“移动到任务0中执行”这个过程由宏move_to_user_mode（include/asm/system.h）完成。它把main.c程序执行流从内核态（特权级0）移动到了用户态（特权级3）的任务0中继续运行。在移动之前，系统在对调度程序的初始化过程（sched_init()）中，首先对任务0的运行环境进行了设置。这包括人工预先设置好任务0数据结构各字段的值（include/linux/sched.h）、在全局描述符表中添入任务0的任务状态段（TSS）描述符和局部描述符表（LDT）的段描述符，并把它们分别加载到任务寄存器tr和局部描述符表寄存器ldtr中。

Linux进程控制 – 进程初始化(3)

需要强调的是，内核初始化是一个特殊过程，内核初始化代码也即是任务0的代码。任务0的代码段和数据段分别包含在内核代码段和数据段中。内核初始化程序main.c也即是任务0中的代码，只是在移动到任务0之前系统正以内核态特权级0运行着main.c程序。宏move_to_user_mode的功能就是把运行特权级从内核态的0级变换到用户态的3级，但是仍然继续执行原来的代码指令流。

Linux进程控制 – 进程初始化(4)

在移动到任务0的过程中，宏move_to_user_mode使用了中断返回指令造成特权级改变的方法。该方法的主要思想是在堆栈中构筑中断返回指令需要的内容，把返回地址的段选择符设置成任务0代码段选择符，其特权级为3。此后执行中断返回指令iret时将导致系统CPU从特权级0跳转到外层的特权级3上运行。



Linux进程控制 – 进程初始化(5)

当执行iret指令时，CPU把返回地址送入CS:EIP中，同时弹出堆栈中标志寄存器内容。由于CPU判断出目的代码段的特权级是3，与当前内核态的0级不同。于是CPU会把堆栈中的堆栈段选择符和堆栈指针弹出到SS:ESP中。由于特权级发上了变化，段寄存器DS、ES、FS和GS的值变得无效，此时CPU会把这些段寄存器清零。因此在执行了iret指令后需要重新加载这些段寄存器。此后，系统就开始以特权级3运行在任务0的代码上。所使用的用户态堆栈还是原来在移动之前使用的堆栈。而其内核态堆栈则被指定为其任务数据结构所在页面的顶端开始（PAGE_SIZE + (long)&init_task）。

Linux进程控制 –创建新进程 (1)

Linux系统中创建新进程使用fork()系统调用。所有进程都是通过复制进程0而得到的，都是进程0的子进程。

在创建新进程的过程中，系统首先在任务数组中找出一个还没有被任何进程使用的空项（空槽）。然后系统为新建进程在主内存区中申请一页内存来存放其任务数据结构信息，并复制当前进程任务数据结构中的所有内容作为新进程任务数据结构的模板。为了防止这个还未处理完成的新建进程被调度函数执行，此时应该立刻将新进程状态置为不可中断的等待状态（TASK_UNINTERRUPTIBLE）。

Linux进程控制 –创建新进程 (2)

随后对复制的任务数据结构进行修改。把当前进程设置为新进程的父进程，清除信号位图并复位新进程各统计值，并设置初始运行时间片值为15个系统滴答数（150毫秒）。接着根据当前进程设置任务状态段（TSS）中各寄存器的值。由于创建进程时新进程返回值应为0，所以需要设置tss.eax = 0。新建进程内核态堆栈指针tss.esp0被设置成新进程任务数据结构所在内存页面的顶端，而堆栈段tss.ss0被设置成内核数据段选择符。tss.ldt被设置为局部表描述符在GDT中的索引值。如果当前进程使用了协处理器，把还需要把协处理器的完整状态保存到新进程的tss.i387结构中。

Linux进程控制 –创建新进程 (3)

此后系统设置新任务的代码和数据段基址、限长并复制当前进程内存分页管理的页表。如果父进程中有文件是打开的，则应将对应文件的打开次数增1。接着在GDT中设置新任务的TSS和LDT描述符项，其中基地址信息指向新进程任务结构中的tss和ldt。最后再将新任务设置成可运行状态并返回新进程号。

Linux进程控制 –进程调度(1)

由前面描述可知，Linux进程是抢占式的。被抢占的进程仍然处于TASK_RUNNING状态，只是暂时没有被CPU运行。进程的抢占发生在进程处于用户态执行阶段，在内核态执行时是不能被抢占的。

为了能让进程有效地使用系统资源，又能使进程有较快的响应时间，就需要对进程的切换调度采用一定的调度策略。通常采用基于优先级排队的调度策略。

Linux进程控制 –进程调度(2)

schedule()函数首先扫描任务数组。通过比较每个就绪态（TASK_RUNNING）任务的运行时间递减滴答计数counter的值来确定当前哪个进程运行的时间最少。哪一个的值大，就表示运行时间还不长，于是就选中该进程，并使用任务切换宏函数切换到该进程运行。

如果此时所有处于TASK_RUNNING状态进程的时间片都已经用完，系统就会根据每个进程的优先权值priority，对系统中所有进程（包括正在睡眠的进程）重新计算每个任务需要运行的时间片值counter。计算的公式是：

$$counter = \frac{counter}{2} + priority$$

Linux进程控制 –进程调度(3)

