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`.

```c
14 struct notifier_block
15 { 16   int (*notifier_call)(struct notifier_block *self, 
17       unsigned long, void *);
18   struct notifier_block *next;
19   int priority;
20 };
```

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.

```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.

```c
63  int notifier_chain_register(struct notifier_block **list, 
64       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.

```c
41 /* netdevice notifier chain */
42 #define NETDEV_UP 0x0001 
43 /* For now you can't veto a device up/down */
44 #define NETDEV_DOWN 0x0002
45 #define NETDEV_REBOOT 0x0003 
46    /* Tell a protocol stack a network interface detected a hardware crash and restarted 
47       - we can use this eg to kick tcp sessions 
48       once done */
49 #define NETDEV_CHANGE 0x0004 
50    /* Notify devstate change */
51 #define NETDEV_REGISTER 0x0005
52 #define NETDEV_UNREGISTER 0x0006
53 #define NETDEV_CHANGEMTU 0x0007
54 #define NETDEV_CHANGEADDR 0x0008
55 #define NETDEV_GOING_DOWN 0x0009
56 #define NETDEV_CHANGENAME 0x000A
57```
An example registration

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

```c
struct notifier_block rtnetlink_dev_notifier = {
    rtnetlink_event, // handler
    NULL,            // parameter
    0                 // priority
};

void __init rtnetlink_init(void)
{
    #ifdef RTNL_DEBUG
        printk("Initiank_init(void)
    #endif
    rtnl = netlink_kernel_create(NETLINK_ROUTE,
                                  rtnetlink_rcv);
    if (rtnl == NULL)
        panic("rtnetlink_init: cannot initialize
                rtnetlink\n");
    netlink_set_nonroot(NETLINK_ROUTE, NL_NONROOT_RECV);
    register_netdevice_notifier(&rtnetlink_dev_notifier);
    rtnetlink_links[PF_PACKET] = link_rtnetlink_table;
}
```
**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 used 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,  
   unsigned long val, void *v)  
{  
   int ret=NOTIFY_DONE;  
   struct notifier_block *nb = *n;  
126  
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.

```c
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/nsyms.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/rnetlink.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/ipt_MASQUERADE.c`, line 190
- `net/ipv4/fib.frontend.c`, line 2562
- `net/ipv4/netfilter/ip_queue.c`, line 647
- `net/ipv6/ipv6_sockglue.c`, line 563
- `net/ipv6/netfilter/ip6_queue.c`, line 703
- `net/bridge/br.c`, line 51
- `net/decnet/af_decnet.c`, line 2260
- `net/decnet/dn_rules.c`, line 363
- `net/ipv6/ipv6_sockglue.c`, line 563
- `net/bridge/br.c`, line 51
- `net/decnet/af_decnet.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/bpgether.c`, line 614
- `drivers/net/pppoe.c`, line 1065
- `drivers/net/bonding.c`, line 2010

Referenced (in 35 files total) in:
- `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

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()`.

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()`.

Multicast routing support via `mrouted`.

Recall that `fib_rules` aren't in play unless IP_MULTIPLE_TABLES is configured.

Recall that `fib_rules` aren't in play unless IP_MULTIPLE_TABLES is configured.

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