**task\_struct 数据结构**[收藏](JavaScript:d=document;t=d.selection?(d.selection.type!='None'?d.selection.createRange().text:''):(d.getSelection?d.getSelection():'');void(saveit=window.open('http://wz.csdn.net/storeit.aspx?t='+escape(d.title)+'&u='+escape(d.location.href)+'&c='+escape(t),'saveit','scrollbars=no,width=590,height=300,left=75,top=20,status=no,resizable=yes'));saveit.focus();)

在linux 中每一个进程都由task\_struct 数据结构来定义. task\_struct就是我们通常所说的PCB.她是对进程控制的唯一手段也是最有效的手段. 当我们调用fork() 时, 系统会为我们产生一个task\_struct结构。然后从父进程,那里继承一些数据, 并把新的进程插入到进程树中, 以待进行进程管理。因此了解task\_struct的结构对于我们理解任务调度(在linux 中任务和进程是同一概念)的关键。

    在进行剖析task\_struct的定义之前，我们先按照我们的理论推一下它的结构：

1、进程状态 ,将纪录进程在等待,运行,或死锁

2、调度信息, 由哪个调度函数调度,怎样调度等

3、进程的通讯状况

4、因为要插入进程树,必须有联系父子兄弟的指针, 当然是task\_struct型

5、时间信息, 比如计算好执行的时间, 以便cpu 分配

6、标号 ,决定改进程归属

7、可以读写打开的一些文件信息

8、 进程上下文和内核上下文

9、处理器上下文

10、内存信息

因为每一个PCB都是这样的, 只有这些结构, 才能满足一个进程的所有要求。打开/include/linux/sched.h可以找到task\_struct 的定义

