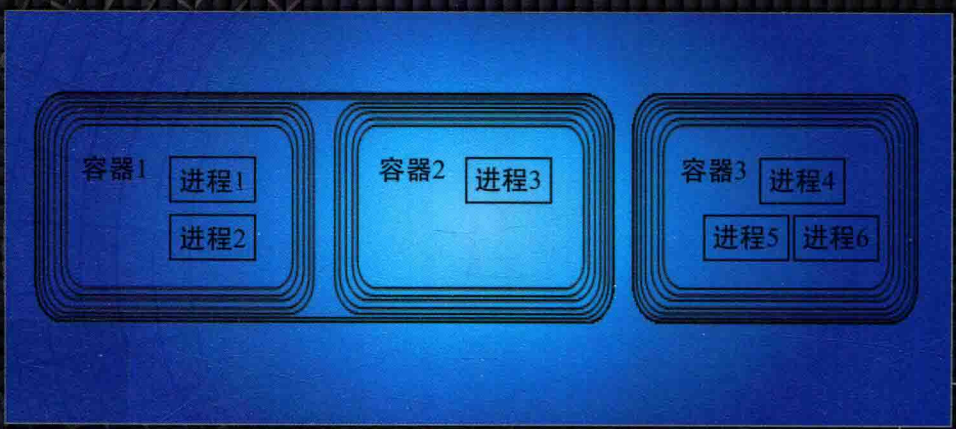
网络与信息安全前沿技术丛书

基于名字空间的安全程序设计

郭玉东 著

Namespaces Based Security Programming



国防工业出版社
National Defense Industry Press

网络与信息安全前沿技术丛书

郭玉东 著

基于名字空间的安全程序设计

Namespaces Based Security Programming

本书主要内容：①概要分析Linux的传统安全机制；②深入分析七类名字空间的工作机理；③探讨基于名字空间的安全程序设计方法；④给出若干程序片段和一个完整的程序示例。

国防工业出版社
National Defense Industry Press
·北京·

图书在版编目(CIP)数据

基于名字空间的安全程序设计 / 郭玉东著. —北京：国防工业出版社，2018.12
（网络与信息安全前沿技术丛书）
ISBN 978-7-118-11771-4

Ⅰ. ①基… Ⅱ. ①郭… Ⅲ. ①计算机网络-网络安全-程序设计-高等学校-教材 Ⅳ. ①TP311.1

中国版本图书馆 CIP 数据核字(2019)第 030063 号

※

国防工业出版社出版发行
（北京市海淀区紫竹院南路 23 号　邮政编码 100048）
天津嘉恒印务有限公司印刷
新华书店经售
*
开本 710×1000　1/16　印张 21½　字数 412 千字
2018 年 12 月第 1 版第 1 次印刷　印数 1—1500 册　定价 128.00 元

（本书如有印装错误，我社负责调换）

国防书店：(010)88540777　　发行邮购：(010)88540776
发行传真：(010)88540755　　发行业务：(010)88540717

谨以此书献给我的妻子刘君和女儿郭刻羽

《网络与信息安全前沿技术丛书》编委会

丛书序

网络的触角正伸向全球各个角落，高速发展的信息技术已渗透到各行各业，不仅推动了产业革命、军事革命，还深刻改变着人们的工作、学习和生活方式。然而，在人们享受信息技术带来巨大利益的同时，一次又一次网络信息安全领域发生的重大事件告诫人们，网络与信息安全已直接关系到国家安全和社会稳定，成为我们面临的新的综合性挑战，没有过硬的技术，没有一支高水平的人才队伍，就不可能在未来国际博弈中赢得主动权。

网络与信息安全是一门跨多个领域的综合性学科，涉及计算机科学、网络技术、通信技术、密码技术、信息安全技术、应用数学、数论、信息论等。“道高一尺，魔高一丈”，网络与信息安全技术在博弈中快速发展，出版一套覆盖面较全、反映网络与信息安全方面新知识、新技术、新发展的丛书有着十分迫切的现实需求。

