Linux内核编程：系统调用

主讲：王利涛

- 基本概念
  - 一个系统调用的例子
  - 什么是系统调用？软件复用的角度
  - 为什么需要系统调用？
  - 学习系统调用有什么意义？

- 本期课程主要内容
  - 系统调用处理流程
  - 系统调用在内核中的实现
  - 如何添加一个系统调用
  - 快速系统调用
  - 虚拟系统调用
  - VDSO
  - 文件读写流程：read、write

01 软中断：系统调用的入口

系统调用的实现

- 权限管理
  - 程序的用户态、内核态
  - 操作系统 + CPU软中断： swi/svc
  - CPU的运行级别： 普通权限、特权
    - ARM32：
      - » 普通模式：User
      - » 特权模式： FIQ、IRQ、SVC、ABT、UND
    - ARM64：EL0、EL1、EL2、EL3
    - X86： ring0~ring3

系统调用的入口

- 系统调用号
  - ARM : swi、svc
  - 系统调用接口：read、write、printf
  - 内核中的实现：sys_read、sys_write
  - 系统调用号：
    - 32位ARM：3、4
    - 64位ARM：0、1

系统调用的入口

- 数据传递
  - 软中断指令：
    - X86 int 80H
    - ARM swi svc
  - 用户函数的参数传递
    - ARM：R0、R1、R2、R3、R4、R5、R6
    - ARM64：X1、X2、X3、X4、X5
  - 系统调用号
    - ARM：R7
    - ARM64：X8
  - 内核函数的返回值
    - ARM：R0
    - ARM64：X0

02 软中断：系统调用的入口(X86)

系统调用的实现(X86)

- 权限管理
  - 操作系统 + X86 CPU软中断： int 80h/syscall
  - 程序的用户态、内核态
  - CPU的运行级别
    - ring0
    - ring1
    - ring2
    - ring3

系统调用的实现(X86)

- 系统调用号
  - 系统调用接口：read、write
  - 内核中的实现：sys_read、sys_write
  - 系统调用号：
    - 32位X86：3、4
    - 64位X86：0、1

系统调用的实现(X86)

- 数据传递
  - 用户函数的参数传递
    - X86-32：ebx、ecx、edx、esi、edi、ebp
    - X86-64：rdi、rsi、rdx、r10、r8、r9
  - 系统调用号
    - X86：eax
    - X86-64：rax
  - 内核函数的返回值
    - X86：eax、edx
    - X86-64：rax、rdx

03 系统调用接口的封装

系统调用接口的封装

- C标准库
  - 包含一系列系统调用接口的封装
    - read、write、fork、open…
  - X86平台
  - ARM平台

04 系统调用接口的封装

系统调用接口的封装

- syscall
  - 在C标准库中没有封装的系统调用
  - syscall是一个库函数
  - 封装了系统调用的汇编接口
    - 系统调用前保存CPU寄存器
    - 从系统调用返回后，恢复寄存器

05 系统调用流程分析

系统调用流程分析

- 系统调用：kill-Linux-5.10-arm-vexpress
  - 接口封装：/usr/arm-linux-gnueabi/lib/libc.a
  - 系统调用号：arch/arm/include/generated/calls-eabi.S
  - 内核实现：kernel/signal.c
  - 中断处理：arch/arm/kernel/entry-common.S

06 添加一个系统调用

添加一个系统调用

- 实验流程
  - 内核版本：5.10.4
  - 增加内核对应的实现函数
  - 在系统调用表中增加一个系统调用号及入口
  - 编写测试程序：在应用层发起系统调用

07 系统调用的开销

系统调用的开销

- 主要开销
  - 中断
  - 抢占系统、任务调度
  - 同步
  - IO等待

系统调用的开销

- 解决之道
  - 快速系统调用： fast system call
  - 虚拟系统调用： vsyscall
  - VDSO

08 快速系统调用

快速系统调用

- X86处理器
  - 专门为系统调用设计的汇编指令
  - Intel：sysenter、sysexit
  - AMD：syscall、sysret
  - 简化了系统调用和返回流程
    - 预加载参数、不做权限检查
    - 不再查表，直接从寄存器取值，实现快速跳转：MSR
    - 不需要保存地址和返回地址等信息

09 虚拟系统调用：vsyscall

虚拟系统调用

- 继续提升系统调用性能
  - int 80H
  - swi/svc
  - sysenter/sysexit
  - syscall/sysret
  - Vsyscall： time

虚拟系统调用

- 编程实验
  - 使用系统调用time获取当前时间
  - 使用虚拟系统调用接口获取当前时间
  - 比较结果并分析

10 虚拟动态共享对象：VDSO

虚拟动态共享对象：VDSO

- vsyscall的局限性
  - 分配的内存有限
  - 只支持4个系统调用
  - 在进程中的位置是静态的、固定的，易受攻击
- VDSO的改进
  - 提供超过4个系统调用
  - 在进程中的地址是随机的: time

