Networking Topics

Andi Kleen
SuSE Labs

Contents

- Non blocking Sockets
- Unix Sockets
- Netlink
- Error handling
- Path MTU discovery
- IPv6 sockets

Non blocking sockets

- Allows well scaling servers without threads
- Not much locking overhead (=none)
- Requires state machines
- fcntl(socketfd, F_SETFL, O_NONBLOCK);
- Needed to handle many sockets (threads are costly)

Network events

- Incoming data
- Socket ready for writing (socket buffer has room)
- Connection finished
- Error occurred
- Disconnect
- Urgent data arrived.

poll/select

- Ask the kernel in a main loop about events on the descriptors with poll(2)
- Process event, run state machine on socket and continue
- Copies a full table in and out the kernel
- Does not scale well: kernel and user has to walk big tables.
- Very portable and great for small servers.

Signals vs Realtime signals

- Signal is just a bit in a mask (cannot be lost)
- Many events compress into one bit
- Realtime signals between SIGRTMIN and SIGRTMAX
- Realtime signals carry data and are delivered in order
- Can go lost when the queue overflows

Queued SIGIO

- You get a signal for an event.
- Scales well, no big tables to copy or search.
- Kernel supplies siginfo to the signal handler
- Signals are tied to threads or process groups.

Queued SIGIO HOWTO

- fcntl(socketd, F_SETOWN, getpid())
- fcntl(socketfd, F_SETSIG, rtsig)
- SA_SIGINFO signal handler gets siginfo_t argument.
- siginfo- >si_fd contains fd
- On overflow you get SIGIO and use poll to pick up events.
- sigtimedwait is a nice main loop if you don’t want signal handlers.

Unix Sockets

- Some basics:
- Unix sockets are for local communication
- PF_UNIX; AF_UNIX in POSIX speak
- Two flavors: stream socket and datagram socket.
- Fast (your X runs through them)
- Commonly used for local desktop use (e.g. GNOME's Orbit ORB or X11)

Abstract namespace

- Socket endpoints of well known services are found via socket nodes in the filesystem.
- They do not go away after reboot or when the server crashes.
- There is no easy way to check if a server has crashed so recovery is difficult.
- Abstract namespace is a non portable trick to solve these problems

Abstract namespace 2

- How to use? Simply pass a 0 byte as the first character of the sockaddr_un.sun_path and then the abstract name.
- Abstract name only exists as a hash table internally.
- Goes away when the last reference is gone.
- Very simple semantics unlike file system objects

Control messages

- Berkeley and POSIX sockets support control messages since some time.
- Only works for SOCK_DGRAM sockets.
- Control messages are passed out of band with datagrams by the kernel.
- Sockets API supplies some standard macros to encode them.
- Standardized in POSIX/IPv6 API.

Control messages, what good for?

- Credentials passing for Unix sockets.
- File descriptor passing for Unix sockets.
- Setting and receiving interface index/TOS/TTL for IP and IPv6 packets.
- Sending and receiving IP options (alternative to RAW sockets)
- Sending and receiving IPv6 extension headers.

Credentials passing

- Often local servers want to check the user and group id of client processes.
- Management using group rights of file system sockets is clumsy and works only for well defined restrictions, not for logging.
- Credentials passing gives you the process and user and group id of the process that sent the message.
- Relatively portable if well encapsulated.

Credentials passing, HOWTO

- SO_PASSCRED enables sending of credentials.
- For connected SOCK_STREAM sockets: use the SO_PEERCRED getsockopt.
- For SOCK_DGRAM the senders can send an SCM_CREDENTIALS control message with the datagram. It contains pid/uid/gid
- Sender sets its own values, but kernel checks them. Root can override it. If client sends nothing the kernel fills in defaults.

File descriptor passing

- Passing file descriptors from one process to another (»remote dup«)
- Pass a SCM_RIGHTS control message via a PF_UNIX socket. It contains an fd array.
- Use at least a one byte message to carry it.
- Allows authentication servers for fd resources
- Allows you to avoid threads for more fault encapsulation.