适逢此时，欣闻由我国网络与信息安全领域著名专家何德全院士任编委会主任，以国家保密通信重点实验室为核心，集聚国内信息安全界知名专家学者，潜心数年编写的“网络与信息安全前沿技术丛书”即将分期出版。丛书有如下3个特点。一是全面系统。丛书涵盖了密码理论与技术、网络与信息安全基础技术、信息安全防御体系，以及近年来快速发展的大数据、云计算、移动互联网、物联网等方面的安全问题。二是适应面宽。丛书既很好地阐述了相关概念、技术原理等基础性知识，又较全面介绍了相关领域前沿技术的最新发展，特别是凝聚了

作者们多年来在该领域从事科技攻关的实践经验，可适应不同层次读者的需求。三是权威性好。编委会由我国网络和信息安全领域权威专家学者组成，各分册作者又均为我国相关领域的知名学者、学术带头人，理论水平高，并有长期科研攻关的丰富积累。

我认为该丛书是一套难得的系统研究网络信息安全技术及应用的综合性书籍，相信丛书的出版既能为公众了解信息安全知识、提升安全防护意识提供很好的选择，又能为从事网络信息安全人才培养的教师和从事相关领域技术攻关的科技工作者提供重要的参考。

作为特别关注网络信息安全技术发展的一名科技人员，我特别感谢何德全院士等专家学者为撰写本丛书付出的艰辛劳动和做出的重要贡献，愿意向读者推荐该套丛书，并作序。

前言

作为一种通用的操作系统，Linux 内核提供了很多基础性的支持机制，以这些机制为基础，可以开发出各种类型的服务程序，满足用户各种类型的需求。为了应对日益严峻的安全形势，Linux 也提供了一些安全支持机制，如身份认证框架、访问控制框架、报文过滤框架、数据变换算法管理框架、地址空间随机化等，利用这些安全机制，可开发出不同种类的安全程序，为用户提供不同类型的安全服务，如身份认证、强制访问、防火墙、加密通信、加密存储等，以提升系统的安全性。除专用的安全机制之外，Linux 中的另一些支持机制，如虚拟化、名字空间、容器等，虽不是专门为安全设计的，但也可用于服务系统或服务程序的安全开发，也能提升系统的安全，其中最具代表性的是名字空间机制。

名字空间（Namespace）是名字的集合。在操作系统中，名字用于标识对象。在早期的版本中，Linux 所管理的对象大都用全局的名字或 ID 号标识，如用于标识用户的 UID、GID，用于标识进程的 PID，用于标识 IPC 对象的 KEY 和 ID 号，用于标识文件的路径名，用于标识网络协议栈的网络设备名、IP 地址、路由表等。全局的对象名可以被系统中的所有进程看到，且各进程看到的名字都是一样的，因而由全局名字所标识的对象也可以被所有的进程访问到。这种全局的、单一的名字空间机制简化了设计，但也带来了安全风险，恶意进程能比较方便地探测、访问系统中的对象，并可通过对对象状态的篡改影响其他进程的运行。为了解决单一名字空间的问题，从 Linux 2.4.19 版开始，陆续引入了多种名字空间机制，试图限制各对象名字的可见范围，将全局的名字转化成局部的名字，进而将全局的对象转化成局部的对象，将全局的资源转化成局部的资源。

目前的 Linux 已提供了 7 种名字空间机制。USER 名字空间用于封装进程可见的用户标识，如 UID、GID 等，实现用户的局部化。UTS 名字空间用于封装进程可见的系统名，如主机名、机器名、域名、操作系统名、版本号等，实现系统平台的局部化。MNT 名字空间用于封装进程可见的文件系

统实体名，如文件路径名，实现文件系统结构的局部化。PID 名字空间用于封装进程可见的进程名，如 PID，实现进程的局部化。IPC 名字空间用于封装进程可见的 IPC 对象名(Key)与 ID 号，实现 IPC 对象的局部化。NET 名字空间用于封装进程可见的网络实体名，如网络设备名、协议地址、路由表、防火墙规则等，实现网络协议栈的局部化。CGROUP 名字空间用于封装进程可见的控制群名，实现控制群的局部化。加上已经局部化的处理器和虚拟内存，Linux 已实现了绝大部分对象或资源的局部化。

