第7章 Linux进程管理

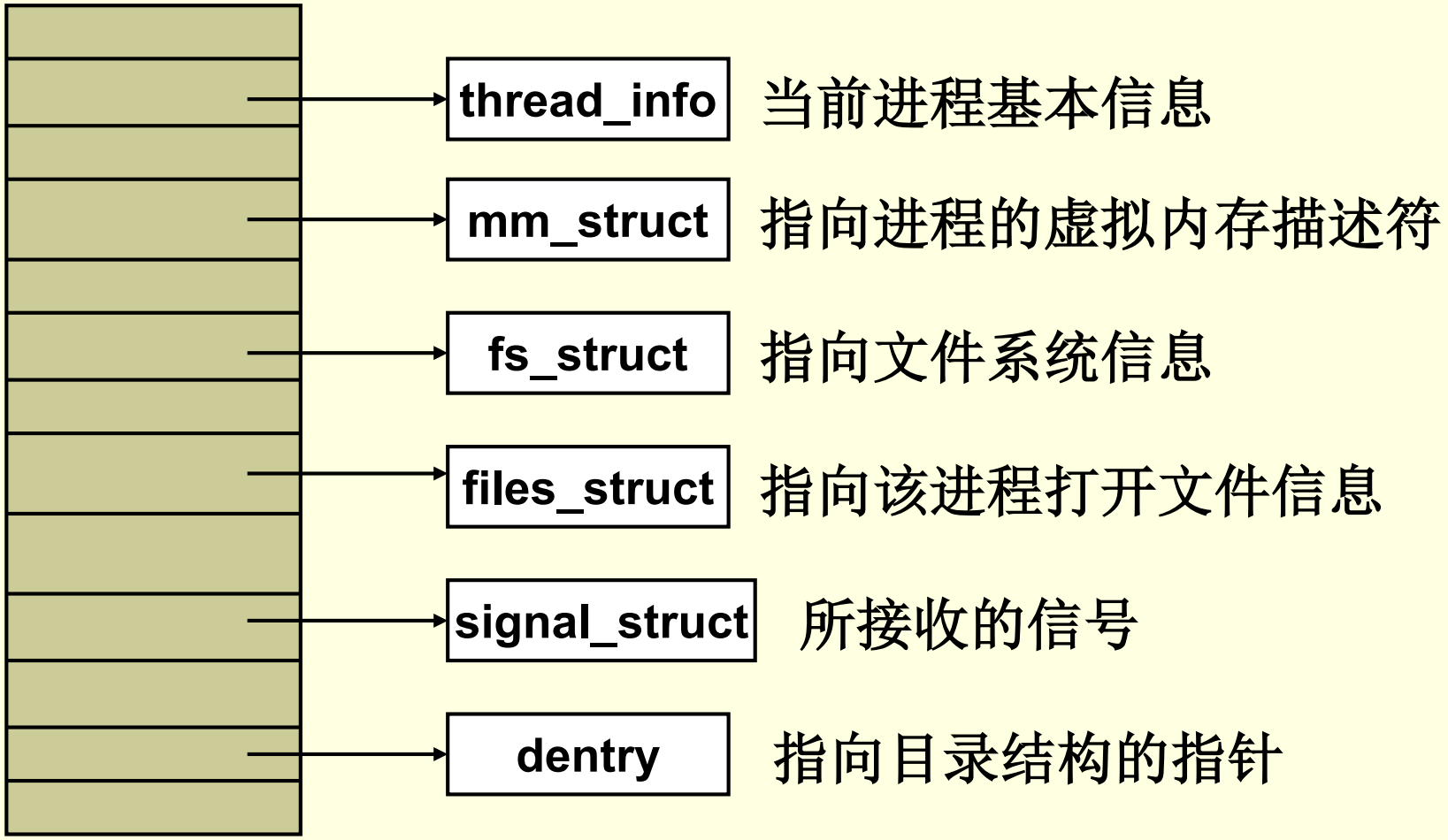
北京理工大学计算机学院

- 现代操作系统允许一个进程有多个执行流，即在相同的地址空间中可执行多个指令序列。
- 每个执行流用一个线程表示，一个进程可以有多个线程。
- Linux使用轻量级进程实现对多线程应用程序的支持，一个轻量级进程就是一个线程。