struct task\_struct {

volatile long state; /\*说明了该进程是否可以执行,还是可中断等信息\*/  
unsigned long flags; /\*Flage 是进程号,在调用fork()时给出\*/  
int sigpending; /\*进程上是否有待处理的信号\*/  
mm\_segment\_t addr\_limit;  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\*\*进程地址空间,区分内核进程与普通进程在内存存放的位置不同\*/  
/\*\*\*\*0-0xBFFFFFFF for user-thead    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\*\*\*\*0-0xFFFFFFFF for kernel-thread \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
volatile long need\_resched;  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\*\*\*\*\*\*\*\*\*\*调度标志,表示该进程是否需要重新调度,\*\*\*\*\*\*\*\*\*\*\*\*/  
/\*\*\*\*\*\*\*\*\*\*若非0,则当从内核态返回到用户态,会发生调度\*\*\*\*\*\*\*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int lock\_depth; /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*锁深度\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
long nice; /\*\*\*\*\*\*\*\*\*\*\*\*\*进程的基本时间片\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
unsigned long policy;  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\*进程的调度策略,有三种\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\*实时进程:SCHED\_FIFO,SCHED\_RR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\*分时进程:SCHED\_OTHER\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
struct mm\_struct \*mm; //进程内存管理信息   
int processor;  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\*若进程不在任何CPU上运行，  
/\*cpus\_runnable 的值是0，否则是1。  
/\*这个值在运行队列被锁时更新.\*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
unsigned long cpus\_runnable, cpus\_allowed;  
struct list\_head run\_list; /\*\*\*\*指向运行队列的指针\*\*\*\*\*\*\*\*\*/  
unsigned long sleep\_time; /\*\*\*\*\*进程的睡眠时间\*\*\*\*\*\*\*\*\*\*\*\*\*/  
struct task\_struct \*next\_task, \*prev\_task;  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\*用于将系统中所有的进程连成一个双向循环链表\*/  
/\*其根是init\_task.\*/   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
struct mm\_struct \*active\_mm;   
struct list\_head local\_pages;/\*\*指向本地页面\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
unsigned int allocation\_order, nr\_local\_pages;  
struct linux\_binfmt \*binfmt;/\*进程所运行的可执行文件的格式\*/   
int exit\_code, exit\_signal;   
int pdeath\_signal;/\*父进程终止是向子进程发送的信号\*\*\*\*\*\*\*\*\*/   
unsigned long personality;  
/\*Linux可以运行由其他UNIX操作系统生成的符合iBCS2标准的程序\*/  
int did\_exec:1;   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\*按POSIX要求设计的布尔量,区分进程正在执行从\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\*父进程中继承的代码,还是执行由execve装入的新程序代码\*\*\*\*\*\*/   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
pid\_t pid;/\*\*\*\*\*\*\*\*\*\*进程标识符,用来代表一个进程\*\*\*\*\*\*\*\*\*\*\*/  
pid\_t pgrp;/\*\*\*\*\*\*\*\*进程组标识,表示进程所属的进程组\*\*\*\*\*\*\*\*/  
pid\_t tty\_old\_pgrp;/\*\*\*\*\*\*\*进程控制终端所在的组标识\*\*\*\*\*\*\*\*/  
pid\_t session;/\*\*\*\*\*\*\*\*\*\*\*\*\*进程的会话标识\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
pid\_t tgid;  
int leader; /\*\*\*\*\*\*\*\*\*\*\*\*\*标志,表示进程是否为会话主管\*\*\*\*\*\*/  
struct task\_struct \*p\_opptr,\*p\_pptr,\*p\_cptr,\*p\_ysptr,\*p\_osptr;   
struct list\_head thread\_group; /\*\*\*\*线程链表\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
struct task\_struct \*pidhash\_next;/\*用于将进程链入HASH表pidhash   
struct task\_struct \*\*pidhash\_pprev;   
wait\_queue\_head\_t wait\_chldexit; /\*供wait4()使用\*\*\*\*\*\*\*\*\*\*\*/   
struct completion \*vfork\_done; /\* 供vfork() 使用\*\*\*\*\*\*\*\*\*\*\*/  
unsigned long rt\_priority;  
/\*\*\*\*实时优先级，用它计算实时进程调度时的weight值，/\*\*\*\*\*\*\*/  
/\*it\_real\_value，it\_real\_incr用于REAL定时器，单位为jiffies\*/  
系统根据it\_real\_value //设置定时器的第一个终止时间。  
在定时器到期时，向进程发送SIGALRM信号，同时根据  
it\_real\_incr重置终止时间，it\_prof\_value，it\_prof\_incr  
用于Profile定时器，单位为jiffies。当进程运行时，  
不管在何种状态下，每个tick都使it\_prof\_value值减一，  
当减到0时，向进程发送信号SIGPROF，并根据it\_prof\_incr重置时间   
it\_virt\_value，it\_virt\_value用于Virtual定时器，单位为jiffies。  
当进程运行时，不管在何种状态下，每个tick都使it\_virt\_value值减一  
当减到0时，向进程发送信号SIGVTALRM，根据it\_virt\_incr重置初值。   
Real定时器根据系统时间实时更新，不管进程是否在运行   
Virtual定时器只在进程运行时，根据进程在用户态消耗的时间更新   
Profile定时器在进程运行时，根据进程消耗的时  
（不管在用户态还是内核态）更新\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/   
unsigned long it\_real\_value, it\_prof\_value, it\_virt\_value;   
unsigned long it\_real\_incr, it\_prof\_incr, it\_virt\_value;   
struct timer\_list real\_timer;//指向实时定时器的指针   
struct tms times; //记录进程消耗的时间，   
unsigned long start\_time;//进程创建的时间   
long per\_cpu\_utime[NR\_CPUS], per\_cpu\_stime[NR\_CPUS]; //记录进程在每个CPU上所消耗的用户态时间和核心态时间   
/\* mm fault and swap info: this can arguably be seen as either mm-specific or thread-specific \*/   
//内存缺页和交换信息：   
//min\_flt, maj\_flt累计进程的次缺页数（Copy on　Write页和匿名页）和主缺页数（从映射文件或交换设备读入的页面数）；   
//nswap记录进程累计换出的页面数，即写到交换设备上的页面数。   
//cmin\_flt, cmaj\_flt, cnswap记录本进程为祖先的所有子孙进程的累计次缺页数，主缺页数和换出页面数。在父进程   
//回收终止的子进程时，父进程会将子进程的这些信息累计到自己结构的这些域中   
unsigned long min\_flt, maj\_flt, nswap, cmin\_flt, cmaj\_flt, cnswap;   
int swappable:1; //表示进程的虚拟地址空间是否允许换出   
/\* process credentials \*////进程认证信息   
//uid,gid为运行该进程的用户的用户标识符和组标识符，通常是进程创建者的uid，gid //euid，egid为有效uid,gid   
//fsuid，fsgid为文件系统uid,gid，这两个ID号通常与有效uid,gid相等，在检查对于文件系统的访问权限时使用他们。   
//suid，sgid为备份uid,gid   
uid\_t uid,euid,suid,fsuid;   
gid\_t gid,egid,sgid,fsgid;   
int ngroups; //记录进程在多少个用户组中   
gid\_t groups[NGROUPS]; //记录进程所在的组   
kernel\_cap\_t cap\_effective, cap\_inheritable, cap\_permitted;//进程的权能，分别是有效位集合，继承位集合，允许位集合   
int keep\_capabilities:1;   
struct user\_struct \*user;   
/\* limits \*/   
struct rlimit rlim[RLIM\_NLIMITS]; //与进程相关的资源限制信息   
unsigned short used\_math; //是否使用FPU   
char comm[16]; //进程正在运行的可执行文件名   
/\* file system info \*///文件系统信息   
int link\_count, total\_link\_count;   
struct tty\_struct \*tty; /\* NULL if no tty 进程所在的控制终端，如果不需要控制终端，则该指针为空\*/   
unsigned int locks; /\* How many file locks are being held \*/   
/\* ipc stuff \*///进程间通信信息   
struct sem\_undo \*semundo; //进程在信号灯上的所有undo操作   
struct sem\_queue \*semsleeping; //当进程因为信号灯操作而挂起时，他在该队列中记录等待的操作   
/\* CPU-specific state of this task \*///进程的CPU状态，切换时，要保存到停止进程的   
task\_struct中   
struct thread\_struct thread;   
/\* filesystem information文件系统信息\*/   
struct fs\_struct \*fs;   
/\* open file information \*///打开文件信息   
struct files\_struct \*files;   
/\* signal handlers \*///信号处理函数   
spinlock\_t sigmask\_lock; /\* Protects signal and blocked \*/   
struct signal\_struct \*sig; //信号处理函数，   
sigset\_t blocked; //进程当前要阻塞的信号，每个信号对应一位   
struct sigpending pending; //进程上是否有待处理的信号   
unsigned long sas\_ss\_sp;   
size\_t sas\_ss\_size;   
int (\*notifier)(void \*priv);   
void \*notifier\_data;   
sigset\_t \*notifier\_mask;/\* Thread group tracking \*/   
u32 parent\_exec\_id;   
u32 self\_exec\_id;   
/\* Protection of (de-)allocation: mm, files, fs, tty \*/   
spinlock\_t alloc\_lock;   
void \*journal\_info;/\* journalling filesystem info \*/  
};