在引入名字空间之后，Linux 系统中的所有进程全都运行在名字空间之中。利用目前提供的 7 类名字空间机制，Linux 已可为进程营造出一个虚拟的、相对封闭的运行环境，称为容器(Container)。名字空间是构造容器的基础。事实上，Linux 中的容器仅是一个虚的概念，并不存在称为容器的实体，7 类不同的名字空间实例合在一起就是一个容器，或者说一个容器就是位于 7 类特定名字空间中的一组进程。以名字空间为基础，可以构造出各种形状、各种尺寸、各种寿命的容器。

基于名字空间的容器技术可用于弥合软件开发与部署之间的鸿沟，改变软件开发、交付和运行的方式，降低开发与运维人员之间沟通的复杂性，协助开发出更为强健的软件，提高软件的可靠性、移植性和可扩展性等，是最热门的技术之一。目前已涌现出了一大批相关产品，如 Docker、Kubernets、CoreOS、Mesos、RancherOS 等。

然而，基于名字空间的容器技术还可用于提升软件的安全性。事实上，名字空间和基于名字空间的容器是进程的一种封装方式，或者说是封装在一起的一组进程。与进程组相比，容器具有更好的安全特性，如具有更好的独立性、封装性、隔离性等，而且容器的构建方式灵活，资源消耗量少，启动速度快，也适合于构建更加安全的软件运行环境或开发更加安全的软件。

本书深入分析 Linux 中各名字空间机制的组成结构与工作机理，探讨利用名字空间机制开发安全应用程序、构建安全运行环境的方法。全书由 11 章组成。

第 1 章综述 Linux 操作系统的组成结构，包括内核各个主要子系统的结构及工作机理，并介绍后面各章节将要用到的主要接口函数。

第 2 章分析 Linux 提供的常规安全机制，包括通用的身份认证框架 PAM、基于 UID 和权能的访问控制方法、通用的访问控制框架 LSM 和报文过滤框架 Netfilter、算法管理框架 Crypto API 和基于 Crypto API 的加密通信与加密存储方法、随机化方法、虚拟化方法等。

第 3 章概述 Linux 的名字空间管理结构，讨论 proc 文件系统中的名字空间管理文件，介绍名字空间管理函数和管理命令的使用方法，并给出几个应用实例。

第4章从UID和GID入手，全面分析USER名字空间机制的组成结构和工作机理，探讨USER名字空间对进程权能、进程证书的影响，并介绍USER名字空间接口文件。

第5章分析UTS名字空间的组成结构和工作机理，给出两个应用实例。

第6章从Linux文件系统的目录树、安装树、共享子树入手，全面分析MNT名字空间的组成结构和工作机理，讨论MNT名字空间对文件路径名的影响，介绍MNT名字空间接口文件，并给出若干应用实例。

第7章从进程PID号入手，全面分析PID名字空间的组成结构和工作机理，讨论PID名字空间对进程ID号的影响，介绍PID名字空间接口文件，探讨PID名字空间中的进程运作方式，并给出若干应用实例。

第8章讨论System V和POSIX的进程间通信(IPC)机制，分析IPC名字空间的组成结构和工作机理，介绍IPC名字空间接口文件，并给出若干应用实例。

第9章概述网络协议栈的结构，较为全面地讨论协议栈管理参数、协议栈和防火墙管理命令，全面分析NET名字空间的组成结构和工作机理，介绍NET名字空间管理命令和接口文件，并给出若干管理实例。

第10章从进程与资源入手，较为全面地分析控制群树、限定树的组成结构与工作机理，讨论进程与控制群之间的关系，介绍CGROUP的几个主要子系统的使用方法，探讨CGROUP名字空间的组成结构。

第11章概述名字空间的安全特性，给出一种基于名字空间的动态服务程序框架，并以一个程序实例以验证其可行性。

本书的主要内容来源于Linux内核源代码和Linux操作系统自带的文档，如man手册，是对作者多年教学与科研工作的总结，可作为容器开发与维护人员、安全程序设计人员的参考书，也可用于研究生课程的教材。本书内容翔实，所有实例都在Ubuntu Linux系统中验证通过。