虚拟动态共享对象：VDSO

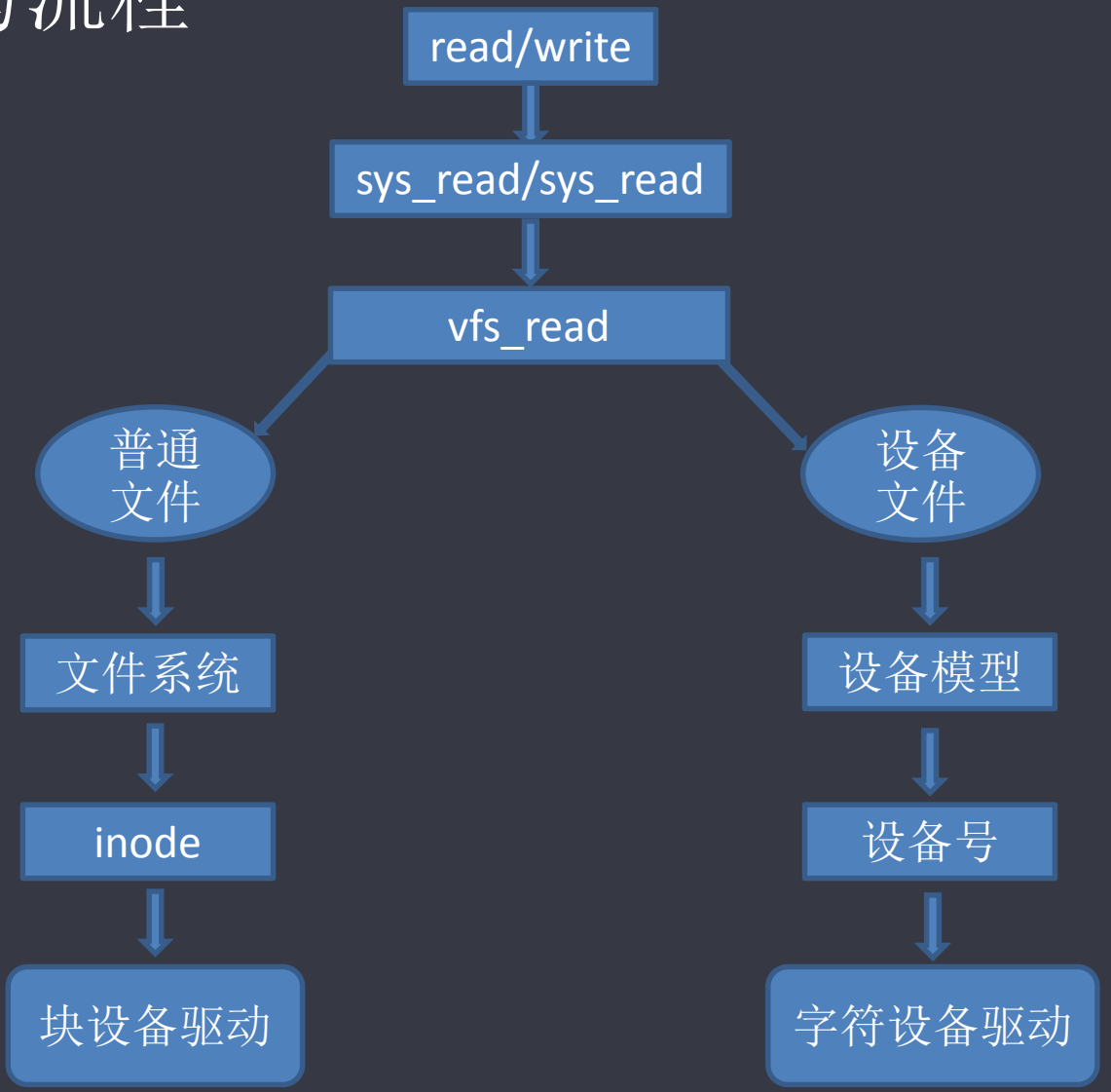
- VDSO： virtual dynamic shared object
  - # cat /proc/self/maps
  - 源码在内核中实现
    - arch/arm/kernel/vdso.c
    - 关键函数：vdso_mremap、install_vvar
  - 速度最快
  - 开销最小，基本等价于函数调用开销
- VVAR： VDSO data page
  - struct vdso_data