7.1 Linux进程组成

- 在用户态运行时，进程映像包含有代码段、数据段和私有用户栈。
- 进程在核心态运行时，访问内核的代码段和数据段，并使用各自的核心栈。
- 进程描述符（task_struct）：描述进程的数据结构。

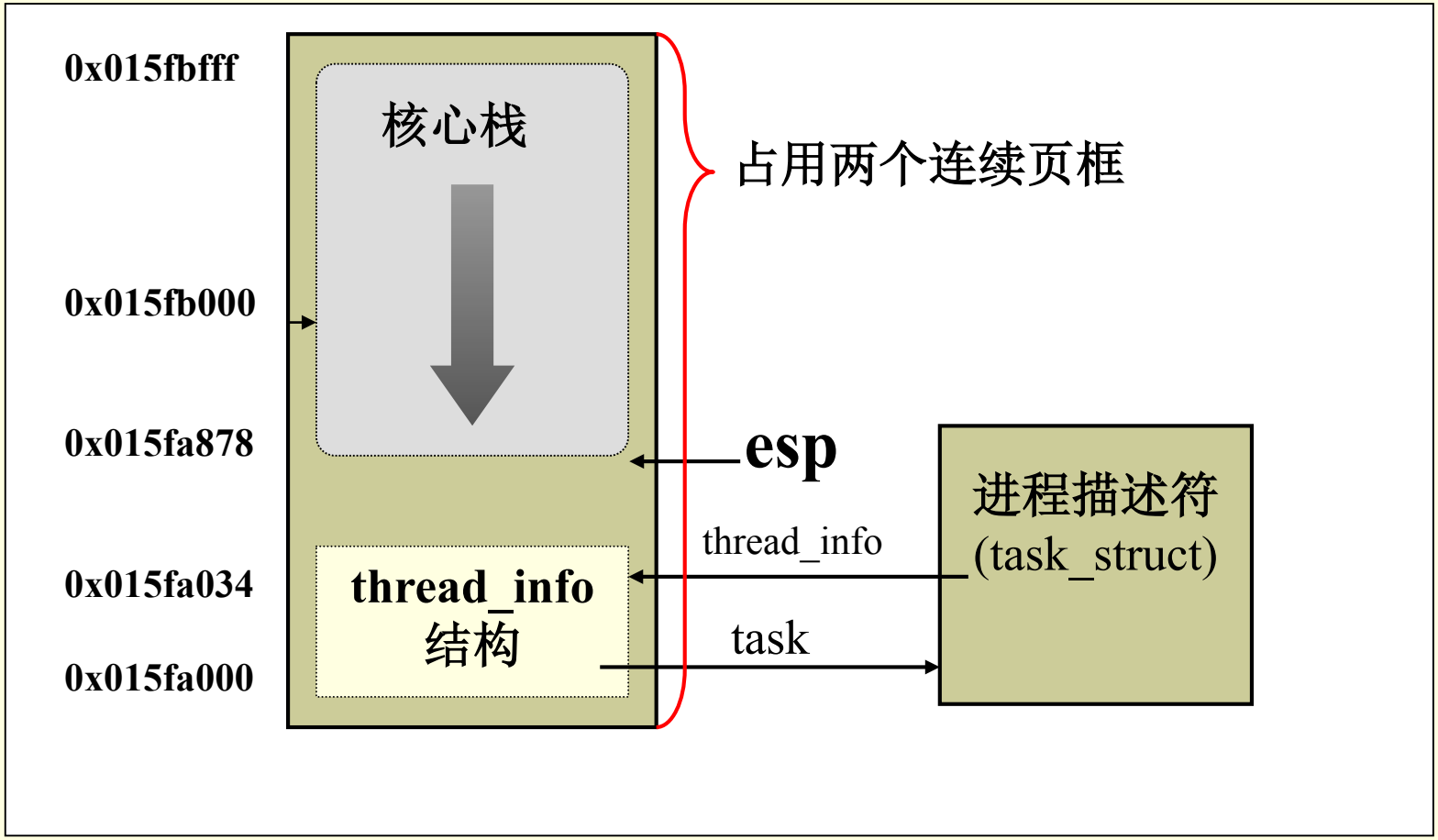
task_struct


thread_info
当前进程基本信息
mm_struct
指向进程的虚拟内存描述符
fs_struct
指向文件系统信息
files_struct
指向该进程打开文件信息
signal_struct
所接收的信号
dentry
指向目录结构的指针

