

## Notifier Chains

The notifier chain facility is a general mechanism provided by the kernel. It is designed to provide a way for kernel elements to express interest in being informed about the occurrence of general asynchronous events. The basic building block of the mechanism is the *struct notifier\_block* which is defined in *include/linux/notifier.h*. The block contains a pointer to the function to be called when the event occurs. The parameters passed to the notifier function include:

- a pointer to the notifier block itself,
- an event code such as `NETDEV_REGISTER` or `NETDEV_UNREGISTER`,
- and a pointer to an unspecified private data type which in the case of the network chain points to the associated *struct netdevice*.

```
14 struct notifier_block
15 {
16     int (*notifier_call)(struct notifier_block *self,
17                          unsigned long, void *);
17     struct notifier_block *next;
18     int priority;
19 };

181
182 static struct notifier_block *netdev_chain=NULL;
183
```

The kernel function *notifier\_chain\_register()* assembles related notifier blocks into notifier chains. Modules within the networking subsystem use the *register\_netdevice\_notifier()* function defined in *net/core/dev.c* to add their own notifier blocks to the *netdev\_chain* which is statically initialized as `NULL` in *dev.c*.

```
850 int register_netdevice_notifier(struct notifier_block *nb)
851 {
852     return notifier_chain_register(&netdev_chain, nb);
853 }
```

### Adding the *notifier\_block* to the chain.

The kernel routine *notifier\_chain\_register()* links the notifier block into the specified chain in priority order.

```
63
64 int notifier_chain_register(struct notifier_block **list,
        struct notifier_block *n)
65 {
66     write_lock(&notifier_lock);
67     while(*list)
68     {
69         if(n->priority > (*list)->priority)
70             break;
71         list= &((*list)->next);
72     }
73     n->next = *list;
74     *list=n;
75     write_unlock(&notifier_lock);
76     return 0;
77 }
```

Here are the notifiers associated with *net\_device* events.

```
41 /* netdevice notifier chain */
42 #define NETDEV_UP 0x0001
        /* For now you can't veto a device up/down */
43 #define NETDEV_DOWN 0x0002
44 #define NETDEV_REBOOT 0x0003
        /* Tell a protocol stack a network interface
45     detected a hardware crash and restarted
46     - we can use this eg to kick tcp sessions
47         once done */
48 #define NETDEV_CHANGE 0x0004
        /* Notify devstate change */
49 #define NETDEV_REGISTER 0x0005
50 #define NETDEV_UNREGISTER 0x0006
51 #define NETDEV_CHANGEMTU 0x0007
52 #define NETDEV_CHANGEADDR 0x0008
53 #define NETDEV_GOING_DOWN 0x0009
54 #define NETDEV_CHANGENAME 0x000A
55
```

## An example registration

Here is the notifier block register by the *netlink* component.

```
516 struct notifier_block rtnetlink_dev_notifier = {
517     rtnetlink_event,    // handler
518     NULL,               // parameter
519     0,                  // priority
520 };

521 void __init rtnetlink_init(void)
522 {
523     #ifdef RTNL_DEBUG
524         printk("Initiank_init(void)
525     {
526     #ifdef RTNL_DEBUG
527         printk("Initializing RT netlink socket\n");
528     #endif
529     rtnl = netlink_kernel_create(NETLINK_ROUTE,
530                                rtnetlink_rcv);
531     if (rtnl == NULL)
532         panic("rtnetlink_init: cannot initialize
533             rtnetlink\n");
534     netlink_set_nonroot(NETLINK_ROUTE, NL_NONROOT_RECV);
535     register_netdevice_notifier(&rtnetlink_dev_notifier);
536     rtnetlink_links[PF_PACKET] = link_rtnetlink_table;
537 }
538 }
```

### Invoking *notifier\_call\_chain()*

When a function such as *netdev\_init()* makes the call to *notifier\_call\_chain()*, it results in a callback being made for every notifier block that is in the chain. These notifier callback functions typically contain a *switch()* block which they use to select and process only those event types in which they are interested.

```
2557 /* Notify protocols, that a new device appeared. */
2558     notifier_call_chain(&netdev_chain, NETDEV_REGISTER, dev);
```

The *handlers* are invoked in priority order and as shown below a handler can abort the process by returning a value with the *NOTIFY\_STOP\_MASK* set.

```
122 int notifier_call_chain(struct notifier_block **n,
123                          unsigned long val, void *v)
124 {
125     int ret=NOTIFY_DONE;
126     struct notifier_block *nb = *n;
127     while(nb)
128     {
129         ret = nb->notifier_call(nb,val,v);
130         if(ret & NOTIFY_STOP_MASK)
131         {
132             return ret;
133         }
134         nb=nb->next;
135     }
136     return ret;
137 }
```

