



下载APP



39 | 瞧一瞧Linux：详解socket实现与网络编程接口

2021-08-06 LMOS

《操作系统实战45讲》

课程介绍 >



讲述：陈晨

时长 11:46 大小 10.79M



你好，我是 LMOS。

前面我们了解了网络的宏观架构，建立了网络模块知识的大局观，也进行了实际的组网实践。现在我们来瞧一瞧 Linux 的网络程序，不过想要入门 Linux 的网络编程，套接字也是一个绕不开的重要知识点，正是有了套接字，Linux 系统才拥有了网络通信的能力。而且网络协议的最底层也是套接字，有了这个基础，你再去查看相关的网络协议的时候也会更加轻松。

我会通过两节课的内容带你了解套接字的原理和具体实现。这节课，我们先来了解套接字的作用、工作原理和关键数据结构。下一节课，我们再一起研究它在 Linux 内核中的设计与实现。



好，让我们开始今天的学习吧。

如何理解套接字

根据底层网络机制的差异，计算机网络世界中定义了不同协议族的套接字（socket），比如 DARPA Internet 地址（Internet 套接字）、本地节点的路径名（Unix 套接字）、CCITT X.25 地址（X.25 套接字）等。

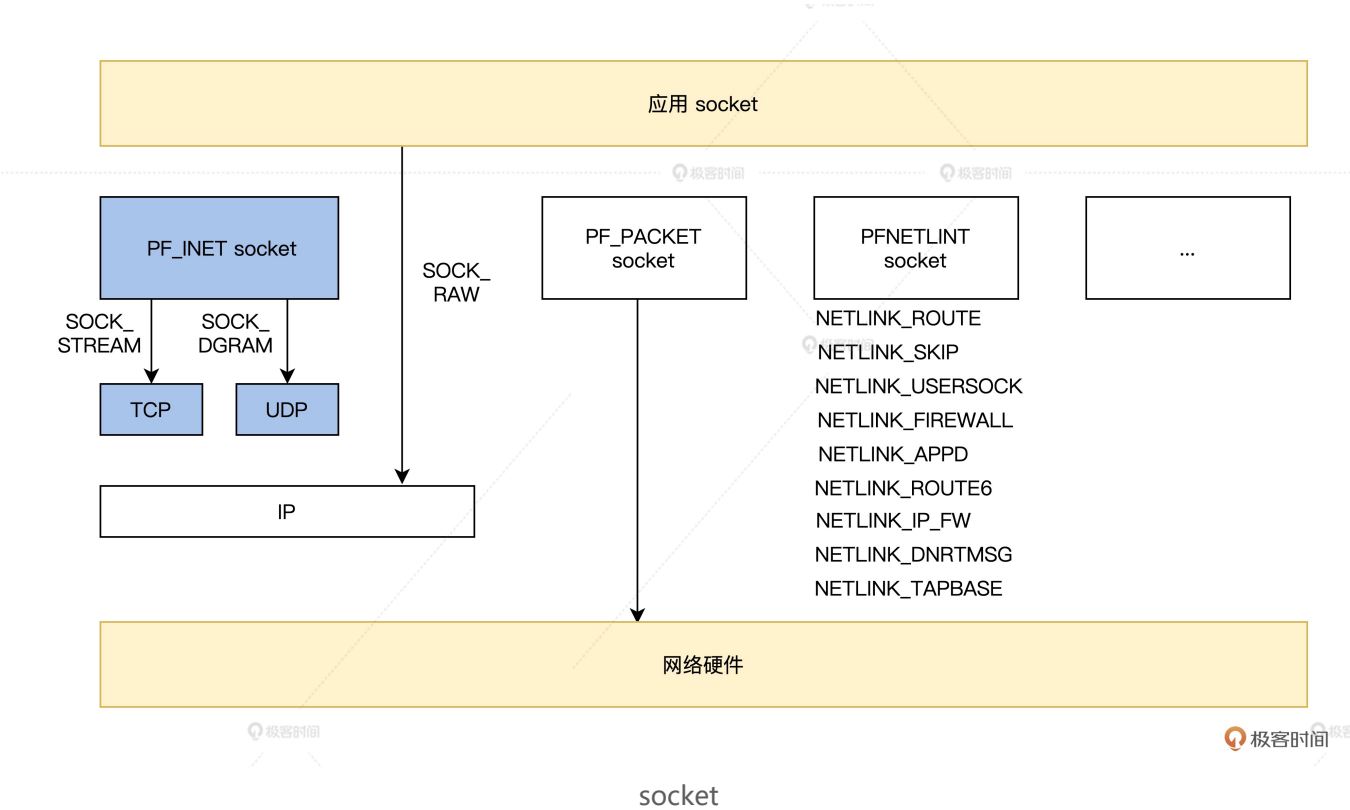
我们会重点讲解跟网络子系统和 TCP/IP 协议栈息息相关的一种套接字——Internet 套接字。如果你对其他类型的套接字有兴趣，可以自行阅读这里的 [🔗 资料](#)。