PID

- 每个进程都有一个唯一的标识符PID。新创建的进程的PID通常是前一个进程PID加1。默认情况下，最大的PID号是32767。
- 为了循环使用PID号，内核通过一个pidmap_array位图来管理PID号。
- pid_t tgid; 线程组标识符
- struct pid pids[ ]; 用于在PID哈希表中构造进程链表。

进程核心栈



进程核心栈、thread_info结构和进程描述符之间的关系

thread_info

存放当前进程基本信息

```
struct thread_info {
    struct task_struct  task; / *进程描述符指针*/
    unsigned long  flags;     /*重调度标志TIF_NEED_RESCHED*/
    struct exec_domain  exec_domain;
    int  preempt_count;    /*软中断计数器*/
    __u32  cpu;
    struct restart_block  restart_block;
};
```

- 由于esp寄存器存放的是核心栈的栈顶指针，内核很容易从esp寄存器的值获得正在CPU上运行的进程的thread_info结构的地址。
- 进程刚从用户态切换到核心态时，其核心栈为空，只要将栈顶指针减去8k，就能得到thread_info结构的地址。

7.1.2 进程的状态

① 可运行状态：进程正在或准备在CPU上运行的状态。

② 可中断的等待状态：进程睡眠等待系统资源可用或收到一个信号后，进程被唤醒。

③ 不可中断的等待状态：进程睡眠等待一个不可被中断的事件的发生。如进程等待设备驱动程序探测设备的状态。

④ 暂停状态；⑤跟踪状态；⑥僵死状态；⑦死亡状态

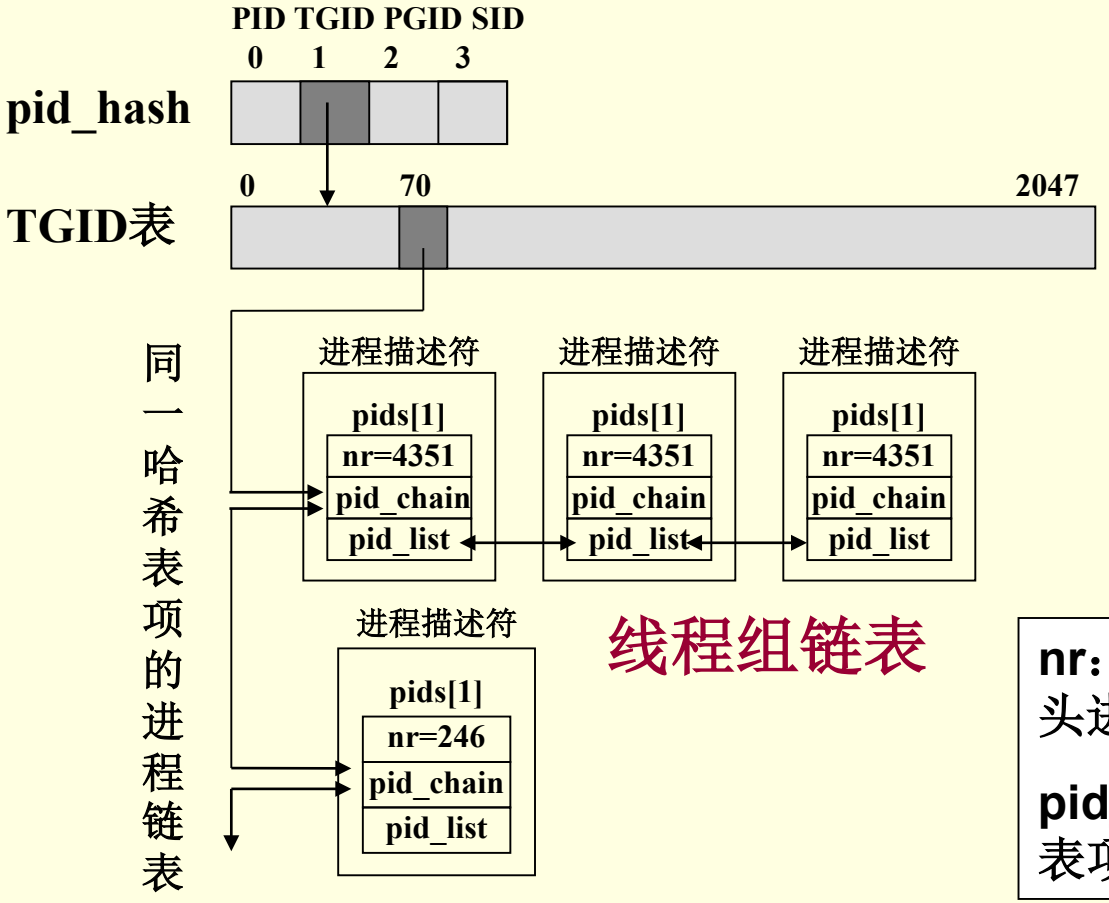
7.2 Linux进程链表

- 传统进程链表
  - ① 所有进程链表：链表头是0号进程
  - ② 可运行进程链表：按照它们的优先级可构建140个可运行进程队列。每个处理机都有自己的可运行进程队列。
  - ③ 子进程链表
  - ④ 兄弟进程链表
  - ⑤ 等待进程链表。互斥等待访问临界资源的进程；非互斥等待的进程，所有进程都被唤醒。