虚拟动态共享库：VDSO

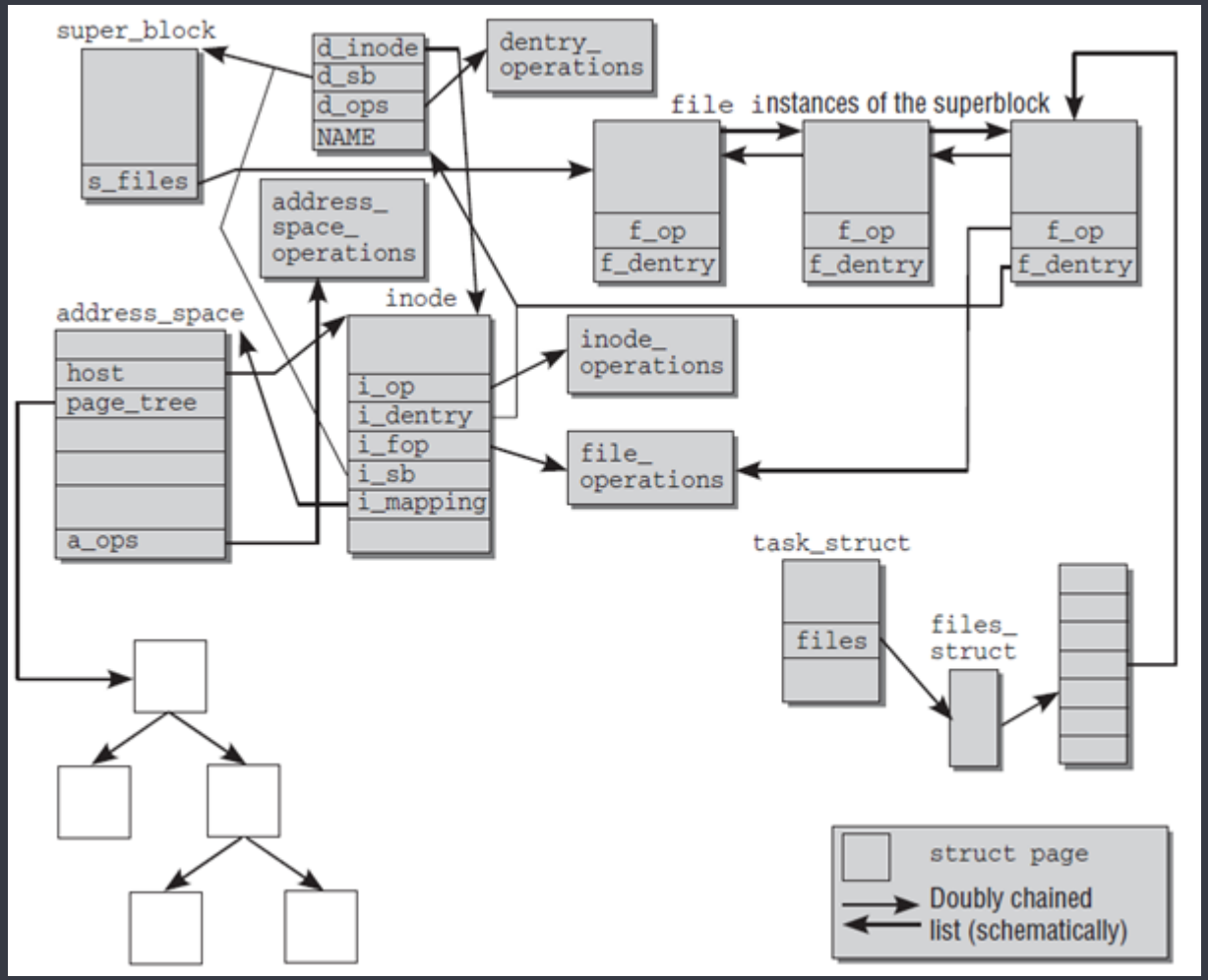
- 实验：反汇编VDSO动态库
  - 将VDSO指令从内存中dump出来
  - 反汇编为汇编代码
  - 分析汇编代码

11 文件的读写流程

- 文件的读写流程



一切皆文件


super_block
s_files
d_inode
d_sb
d_ops
NAME
dentry_
operations
file instances of the superblock
f_op
f_dentry
f_op
f_dentry
f_op
f_dentry
address_
space_
operations
inode
address_space
host
page_tree
a_ops
i_op
i_dentry
i_fop
i_sb
i_mapping
inode_
operations
file_
operations
task_struct
files
files_
struct
struct page
Doubly chained
list (schematically)

更多信息

王利涛老师个人店：https://wanglitao.taobao.com
嵌入式在线教程网：www.zhaixue.cc
嵌入式技术交流群：
宅学部落02群：398294860
宅学部落03群：559671596
宅学部落04群：528718820

欢迎关注公众号：