Netlink

- Message based kernel/user space communication.
- Simple protocol to detect message loss (e.g. because of out of memory)
- User interface via PF_NETLINK sockets.
- Currently used for routing messages, interface setup, firewalling, netlink queuing, arpd, ethertap. Each has its own netlink family.

Netlink messages

- Has a common header with sequence number, type, flags, length, sender pid.
- Sender can request an ACK or an ECHO for reliability.
- Multipart messages are used for table dumps.
- Passes back a nlmsgerr message when a problem occurs.

Sending a netlink message

- Netlink message buffers are set up through macros from linux/netlink.h
- Find the length of the buffer using NLMSG_SPACE passing payload length
- Allocate a buffer. Setup nlmsghdr at beginning of buffer. Nlmsg_length is computed by NLMSG_LENGTH.
- Get a pointer to payload using NLMSG_DATA and set it up.

Receiving a netlink message

- Fill a buffer using recvmsg() from a netlink socket.
- First nlmsghdr is beginning of buffer.
- Check if it is not truncated using NLMSG_OK
- Check the type and it you're interested in it get the payload using NLMSG_DATA. For rtnetlink don't forget the rta attributes.
- Get next message using NLMSG_NEXT

Netlink multicast groups

- sockaddr_nl contains a nl_groups bitmask that allows 32 multicast groups.
- Groups are specific for the netlink family.
- Only root or the kernel can send to a multicast group.
- User processes bind to them.
- Useful for listening to updates of some common resource.

Rtnetlink

- Rtnetlink is used to configure the IP stack.
- Superset of the old ioctl interface.
- Can configure and watch interfaces, routes, IP addresses, routing rules, neighbours (ARP entries), queueing disciplines and other stuff.
- Kernel uses it internally (ioctls are turned into netlink)
- User interface in iproute2
- Some groups: Link, Neighbour, Route, Mroute, TC

Rtnetlink messages

- Messages start with a standard netlink header (struct nlmsghdr) and a type specific header.
- They come in NEW, GET, DEL flavours for each object that can be touched.
- GET can dump all objects in the database or only matching one.
- Messages carry attributes after the main headers.
- Attributes are like small netlink messages with a rta_attr header.

A few rtnetlink messages:

- NEW/GET/DEL
- ROUTE: struct rtmsghdr and describes a routing table entry. Has lots of attributes like RTA_GATEWAY, RTA_OIF, RTA_IIF etc.
- ADDR: struct ifaddrmsg and describes a local IP address. Has attributes like IFA_LOCAL (local IP), IFA_LABEL (alias name), etc.
- See include/linux/rtnetlink.h and rtnetlink(7) for a lot more messages and the details.

Some rtnetlink applications

- Waiting for interface up and down by binding to RTMGRP_LINK and watching for link up/down
  [when the network driver supports the netif_carrier* interface in 2.4 this allows HA failover and watching for network problems]
- Maintaining an copy of the routing table.
- Maintaining a table of the local IP addresses.
- ...

Kernel netlink

- Works using skbuffs.
- Sending can be non blocking (netlink_unicast/broadcast)
- User context calls callback
- netlink_dump calls your callback with a skbuff for RTM_GET
- netlink_ack acks packets if requested.

Netlink resources

- Man pages: netlink(7), rtnetlink(7)
- libnetlink from iproute2 for higher level interface and some utility functions
- /usr/include/linux/netlink.h
- /usr/include/linux/rtnetlink.h
- Examples: zebra, bird, iproute2

Error handling

- Networks generate errors (surprise!)
- They are generated locally, by routers on the path or by the target host.
- Some errors are fatal, others just need action (retransmit)
- Incoming errors from remote set a pending error on the socket that caused them and reported on the next operation on the socket.
- TCP does (nearly) all the work for you

