Netlink Socket

## 1、Netlink通信机制

Netlink套接字是用以实现用户进程与内核进程通信的一种特殊的进程间通信(IPC),也是网络应用程序与内核通信的最常用的接口。

* 用户态应用使用标准的socket API就可以使用netlink提供的强大功能，
* 内核态需要使用专门的内核API来使用netlink。

Netlink相对于系统调用，ioctl以及/proc文件系统而言具有以下优点：

* netlink使用简单，只需要在include/linux/netlink.h中增加一个新类型的 netlink 协议定义即可,(如#define NETLINK\_TEST 20 然后，内核和用户态应用就可以立即通过 socket API使用该netlink协议类型进行数据交换);
* netlink是一种异步通信机制，在内核与用户态应用之间传递的消息保存在socket缓存队列中，发送消息只是把消息保存在接收者的socket的接收队列，而不需要等待接收者收到消息；
* 使用netlink的内核部分可以采用模块的方式实现，使用netlink的应用部分和内核部分没有编译时依赖;
* netlink支持多播，内核模块或应用可以把消息多播给一个netlink组，属于该neilink 组的任何内核模块或应用都能接收到该消息，内核事件向用户态的通知机制就使用了这一特性；
* 内核可以使用 netlink 首先发起会话;

## 2、用户态编程

**（1）用户态编程接口**

用户态应用使用标准的socket API有sendto()，recvfrom()； sendmsg(), recvmsg()

**（2）sockaddr\_nl结构**

struct sockaddr\_nl是netlink通信地址，跟普通socket struct sockaddr\_in类似。

struct sockaddr\_nl {

\_\_kernel\_sa\_family\_t nl\_family; */\* AF\_NETLINK （跟AF\_INET对应）\*/*

unsigned short nl\_pad; */\* zero \*/*

\_\_u32 nl\_pid; */\* port ID （通信端口号）\*/*

\_\_u32 nl\_groups; */\* multicast groups mask \*/*

};

**（3）nlmsghd结构**

nlmsghd是netlink的消息头。

struct nlmsghdr {

\_\_u32 nlmsg\_len; */\* 整个netlink消息的长度(包括消息头) \*/*

\_\_u16 nlmsg\_type; */\* 消息类型 \*/*

\_\_u16 nlmsg\_flags; */\* 消息标记，用于表示消息的类型 \*/*

\_\_u32 nlmsg\_seq; */\* 消息序列号，用于消息排队，可选的，并不强制使用 \*/*

\_\_u32 nlmsg\_pid; */\* 发送端口的ID号，对于内核来说该值就是0，对于用户进程来说就是其socket所绑定的ID号 \*/*

};

上面的nlmsg\_type的类型有：

#define NLMSG\_NOOP 0x1 */\* 不执行任何动作，必须将该消息丢弃. \*/*

#define NLMSG\_ERROR 0x2 */\* 消息发生错误 \*/*

#define NLMSG\_DONE 0x3 */\* 标识分组消息的末尾 \*/*

#define NLMSG\_OVERRUN 0x4 */\* 缓冲区溢出，表示某些消息已经丢失 \*/*

#define NLMSG\_MIN\_TYPE 0x10 */\* 预留 \*/*

上面nlmsg\_flags的值可以有：

*/\* Flags values \*/*

#define NLM\_F\_REQUEST 1 */\* It is request message. \*/*

#define NLM\_F\_MULTI 2 */\* Multipart message, terminated by NLMSG\_DONE \*/*

#define NLM\_F\_ACK 4 */\* Reply with ack, with zero or error code \*/*

#define NLM\_F\_ECHO 8 */\* Echo this request \*/*

#define NLM\_F\_DUMP\_INTR 16 */\* Dump was inconsistent due to sequence change \*/*

*/\* Modifiers to GET request \*/*

#define NLM\_F\_ROOT 0x100 */\* specify tree root \*/*

#define NLM\_F\_MATCH 0x200 */\* return all matching \*/*

#define NLM\_F\_ATOMIC 0x400 */\* atomic GET \*/*

#define NLM\_F\_DUMP (NLM\_F\_ROOT | NLM\_F\_MATCH)