Internet 套接字是 TCP/IP 协议栈中传输层协议的接口，也是传输层以上所有协议的实现。

同时，套接字接口在网络程序功能中是内核与应用层之间的接口。TCP/IP 协议栈的所有数据和控制功能都来自于套接字接口，与 OSI 网络分层模型相比，TCP/IP 协议栈本身在传输层以上就不包含任何其他协议。

在 Linux 操作系统中，替代传输层以上协议实体的标准接口，称为套接字，它负责实现传输层以上所有的功能，可以说套接字是 TCP/IP 协议栈对外的窗口。

Linux 套接字 API 适合所有的应用标准，现在的应用层协议也全部移植到了 Linux 系统中。但请你注意，在套接字层下的基础体系结构实现却是 Linux 系统独有的，Linux 内核支持的套接字结构如图所示。



我们创建套接字时，可以通过参数选择协议族，为应用程序指定不同的网络机制。如果指定为 PF_INET 协议族，这里的套接字就叫做 INET 套接字，它的套接字接口函数提供了 TCP/IP 网络服务功能。现在我先带你了解一下套接字的数据结构。

套接字的数据结构

在 Linux 操作系统下，对套接字、套接字的属性、套接字传输的数据格式还有管理套接字连接状态的数据结构分别做了一系列抽象定义。

每个程序使用的套接字都有一个 **struct socket** 数据结构与 **struct sock** 数据结构的实例。

Linux 内核在套接字层定义了包含套接字通用属性的数据结构，分别是 struct socket 与 struct sock，它们独立于具体协议；而具体的协议族与协议实例继承了通用套接字的属性，加入协议相关属性，就形成了管理协议本身套接字的结构。

struct socket 数据结构

struct socket 是套接字结构类型，每个套接字在内核中都对应唯一的 struct socket 结构（用户程序通过唯一的套接字描述符来表示套接字，且描述符与 struct socket 结构一一对应）。

我们来看看 struct socket 数据结构是什么样，代码如下，我相信配合注释你有能力理解它。

[复制代码](#)

```

1 struct socket {
2     socket_state          state; // 套接字的状态
3     unsigned long         flags; // 套接字的设置标志。存放套接字等待缓冲区的状态信息
4     struct fasync_struct   *fasync_list; // 等待被唤醒的套接字列表，该链表用于异步
5     struct file            *file; // 套接字所属的文件描述符
6     struct sock            *sk; // 指向存放套接字属性的结构指针
7     wait_queue_head_t      wait; // 套接字的等待队列
8     short                 type; // 套接字的类型。其取值为SOCK_XXXX形式
9     const struct proto_ops *ops; // 套接字层的操作函数块
10 }

```

struct sock 数据结构

在 Linux 内核的早期版本中，struct sock 数据结构非常复杂。从 Linux2.6 版本以后，从两个方面对该数据结构做了优化。

其一是将 struct sock 数据结构划分成了两个部分。一部分为描述**套接字的共有属性**，所有协议族的这些属性都一样；另一部分属性定义在了 **struct sock_common** 数据结构中。

其二是为新套接字创建 struct sock 数据结构实例时，会从协议特有的缓冲槽中分配内存，不再从通用缓冲槽中分配内存。

struct sock 数据结构包含了大量的内核管理套接字的信息，内核把最重要的成员存放在 struct sock_common 数据结构中，struct sock_common 数据结构嵌入在 struct sock 结构中，它是 struct sock 数据结构的第一个成员。

struct sock_common 数据结构是套接字在网络中的最小描述，它包含了内核管理套接字最重要信息的集合。而 struct sock 数据结构中包含了套接字的全部信息与特点，有的特性很少用到，甚至根本就没有用到。我们这里就看一下 struct sock_common 的数据结构，代码如下。