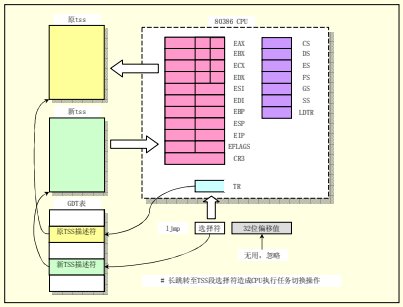
然后schdeule()函数重新扫描任务数组中所有处于TASK_RUNNING状态，重复上述过程，直到选择出一个进程为止。最后调用switch_to()执行实际的进程切换操作。

如果此时没有其它进程可运行，系统就会选择进程0运行。进程0会调用pause()把自己置为可中断的睡眠状态并再次调用schedule()。不过在调度进程运行时，schedule()并不在意进程0处于什么状态。只要系统空闲就调度进程0运行。

Linux进程控制 –进程切换(1)

执行实际进程切换的任务由switch_to()宏定义的一段汇编代码完成。在进行切换之前，switch_to()首先检查要切换到的进程是否就是当前进程，如果是则什么也不做，直接退出。否则就首先把内核全局变量current置为新任务的指针，然后长跳转到新任务的任务状态段TSS组成的地址处，造成CPU执行任务切换操作。此时CPU会把其所有寄存器的状态保存到当前任务寄存器TR中TSS段选择符所指向的当前进程任务数据结构的tss结构中，然后把新任务状态段选择符所指向的新任务数据结构中tss结构中的寄存器信息恢复到CPU中，系统就正式开始运行新切换的任务了。

Linux进程控制 –进程切换(2)



Linux进程控制 –进程终止(1)

当一个进程结束了运行或在半途中终止了运行，那么内核就需要释放该进程所占用的系统资源。这包括进程运行时打开的文件、申请的内存等。

Linux进程控制 –进程终止(2)

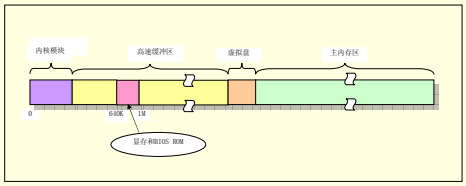
当一个用户程序调用exit()系统调用时，就会执行内核函数do_exit()。该函数会首先释放进程代码段和数据段占用的内存页面，关闭进程打开着的所有文件，对进程使用的当前工作目录、根目录和运行程序的i节点进行同步操作。如果进程有子进程，则让init进程作为其所有子进程的父进程。如果进程是一个会话头进程并且有控制终端，则释放控制终端，并向属于该会话的所有进程发送挂断信号SIGHUP，这通常会终止该会话中的所有进程。然后把进程状态置为僵死状态TASK_ZOMBIE。并向其原父进程发送SIGCHLD信号，通知其某个子进程已经终止。最后do_exit()调用调度函数去执行其它进程。由此可见在进程被终止时，它的任务数据结构仍然保留着。因为其父进程还需要使用其中的信息。

Linux进程控制 –进程终止(3)

在子进程在执行期间，父进程通常使用wait()或waitpid()函数等待其某个子进程终止。当等待的子进程被终止并处于僵死状态时，父进程就会把子进程运行所使用的时间累加到自己进程中。最终释放已终止子进程任务数据结构所占用的内存页面，并置空子进程在任务数组中占用的指针项。

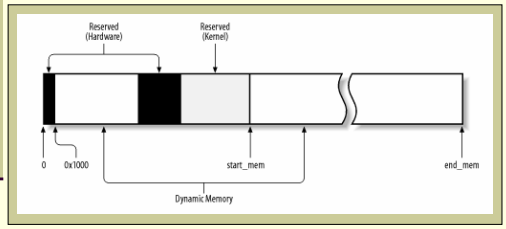
Linux内核对内存的使用方法 (1)

Old方法：

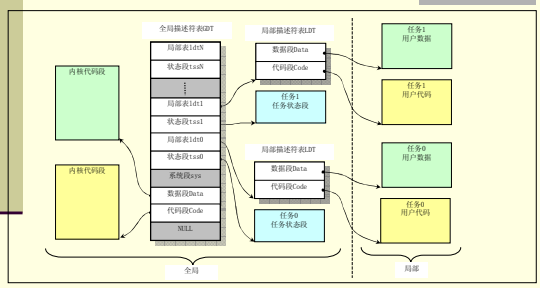


Linux内核对内存的使用方法 (2)

New方法：



Linux内核对内存的使用方法 (3)



Linux系统中堆栈的使用方法 (1)

Linux 系统中共使用了四种堆栈。一种是系统初始化时临时使用的堆栈；一种是供内核程序自己使用的堆栈（内核堆栈），只有一个，位于系统地址空间固定的位置，也是后来任务0的用户态堆栈；另一种是每个任务通过系统调用，执行内核程序时使用的堆栈，我们称之为任务的内核态堆栈，每个任务都有自己独立的内核态堆栈；最后一种是任务在用户态执行的堆栈，位于任务（进程）地址空间的末端。