*/\* Modifiers to NEW request \*/*

#define NLM\_F\_REPLACE 0x100 */\* Override existing \*/*

#define NLM\_F\_EXCL 0x200 */\* Do not touch, if it exists \*/*

#define NLM\_F\_CREATE 0x400 */\* Create, if it does not exist \*/*

#define NLM\_F\_APPEND 0x800 */\* Add to end of list \*/*

（4）msghdr结构

struct iovec { */\* Scatter/gather array items \*/*

void \*iov\_base; */\* Starting address \*/*

size\_t iov\_len; */\* Number of bytes to transfer \*/*

};

struct msghdr {

void \*msg\_name; */\* 数据的目的地址，网络包指向sockaddr\_in, netlink则指向sockaddr\_nl \*/*

socklen\_t msg\_namelen; */\* msg\_name所代表的地址长度 \*/*

struct iovec \*msg\_iov; */\* 指向的是缓冲区数组 \*/*

size\_t msg\_iovlen; */\* 缓冲区数组长度 \*/*

void \*msg\_control; */\* 辅助数据，控制信息(发送任何的控制信息) \*/*

size\_t msg\_controllen; */\* 辅助信息长度 \*/*

int msg\_flags; */\* 消息标识 \*/*

};

iov\_base指向数据包缓冲区，即参数buff，iov\_len是buff的长度。msghdr中允许一次传递多个buff，以数组的形式组织在msg\_iov中，msg\_iovlen就记录数组的长度（即有多少个buff）。

## 3、内核态编程

**（1）netlink消息类型**

#define NETLINK\_ROUTE 0 */\* Routing/device hook \*/*

#define NETLINK\_UNUSED 1 */\* Unused number \*/*

#define NETLINK\_USERSOCK 2 */\* Reserved for user mode socket protocols \*/*

#define NETLINK\_FIREWALL 3 */\* Unused number, formerly ip\_queue \*/*

#define NETLINK\_SOCK\_DIAG 4 */\* socket monitoring \*/*

#define NETLINK\_NFLOG 5 */\* netfilter/iptables ULOG \*/*

#define NETLINK\_XFRM 6 */\* ipsec \*/*

#define NETLINK\_SELINUX 7 */\* SELinux event notifications \*/*

#define NETLINK\_ISCSI 8 */\* Open-iSCSI \*/*

#define NETLINK\_AUDIT 9 */\* auditing \*/*

#define NETLINK\_FIB\_LOOKUP 10

#define NETLINK\_CONNECTOR 11

#define NETLINK\_NETFILTER 12 */\* netfilter subsystem \*/*

#define NETLINK\_IP6\_FW 13

#define NETLINK\_DNRTMSG 14 */\* DECnet routing messages \*/*

#define NETLINK\_KOBJECT\_UEVENT 15 */\* Kernel messages to userspace \*/*

#define NETLINK\_GENERIC 16

*/\* leave room for NETLINK\_DM (DM Events) \*/*

#define NETLINK\_SCSITRANSPORT 18 */\* SCSI Transports \*/*

#define NETLINK\_ECRYPTFS 19

#define NETLINK\_RDMA 20

#define NETLINK\_CRYPTO 21 */\* Crypto layer \*/*

#define NETLINK\_INET\_DIAG NETLINK\_SOCK\_DIAG

#define MAX\_LINKS 32

**（2）netlink常用宏**

#define NLMSG\_ALIGNTO 4U

*/\* 宏NLMSG\_ALIGN(len)用于得到不小于len且字节对齐的最小数值 \*/*

#define NLMSG\_ALIGN(*len*) (((len) + NLMSG\_ALIGNTO - 1) & ~(NLMSG\_ALIGNTO - 1))

*/\* Netlink 头部长度 \*/*

#define NLMSG\_HDRLEN ((int)NLMSG\_ALIGN(sizeof(struct nlmsghdr)))