[复制代码](#)

```

1 struct sock_common {
2     unsigned short         skc_family; // *地址族*

```

```
3     volatile unsigned char   skc_state;           /*连接状态*/
4     unsigned char            skc_reuse;           /*SO_REUSEADDR设置*/
5     int                       skc_bound_dev_if;
6     struct hlist_node         skc_node;
7     struct hlist_node         skc_bind_node;      /*哈希表相关*/
8     atomic_t                  skc_refcnt;         /*引用计数*/
9     };
```

结合代码可以看到，系统中 struct sock 数据结构组织在特定协议的哈希链表中，skc_node 是连接哈希链表中成员的哈希节点，skc_hash 是引用的哈希值。接收和发送数据放在数据 struct sock 数据结构的两个等待队列中：sk_receive_queue 和 sk_write_queue。这两个队列中包含的都是 Socket Buffer（后面我会展开讲）。

内核使用 struct sock 数据结构实例中的回调函数，获取套接字上某些事件发生的消息或套接字状态发生变化。其中，使用最频繁的回调函数是 **sk_data_ready**，用户进程等待数据到达时，就会调用该回调函数。

套接字与文件

套接字的连接建立起来后，用户进程就可以使用常规文件操作访问套接字了。

这种方式在内核中如何实现，这要取决于 Linux 虚拟文件系统层（VFS）的实现。在 VFS 中，每个文件都有一个 VFS inode 结构，每个套接字都分配了一个该类型的 inode，套接字中的 inode 指针连接管理常规文件的其他结构。操作文件的函数存放在一个独立的指针表中，代码如下。

[复制代码](#)

```
1 struct inode{
2     struct file_operation *i_fop // 指向默认文件操作函数块
3 }
```

套接字的文件描述符的文件访问的重定向，对网络协议栈各层是透明的。而 inode 和 socket 的链接是通过直接分配一个辅助数据结构来实现的，这个数据结构的代码如下。

[复制代码](#)

```
1 struct socket_slloc {
2     struct socket socket;
3     struct inode vfs_inode;
```



```
4 }
```

套接字缓存

前面我们提到了一个 Socket Buffer，也就是套接字缓存，它代表了一个要发送或者处理的报文。在 Linux 网络子系统中，Socket Buffer 是一个关键的数据结构，因为它贯穿于整个 TCP/IP 协议栈的各层。Linux 内核对网络数据打包处理的全过程中，始终伴随着这个 Socket Buffer。

你可以这样理解，**Socket Buffer 就是网络数据包在内核中的对象实例。**

Socket Buffer 主要由两部分组成。

1. 数据包：存放了在网络中实际流通的数据。
2. 管理数据结构（struct sk_buff）：当在内核中对数据包进行时，内核还需要一些其他的数据来管理数据包和操作数据包，例如协议之间的交换信息，数据的状态，时间等。

Socket Buffer 有什么作用呢？struct sk_buff 数据结构中存放了套接字接收 / 发送的数据。在发送数据时，在套接字层创建了 Socket Buffer 缓冲区与管理数据结构，存放来自应用程序的数据。在接收数据包时，Socket Buffer 则在网络设备的驱动程序中创建，存放来自网络的数据。

在发送和接受数据的过程中，各层协议的头信息会不断从数据包中插入和去掉，sk_buff 结构中描述协议头信息的地址指针也会被不断地赋值和复位。

套接字的初始化

Linux 的网络体系结构可以支持多个协议栈和网络地址类型。内核支持的每一个协议栈都会在套接字层注册一个地址族。这就解释了为什么在套接字层可以有一个通用的 API，供完全不同的协议栈使用。

Linux 内核支持的地址族非常多，TCP/IP 协议栈在套接字层注册的地址族是 AF_INET，AF_INET 地址族是在内核启动时注册到内核中的。TCP/IP 协议栈与 AF_INET 地址族相连