本书的写作过程中得到了信息工程大学网络空间安全学院及教研室领导与同事们的支持，得到了课程组、实验室同行的帮助，在此一并表示感谢。

由于作者水平有限，书中难免有错误和不当之处，敬请读者批评指正。

联系地址：guoyd2@163.com。

郭玉东

2018.5.13

目　录

第1章 Linux组成结构

Linux 是当今最成功的操作系统之一，被广泛应用在手持设备、桌面 PC、服务器乃至巨型计算机中。与其他类型的操作系统一样，Linux 也由内核和应用程序组成[1]。内核运行在内核空间，负责管理计算机系统中的软、硬件资源，并通过资源的分配与回收协调应用程序的运行。应用程序运行在用户空间，完成各种各样的计算和应用处理工作，为用户提供各种类型的服务。内核与应用程序之间有着明确的职能分工和权限划分。内核拥有所有的特权，可以直接操作计算机系统中的所有硬件，使用处理器提供的所有指令和寄存器。应用程序拥有最少的特权，许多资源无权直接访问，许多指令无法执行，在运行过程中必须不断请求内核服务。内核运行在封闭的空间中，仅向应用程序开放一个受到严格管控的小窗口，称为系统调用。应用程序通过系统调用请求内核服务。为便于应用程序使用，Linux 又将内核提供的系统调用包装在函数库中，形成相对稳定的应用程序编程接口（Application Programming Interface，API）。

应用程序的运行离不开内核的支持，内核存在的目的是为应用程序提供服务，两者相辅相成。相比较而言，内核处于更加核心的位置，其组织结构也更加复杂。

1.1 Linux 内核结构

Linux 操作系统大致由三层组成：最底层的是相对封闭的操作系统内核，最上层的是各种类型的应用程序（运行中的应用程序称为用户进程），介于内核和用户进程之间的是应用程序编程接口，如图 1.1 所示[2]。

Linux 内核是相对封闭的，进出内核的途径是中断。用户进程请求内核服务的中断有两种：一种是陷入，即系统调用；另一种是内部中断，又称为异常。外部设备请求内核服务的中断称为外部中断。整个 Linux 内核被中断处理程序包围在封闭的内核空间中。

内核管理的硬件资源可分为处理器、内存、外存、网络设备和其他外部设备等几大类。Linux 内核用一个子系统管理一类硬件资源，因而其内核由进程管理、内

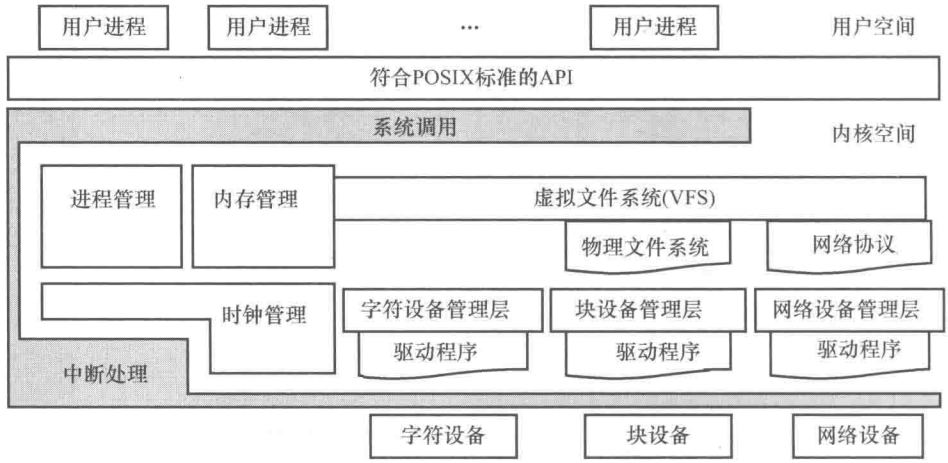


图 1.1　Linux 操作系统组成

存管理、文件系统、网络协议、设备管理等子系统组成。各子系统之间相对独立，又相互关联，并以特定的方式组织起来。内核各子系统的组织结构称为内核结构。

在 Linux 内核结构中可以看到层次结构和微内核的影子，借助于内核模块 KVM 和模拟软件 QEMU 的帮助，Linux 可将自己的内核转化成一个虚拟机监控器（Virtual Machine Monitor，VMM），从而支持虚拟机结构。然而总的来说，Linux 内核仍然是一种单块式的结构，一个子系统中的程序可以直接调用另一个子系统中函数。