4个哈希链表

- 4个哈希链表：根据进程标识符PID在进程链表中检索进程是可行的，但效率相当低。为了加速对进程的检索，内核定义4类哈希表。

PID哈希表



PID：进程pid

TGID：线程组领头进程的pid

PGID：进程组领头进程的pid

SID：会话领头进程的pid

nr：线程组号，即线程组领头进程的PID

pid_chain：链接同一哈希表项的进程

- 同一线程组中的所有轻量级进程的tgid的值相同。假设内核要回收一个线程组中的所有进程，如果根据线程组号tgid检索哈希表，则只能返回一个线程组领头进程的描述符。
- 为了提高效率，内核在构造哈希表的同时也为每个线程组中的进程创建了一个进程链表。

7.3 Linux进程控制

- 创建进程的函数fork()、clone()、vfork()、kernel_thread()。
- 创建子进程函数fork()：创建成功之后，子进程采用写时复制技术读共享父进程的全部地址空间，仅当父或子要写一个页时，才为其复制一个私有的页的副本。
- 创建轻量级进程函数clone()：实现对多线程应用程序的支持。共享进程在内核的很多数据结构，如页表、打开文件表等等。

- vfork()系统调用：创建的子进程能共享父进程的地址空间，为了防止父进程重写子进程需要的数据，先阻塞父进程的执行，直到子进程退出或执行了一个新的程序为止。
- 创建内核线程函数kernel_thread()：Linux有很多内核线程，运行在内核的上下文中。

0号、1号

- 0号进程就是一个内核线程，0号进程是所有进程的祖先进程，又叫idle进程或叫做swapper进程。其进程描述符存放在init_task变量中。每个CPU都有一个0号进程。
- 1号进程是由0号进程创建的内核线程init，负责完成内核的初始化工作。在系统关闭之前，init进程一直存在，它负责创建和监控在操作系统外层执行的所有用户态进程。

do_fork

- 查看PID位图，为子进程分配一个PID。
- 创建子进程描述符，并把子进程插入到父进程所在的运行队列。
- 若设置了CLONE_STOPPED标志，则把子进程置为暂停状态。
- 如果设置了CLONE_VFORK标志，则挂起父进程直到子进程运行结束或者运行了新的程序。
- 返回子进程的PID。