*/\* 计算消息数据len的真实消息长度（消息体 +　消息头）\*/*

#define NLMSG\_LENGTH(*len*) ((len) + NLMSG\_HDRLEN)

*/\* 宏NLMSG\_SPACE(len)返回不小于NLMSG\_LENGTH(len)且字节对齐的最小数值 \*/*

#define NLMSG\_SPACE(*len*) NLMSG\_ALIGN(NLMSG\_LENGTH(len))

*/\* 宏NLMSG\_DATA(nlh)用于取得消息的数据部分的首地址，设置和读取消息数据部分时需要使用该宏 \*/*

#define NLMSG\_DATA(*nlh*) ((void \*)(((char \*)nlh) + NLMSG\_LENGTH(0)))

*/\* 宏NLMSG\_NEXT(nlh,len)用于得到下一个消息的首地址, 同时len 变为剩余消息的长度 \*/*

#define NLMSG\_NEXT(*nlh, len*) ((len) -= NLMSG\_ALIGN((nlh)->nlmsg\_len), \

(struct nlmsghdr \*)(((char \*)(nlh)) + NLMSG\_ALIGN((nlh)->nlmsg\_len)))

*/\* 判断消息是否 >len \*/*

#define NLMSG\_OK(*nlh, len*) ((len) >= (int)sizeof(struct nlmsghdr) && \

(nlh)->nlmsg\_len >= sizeof(struct nlmsghdr) && \

(nlh)->nlmsg\_len <= (len))

*/\* NLMSG\_PAYLOAD(nlh,len) 用于返回payload的长度\*/*

#define NLMSG\_PAYLOAD(*nlh, len*) ((nlh)->nlmsg\_len - NLMSG\_SPACE((len)))

**（3）netlink内核常用函数**

netlink\_kernel\_create()用于创建内核socket用户态通信

|  |  |
| --- | --- |
| 函数原形 | static inline struct sock \*  netlink\_kernel\_create(struct net \*net, int unit, struct netlink\_kernel\_cfg \*cfg) |
| 函数参数 | net: net指向所在的网络命名空间，一般默认传入的是&init\_net(不需要定义);定义在net\_namespace.c(extern struct net init\_net); |
| unit：netlink协议类型 |
| cfg：cfg存放的是netlink内核配置参数 |

cfg可选的内核配置参数：

struct netlink\_kernel\_cfg {

unsigned int groups;

unsigned int flags;

void (\*input)(struct sk\_buff \*skb); */\* input 回调函数 \*/*

struct mutex \*cb\_mutex;

void (\*bind)(int group);

bool (\*compare)(struct net \*net, struct sock \*sk);

};

netlink\_unicast()用于发送单播消息：

|  |  |
| --- | --- |
| 函数原形 | int netlink\_unicast(struct sock \*ssk, struct sk\_buff \*skb, \_\_u32 portid, int nonblock); |
| 函数参数 | ssk: netlink socket |
| skb: skb buff指针 |
| portid：通信的端口号 |
| nonblock：表示该函数是否为非阻塞，如果为1，该函数将在没有接收缓存可利用时立即返回，而如果为0，该函数在没有接收缓存可利用定时睡眠 |

netlink\_broadcast()用于发送多播消息：

|  |  |
| --- | --- |
| 函数原形 | int netlink\_broadcast(struct sock \*ssk, struct sk\_buff \*skb, \_\_u32 portid, \_\_u32 group, gfp\_t allocation); |
| 函数参数 | ssk: 同上（对应netlink\_kernel\_create 返回值）、 |
| skb: 内核skb buff |
| portid：端口id |
| group: 是所有目标多播组对应掩码的"OR"操作的合值 |
| allocation: 指定内核内存分配方式，通常GFP\_ATOMIC用于中断上下文，而GFP\_KERNEL用于其他场合。  这个参数的存在是因为该API可能需要分配一个或多个缓冲区来对多播消息进行clone |