的处理函数，既可以在套接字初始化时与 AF_INET 地址连接起来，也可以在套接字中动态地注册新的协议栈。

套接字层的初始化要为以后各协议初始化 struct sock 数据结构对象、套接字缓冲区 Socket Buffer 对象等做好准备，预留内存空间。

套接字层初始化要完成的基本任务包括后面这三项。

1. 初始化套接字的缓存槽
2. 为 Socket Buffer 创建内存缓存槽
3. 创建虚拟文件系统

初始化函数代码如下所示。

[复制代码](#)

```
1 static int __init sock_init(void) {
2     int err;
3     /*
4      *      初始化.sock缓存
5      */
6     sk_init();
7     /*
8      *      初始化sk_buff缓存
9      *      skb_init();
10     /*      初始化协议模块缓存
11
12     init_inodecache();
13     /* 注册文件系统类型 */
14     err = register_filesystem(&sock_fs_type);
15     if (err) goto out_fs;
16     sock_mnt = kern_mount(&sock_fs_type);
17     if (IS_ERR(sock_mnt)) {
18         err = PTR_ERR(sock_mnt);
19         goto out_mount;
20     }
21 }
```

地址族的值和协议交换表

套接字是一个通用接口，它可以与多个协议族建立接口，每个协议族中又可以实现多个协议实例。


TCP/IP 协议栈处理完输入数据包后，将数据包交给套接字层，放在套接字的接收缓冲区队列（`sk_rcv_queue`）。然后数据包从套接字层离开内核，送给应用层等待数据包的用户程序。用户程序向外发送的数据包缓存在套接字的传送缓冲区队列（`sk_write_queue`），从套接字层进入内核地址空间。

在同一个主机中，可以同时多个协议上打开多个套接字，来接收和发送网络数据，套接字层必须确定哪个套接字是当前数据包的目标套接字。

怎么精准确定呢？

在 Linux 内核里有一个叫做 `struct inet_protosw` 的数据结构，它就负责完成这个功能，具体来看就是管理和描述 `struct proto_ops` 和 `struct proto` 之间的对应关系。这里 `struct proto_ops` 就是系统调用套接字的操作函数块，而 `struct proto` 就是跟内核协议相关的套接字操作函数块。

后面这段代码是 `inet_protosw`。

 复制代码

```
1 struct inet_protosw {
2     struct list_head list;
3     unsigned short   type;          /* AF_INET协议族套接字的类型,如TCP为SOCK_STREAM*/
4     unsigned short   protocol; /* 协议族中某个协议实例的编号。如TCP协议的编码为IPPROTO_
5
6     struct proto      *prot;
7     const struct proto_ops *ops;
8
9     unsigned char     flags;        /* 该套接字属性的相关标志 */
10
11 }
```

结合上面代码我们发现，内核使用 `struct inet_protosw` 数据结构实现的协议交换表，将应用程序通过 `socketcall` 系统调用指定的套接字操作，转换成对某个协议实例实现的套接字操作函数的调用。

struct inet_protosw 类型把 INET 套接字的协议族操作集与传输层协议操作集关联起来。该类型的 **inetsw_array** 数组变量实现了 INET 套接字的协议族操作集与具体的传输层协议关联。由 struct inet_protosw 数据结构类型数组 inetsw_array[] 构成的向量表，称为**协议交换表**，协议交换表满足了套接字支持多协议栈这项功能。

重点回顾

好，这节课的内容告一段落了，我来给你做个总结。这节课我们一起理解了 Linux 内核套接字的概念。

套接字是 UNIX 兼容系统的一大特色，是 UNIX 一切皆是文件操作概念的具体实现，从实现的角度来看，**套接字是通信的抽象描述**；从内核角度看，同时也是一个管理通信过程的对象——**struct socket 结构**。

Linux 的网络体系结构可以支持多个协议栈和网络地址类型，通过地址族的值和协议交换表，Linux 的套接字实现了支持多协议栈这项功能。

我特意为你梳理了这节课最关键的两个要点，需要你重点理解。

1. 从描述 Linux 套接字接口的数据结构、套接字接口初始化过程可知，Linux 套接字体系结构独立于具体网络协议栈的套接字，可以同时支持多个网络协议栈的工作。
2. 套接字内核实现，我们具体分析了套接字从创建的过程。根据分析我们可以发现，任何协议栈都可以在套接字通用体系结构的基础上，派生出具有**协议族特点**的套接字接口。

思考题

套接字也是一种进程间通信机制，它和其他通信机制有什么不同？

欢迎你在留言区记录你的疑惑或者心得，也推荐你把这节课分享给身边的同事、朋友，跟他一起学习进步。

我是 LMOS。我们下节课见！

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 38 | 从单排到团战：详解操作系统的宏观网络架构

下一篇 40 | 瞧一瞧Linux：详解socket的接口实现

小争哥新书

数据结构与算法之美

图书+专栏，双管齐下，拿下算法

打包价 **¥159** 原价¥319

仅限 300 套



精选留言 (6)