Linux系统中堆栈的使用方法 (2)

开机初始化时(bootsect.s，setup.s)

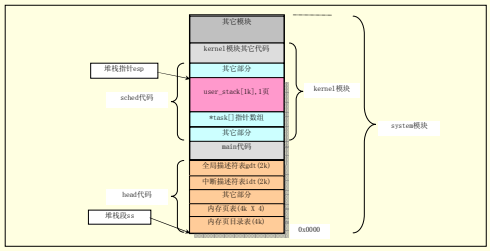
当bootsect代码被ROM BIOS引导加载到物理内存0x7c00处时，并没有设置堆栈段，当然程序也没有使用堆栈。直到bootsect被移动到0x9000:0处时，才把堆栈段寄存器SS设置为0x9000，堆栈指针esp寄存器设置为0xff00，也即堆栈顶端在0x9000:0xff00处。setup.s程序中也沿用了bootsect中设置的堆栈段。这就是系统初始化时临时使用的堆栈。

Linux系统中堆栈的使用方法 (3)

进入保护模式时(head.s)

从head.s程序起，系统开始正式在保护模式下运行。此时堆栈段被设置为内核数据段，堆栈指针esp设置成指向user_stack数组的顶端保留了2页内存（8K）作为堆栈使用。user_stack数组定义在sched.c。此时该堆栈是内核程序自己使用的堆栈。

Linux系统中堆栈的使用方法 (4)



Linux系统中堆栈的使用方法 (5)

初始化时(main.c)

在main.c中，在执行move_to_user_mode()代码之前，系统一直使用上述堆栈。而在执行过move_to_user_mode()之后，main.c的代码被“切换”成任务0中执行。通过执行fork()系统调用，main.c中的init()将在任务1中执行，并使用任务1的堆栈。而main()本身则在被“切换”成为任务0后，仍然继续使用上述内核程序自己的堆栈作为任务0的用户态堆栈。关于任务0所使用堆栈的详细描述见后面说明。

Linux系统中堆栈的使用方法 (6)

任务的堆栈

每个任务都有两个堆栈，分别用于用户态和内核态程序的执行，并且分别称为用户态堆栈和内核态堆栈。这两个堆栈之间的主要区别在于任务的内核态堆栈很小，所保存的数据量最多不能超过（8096 – 任务数据结构）个字节，大约为6K字节。而任务的用户态堆栈却可以在用户线性空间内延伸。

Linux系统中堆栈的使用方法 (7)

在用户态运行时

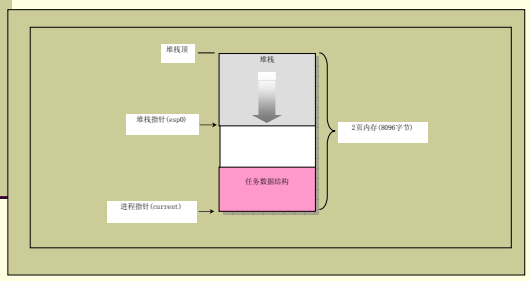
每个任务（除了任务0）有自己的地址空间。当一个任务（进程）刚被创建时，它的用户态堆栈指针被设置在其地址空间的末端，而其内核态堆栈则被设置成位于其任务数据结构所在页面的末端。应用程序在用户态下运行时就一直使用这个堆栈。堆栈实际使用的物理内存则由CPU分页机制确定。由于Linux实现了写时复制功能（Copy on Write），因此在进程被创建后，若该进程及其父进程没有使用堆栈，则两者共享同一堆栈对应的物理内存页面。

Linux系统中堆栈的使用方法 (8)

在内核态运行时

每个任务有其自己的内核态堆栈，与每个任务的任务数据结构（task_struct）放在同一页面内。这是在建立新任务时，fork()程序在任务tss段的内核级堆栈字段(tss.esp0和tss.ss0)中设置的。
内核为新任务申请内存用作保存其task_struct结构数据，而tss结构（段）是task_struct中的一个字段。该任务的内核堆栈段值tss.ss0也被设置成内核数据段，而tss.esp0则指向保存task_struct结构页面的末端。

Linux系统中堆栈的使用方法 (9)

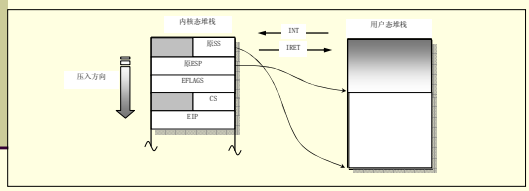


Linux系统中堆栈的使用方法 (10)

任务内核态堆栈与用户态堆栈之间的切换

任务调用系统调用时就会进入内核，执行内核代码。此时内核代码就会使用该任务的内核态堆栈进行操作。当进入内核程序时，由于优先级别发生了改变（从用户态转到内核态），用户态堆栈的堆栈段和堆栈指针以及eflags会被保存在任务的内核态堆栈中。而在执行iret退出内核程序返回到用户程序时，将恢复用户态的堆栈和eflags。

Linux系统中堆栈的使用方法 (11)



End of ppt. Thank you! ☺

www.oldlinux.org
gohigh@sh163.net