进程撤销

- exit()系统调用只终止某一个线程。
- exit_group()系统调用能终止整个线程组。
- 父进程先结束的子进程会成为孤儿进程，系统会强迫所有的孤儿进程成为init进程的子进程。Init进程在用wait()类系统调用检查其终止子进程时，就会撤消所有僵死的子进程。

7.4 Linux进程切换

- 进程切换只发生在核心态。在发生进程切换之前，用户态进程使用的所有寄存器值都已被保存在进程的核心栈中。
- 进程硬件上下文存放在进程描述符的thread_struct thread中。该结构包含的字段涉及到大部分CPU寄存器，但象eax、ebx等通用寄存器的值仍被保留在核心栈中。
- 进程切换：第一步，切换页目录表以安装一个新的地址空间；第二步，切换核心栈和硬件上下文。由schedule()函数完成进程切换。

7.5 Linux进程调度

- 进程调度的目标：对实时和交互式进程的响应速度尽可能快；对批处理进程的吞吐量尽可能大。既要考虑到进程的高低优先级，又要尽可能地避免进程的饥饿现象。
- Linux2.6系统采用可抢先式的动态优先级调度方式。其内核是完全可重入的。无论进程处于用户态还是核心态运行，都可能被抢占CPU，从而使高优先级进程能及时被调度执行，不会被处于内核态运行的低优先级进程延迟。

- Linux系统的调度算法是基于进程过去行为的启发式算法，以确定进程应该被当作交互式进程还是批处理进程。
- 根据所属调度类型的不同，进程可分为：先进先出的实时进程、时间片轮转的实时进程、普通的分时进程。
- 实时进程分配的基本优先数为1～99，而交互式进程和批处理进程的基本优先数为100～139。

1. 普通进程的调度

- 在普通的分时进程调度中，要兼顾到基本时间片和动态优先级。
- 新进程总是继承父进程的静态优先级。静态优先数决定了进程的基本时间片值。
- 调度程序会动态调整进程优先级，适当提升在较长时间间隔内没有获得CPU的进程优先级，适当降低已在CPU上运行了较长时间的进程的优先级，以防止出现进程饥饿现象。

- 动态优先数=max(100, min(静态优先数-bonus+5, 139))
- bonus的取值依赖于进程的过去行为，与进程的平均睡眠时间有关。
- 交互式进程：动态优先数≤3×静态优先数/4+28；否则为批处理进程。
- 通过系统调用可改变普通进程的静态优先级。

2. 实时进程的调度

- 先进先出的实时进程调度
- 时间片轮转的实时进程调度
- 实时进程总是活动进程，通过系统调用用户可改变实时进程的优先级。
- 调度时机：（1）出现了更高优先级的实时进程。（2）进程执行了阻塞操作而进入睡眠状态。（3）进程停止运行或被杀死。（4）进程调用自愿放弃处理机。（5）在基于时间片轮转的实时进程调度过程中，进程用完了自己的时间片。

3. 进程调度所涉及的数据结构

- 在多处理机系统中，每个CPU都有自己的可运行队列，存放在结构类型为runqueue的一维数组变量runqueues中。每个CPU对应数组中的一项。
- 活动进程链表：没有用完自己的时间片。
- 过期进程链表：已经用完了自己的时间片。
- 当所有活动进程都过期之后，过期进程才允许运行。

- 在多处理机系统中，为了平衡各CPU之间的负载，内核会将可运行进程从一个运行队列迁移到另一个运行队列。

7.6 内核同步

- UNIX内核的各个组成部分并不是严格按照顺序依次执行的，而是采用交错方式执行的，以响应来自运行进程的请求和来自外部设备的中断请求。
- 因此会出现多个交叉内核控制路径访问内核共享数据结构而引起竞争。

- 内核同步就是确保在任意时刻只有一个内核控制路径处于临界区。
- 内核使用的同步技术包括：每CPU变量、原子操作、优化和内存屏障、自旋锁、读—拷贝—更新、信号量、禁止本地中断、禁止和激活可延迟函数。