💬 写留言



不及胜于过之 置顶

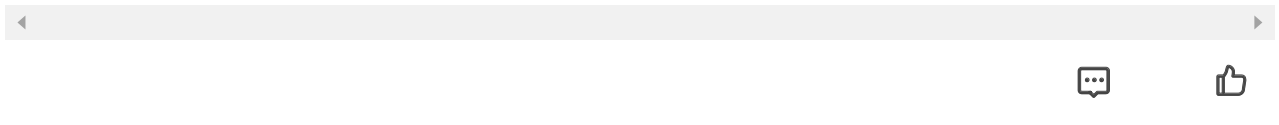
2021-08-09

昨天一天撸完，体会很深，专门写了一个学习总结与linux的爬坡之路，<https://mp.weixin.qq.com/s/XqXlvEfhNPXQ1RSs0XeFUQ>

麻烦多指正，过去一直持续在学linux，这个时候看到您的文章对我是一个很好的沉淀与认知突破，巨感谢大佬

展开

作者回复: 6666



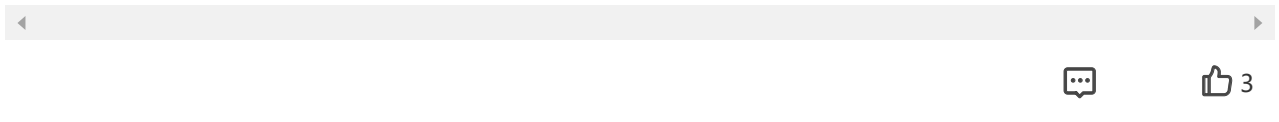
pedro

2021-08-06

进程间的通信方式有很多，比如说管道，共享内存，信号等，但这些通信方式都有一个很大的局限性，那就是无法跨物理机通信，只能与同一个机器上的其它进程通信，而套接字恰好打破了这个桎梏，只要你在网上(网络上)，我就可以通过ip地址打你电话，和你说话！

展开 ∨

作者回复: 是的 铁汁



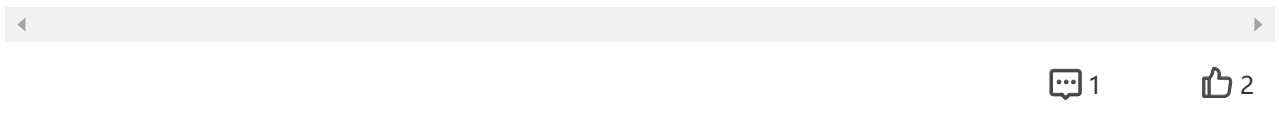
不及胜于过之

2021-08-08

今天一天看完了您的所有课程，收获非常非常大。尤其是：要实现一个功能模块，首先要设计出相应的数据结构(以及这些数据结构的管理数据结构，比如链表等)，基于数据结构设计初始化函数以及该功能模块对应的业务函数。为学习操作系统模块或所有技术项目代码提供了思路，感谢东哥。

展开 ∨

作者回复: 哈哈 对的 基于数据结构才能解决问题



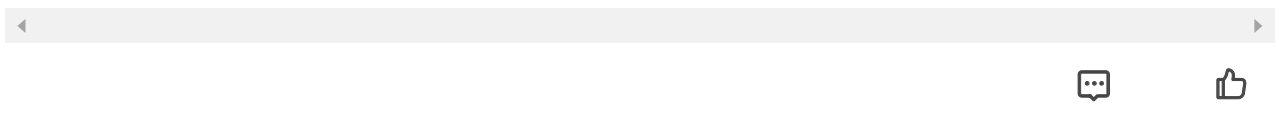
flydream

2021-08-07

什么时候更新啊，吊胃口啊

展开 ∨

编辑回复: 下周一，每周一三更新~



feihui

2021-08-06

老师，有个疑问，文中说到：“结合上面代码我们发现，内核使用 struct inet_protosw 数据结构实现的协议交换表，将应用程序通过 socketcall 系统调用指定的套接字操作，转换成对某个协议实例实现的套接字操作函数的调用。”为什么不直接调用 socket 中的 ops (ops 直接指向具体协议的操作) 呢

展开 ▾



MacBao

2021-08-06

套接字可以跨主机，其他的不可以

展开 ▾

作者回复: 哈哈 铁汁牛逼