## The *netlink* handler

This structure is illustrated below in the *rtnetlink\_event()* callback. The impact of the events shown may be *further* propagated through the network system to recipients of the netlinks message.

```
487 static int rtnetlink_event(struct notifier_block *this,
                             unsigned long event, void *ptr)
488 {
489     struct net_device *dev = ptr;
490     switch (event) {
491     case NETDEV_UNREGISTER:
492         rtmsg_ifinfo(RTM_DELLINK, dev, ~0U);
493         break;
494     case NETDEV_REGISTER:
495         rtmsg_ifinfo(RTM_NEWLINK, dev, ~0U);
496         break;
497     case NETDEV_UP:
498     case NETDEV_DOWN:
499         rtmsg_ifinfo(RTM_NEWLINK, dev,
                       IFF_UP|IFF_RUNNING);
500         break;
501     case NETDEV_CHANGE:
502     case NETDEV_GOING_DOWN:
503         break;
504     default:
505         rtmsg_ifinfo(RTM_NEWLINK, dev, 0);
506         break;
507     }
508     return NOTIFY_DONE;
509 }
510
```

The entire collection of callers of *register\_netdevice\_notifier()* is quite large. Each of the modules shown below has a callback function in the *netdev* chain. However, only the notifiers shown in red have any impact on IP\_V4.

Referenced (in 35 files total) in:

- include/linux/netdevice.h, line 454
- net/netsyms.c, line 465
- net/appletalk/aarp.c, line 859
- net/appletalk/ddp.c, line 1974
- net/ax25/af\_ax25.c, line 1851
- net/core/dev.c, line 850
- net/core/dst.c, line 214
- net/core/rtnetlink.c, line 526
- net/ipv4/devinet.c, line 1140
- net/ipv4/ipmr.c, line 1756
- net/ipv4/fib\_frontend.c, line 652
- net/ipv4/fib\_rules.c, line 466
- net/ipv4/netfilter/ip\_queue.c, line 647
- net/ipv4/netfilter/ipfwadm\_core.c, line 1385
- net/ipv4/netfilter/iptables\_MASQUERADE.c, line 190
- net/ipv4/af\_ipv4.c, line 2562
- net/netrom/af\_netrom.c, line 1311
- net/dechnet/af\_dechnet.c, line 2260
- net/dechnet/dn\_rules.c, line 363
- net/ipv6/ipv6\_sockglue.c, line 563
- net/ipv6/netfilter/ip6\_queue.c, line 703
- net/bridge/br.c, line 51
- net/econet/af\_econet.c, line 1125
- net/x25/af\_x25.c, line 1324
- net/rose/af\_rose.c, line 1463
- net/wanrouter/af\_wanpipe.c, line 2762
- net/packet/af\_packet.c, line 1896
- net/irda/af\_irda.c, line 2590
- net/atm/clip.c:
  - line 739
  - line 740
- net/atm/mpc.c, line 768
- net/8021q/vlan.c, line 99
- drivers/net/wan/lapbether.c, line 478
- drivers/net/hamradio/bpqether.c, line 614
- drivers/net/pppoe.c, line 1065
- drivers/net/bonding.c, line 2010

- register\_netdevice\_notifier
- dst\_dev\_event()
- rtnetlink\_dev\_notifier()
- ip\_netdev\_notifier()
- ip\_mr\_notifier()
- fib\_netdev\_notifier()
- fib\_rules\_notifier()
- ipq\_dev\_notifier()

## Actions associated with NETDEV\_REGISTER

net/core/dst.c, line 214

dst\_dev\_event()

Recall that the *dst\_entry/rtable* structures make up the route cache. No action is taken on REGISTER. On UNREGISTER/DOWN the *dst->output* function is set to *dst\_blackhole()*.

net/core/rtnetlink.c, line 526

rtnetlink\_event()

```
494     case NETDEV_REGISTER:
495         rtmsg_ifinfo(RTM_NEWLINK, dev, ~0U);
496         break;
```

net/ipv4/devinet.c, line 1140

inetdev\_event()

```
802     case NETDEV_REGISTER:
803         printk(KERN_DEBUG "inetdev_event: bug\n");
804         dev->ip_ptr = NULL;
805         break;
```

net/ipv4/ipmr.c, line 1756

ipmr\_device\_event()

Multicast routing support via *mrouted*.

net/ipv4/fib\_frontend.c, line 652

fib\_netdev\_event()

No action.

net/ipv4/fib\_rules.c, line 466

fib\_rules\_event()

```
388     else if (event == NETDEV_REGISTER)
389         fib_rules_attach(dev);
```

Recall that *fib\_rules* aren't in play unless IP\_MULTIPLE\_TABLES is configured.

net/ipv4/netfilter/ip\_queue.c, line 647

ipq\_rcv\_dev\_event()

No action is taken on REGISTER. The packet queue is dumped on DOWN.