1.1.1　中断处理

Linux 的中断处理程序都运行在内核中（特权级为 0），使用当前进程的系统堆栈。当中断发生时，处理器自动将中断返回位置、用户栈顶位置、错误代码等压入系统堆栈。

在 Intel 处理器上，Linux 处理三类中断的流程基本相同，如图 1.2 所示。

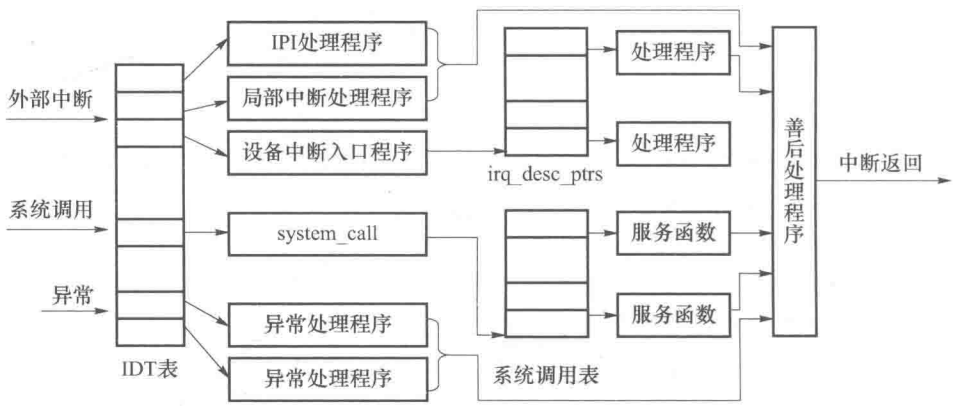


图 1.2　Intel 处理器上的中断处理流程

异常来源于处理器内部,表示处理器在执行指令的过程中检测到了某种错误条件,如被0除、页故障等。处理器预定了各类异常的产生条件、意义、中断向量号及处理方式,Linux内核为每类异常预定义了处理程序。当异常发生时,处理器根据异常的中断向量号跳转到与之对应的异常处理程序,即可完成异常处理。

硬中断来源于系统中的硬件,表示某硬件设备产生了需要引起处理器注意的事件。根据来源的不同,又可将硬中断分为设备中断(来源于外部设备)、局部中断(来源于处理器自身的Local APIC)和处理器间中断(Interprocessor Interrupt,IPI)(来源于其他处理器)。局部和IPI中断的意义与向量号是明确的,Linux内核已为每类局部和IPI中断预定义了处理程序。设备中断来源于外设,中断向量号由设备指定,中断意义由设备驱动程序解释,也应由设备驱动程序处理。为了更安全地处理设备中断,Linux定义了设备中断描述表irq_desc_ptrs和一组接口函数,允许设备驱动程序动态地注册、注销自己的设备中断处理程序。

陷入用于实现系统调用,是进入内核空间的门户,通常由INT n指令产生。Linux的系统调用是预先定义好的,每个系统调用都有一个编号,称为系统调用号,在内核中有一个对应的服务函数。系统调用号与服务函数的对应关系称为系统调用表。Linux内核按统一的方式处理系统调用,即根据系统调用号查系统调用表,获得与之对应的服务函数地址,而后执行服务函数完成系统调用处理。

三类中断的善后处理方式是一致的,主要处理工作包括三项:

(1) 检查当前进程的重调度标志,如当前进程需要重调度,则执行进程调度程序。

(2) 检查当前进程的信号标志,如当前进程有待处理的信号,则执行信号处理程序。

(3) 弹出系统堆栈中的内容,恢复处理器现场,中断返回。

在所有的硬中断中,时钟中断是最特殊的一类。时钟中断是操作系统中与时间相关的所有操作的基础。时钟中断来源于计算机系统中的时钟设备,如可编程间隔定时器(Programmable Interval Timer,PIT)、高精度定时器(High Precision Event Timer,HPET)、Local APIC Timer等。时钟设备可工作在周期中断模式和单发中断模式。计时器是能够提供当前时间值的硬件设备或软件程序,如实时时钟(Real Time Clock,RTC)、时间戳计数器(Time Stamp Counter,TSC)等。以时钟设备和计时器为基础,Linux可提供多种类型的服务,包括时间服务、定时服务、进程账务管理、负载管理等。

初始情况下,时钟设备被设定在周期中断模式,中断频率可以设定,如每秒1000次。Linux在周期性时钟中断处理中完成与时间相关的大部分管理工作。

(1) 时间管理。Linux的初始化程序从RTC中读出当前时间,将其转化成距离时刻1970-01-01 00:00:00的秒数,记录在变量xtime中,称为墙上时间。在此后的每次时钟中断处理中,Linux都会根据当前计时器更新变量xtime。以xtime和计

时器为基础,Linux 提供了多种时间服务,如函数 gettimeofday( )用于获取当前时间、函数 settimeofday( )用于设置当前时间等。

(2) 定时管理。Linux 提供了多类定时器,如给内核使用的核心定时器和高精度定时器,给应用程序使用的时间间隔定时器等。其中时间间隔定时器由用户进程启动,运行在内核空间中,到期时会向进程发送信号。Linux 提供三类时间间隔定时器,分别称为实时(Real)定时器、虚拟(Virtual)定时器和概略(Profile)定时器。实时定时器定的是实际流逝的时间量,到期时发送的信号是 SIGALRM。虚拟定时器定的是进程在用户态消耗的时间量,到期时发送的信号是 SIGVTALRM。概略定时器定的是进程消耗的时间量(包括用户态时间和系统态时间),到期时发送的信号是 SIGPROF。函数 alarm( )启动的是实时定时器,函数 setitimer( )可启动任意一种时间间隔定时器,函数 sleep( )可让进程睡眠若干秒。

(3) 进程账务管理。在账务管理中,Linux 累计当前进程消耗的时间、检查时间间隔定时器、更新进程的时间片等,并在必要时设置当前进程的重调度标志从而触发进程调度。

(4) 负载管理。在负载管理中,Linux 统计系统的负载,并在必要时触发负载平衡程序,以期维持各 CPU 之间的负载均衡。

周期性时钟中断简单但盲目,因而只要有可能,Linux 都试图抛弃传统的周期性时钟中断,改用更加智能、准确的单发式(One Shot)时钟中断。目前的时钟设备大都支持单发中断模式,可在准确的时间点上产生时钟中断。单发式时钟中断通常是不连续的,时间间隔不等甚至可以暂停,产生时钟中断的每一个时间点都必须明确地、显式地设定。

1.1.2 进程管理

操作系统的核心工作是运行程序。在 Linux 操作系统中,所有的程序(包括 Linux 内核)都只能在进程中运行,在进程之外不能运行任何程序。Linux 的进程(Process)是运行中的程序,是用户在操作系统中的代理,是内核的主要服务对象。

Linux 用结构 task_struct 描述进程和线程,其内容大致包括如下几类:

(1) 进程标识。Linux 用进程标识符(Process Identifier,PID)标识进程,系统中的每个进程都有一个 PID。初始进程的 PID 是 0,其他进程的 PID 是在创建时临时分配的。Linux 在目录/proc 中为每个进程建立一个子目录,其中包含进程的所有管理信息,目录名就是进程的 PID 号。

一到多个轻量级进程(又称线程)构成一个线程组,它们共享领头进程的资源,如虚拟地址空间、文件描述符表等。一个线程组中的所有轻量级进程拥有相同的线程组标识符(Thread Group ID,TGID),领头进程的 PID 就是线程组的 TGID。Linux 系统中的每个进程都属于一个进程组(Process group),每个进程组又属于一个会话(Session)。