Getting told about UDP errors

- For connected sockets you just get the pending error.
- For unconnected sockets that talk to multiple targets it is hard to find out where the error came from.
- Linux 2.2 added the error queue interface to solve the problem.
- Error queue is associated with a socket and stores errors.
- Error queue messages tell you where the error

How to process error messages

- Enable IP_RECVERR on the socket
- Do normal IO (sendmsg, recvmsg, etc.)
- On error do a recvmsg with MSG_ERRQUEUE and a msg_control buffer.
- Original destination is in msg_name, error message in a IP_RECVERR control message, original payload in msg_iov. Process it.
- Do another recvmsg/poll on the socket. If it has still an error set repeat.

Error queue messages

```
/* linux/errqueue.h */

struct sock_extended_error {
    u_int32_t ee_errno;     /* errno */
    u_int8_t  ee_origin;    /* Where it came from; see below */
    u_int8_t  ee_type;      /* ICMP type */
    u_int8_t  ee_code;      /* ICMP code */
    u_int32_t ee_info;      /* ICMP specific info (gateway or pmtu) */
    /* data follows */
};

enum { SOCK_EE_ORIGIN_NONE, SOCK_EE_ORIGIN_ICMP, ... };

struct sockaddr_in *SOCK_EE_OFFENDER(struct sock_extended_err *);
```

Path MTU discovery

- Path MTU is the biggest packet size that can go through a internet path without fragmentation
- Fragmentation is bad: slow, increases probability of packet loss, makes congestion avoidance harder, too much work for host and router.
- Path MTU is dynamic and changes.
- TCP does the work for you.
- For UDP/RAW the application has to size its packets correctly

How does PMTUdisc work?

- Sender starts with a reasonable packet size (interface MTU)
- Sets the Don' t Fragment bit in the IP header
- When a router would forward to a smaller MTU he drops it and sends pack a ICMP_FRAG_NEEDED message.
- Sender receives it.
- Readjusts its idea of the MTU and retransmits if appropiate.

PMTUdisc in Linux

- 2.2+ kernel automatically keeps track of path MTUs in a destination cache.
- Can be turned on/off per socket using IP_PMTU_DISCOVER.
- Can be retrieved using IP_PMTU, but only on connected sockets.

PMTUdisc: letting the kernel work

- Set IP_PMTU_DISCOVER to IP_PMTUDISC_WANT
- Connect a socket to destination and use IP_PMTU to retrieve the PMTU.
- Send packet.
- If EMSGSIZE get new MTU and send again
- Does not support retransmits for async events.
- Kernel keeps state for you.

PMTUdisc: keeping your own state

- Keep a table of destinations with MTU
- Set IP_PMTUDISC_WANT and IP_RECVERR
- Retrieve first MTU
- Send packet
- If EMSGSIZE process error queue.
- Set new MTU ee_data for destination (from address).

PMTUdisc problems

- PMTU blackholes by misconfigured firewalls that block ICMP. Check yours!
- Linux missing PMTU blackhole handling.
- PMTUs have to be timed out regularly because of routing changes.

IPv6 socket basics

- More complicated and bigger sockaddr_in6 (memset 0 it)
- Ports are shared with v4 and stay the same
- IPv4 can be used with the v6 API.

sockaddr_in6

- sin6_family: AF_INET6
- sin6_port
- sin6_flowinfo
- sin6_addr
- sin6_scope_id (not in Linux 2.2)
- Kernel transparently hides a IPv4 address if needed.

IPv6 search- n- replace

- sockaddr_in - > sockaddr_in6 (if nothing depends on the size)
- AF/PF_INET - > AF/PF_INET6
- INADDR_ANY - > in6addr_any (+memcpy)
- loopback - > in6addr_loopback

IPv6 name service

- getaddrinfo / freeaddrinfo. Resolves address and port.
- getnameinfo does reverse resolution.
- inet_pton to print IPv6 addresses

IPv6 references

- http://playground.sun.com/pub/ipng/html
- RFC2133 (Basic API)
- RFC2292 (Extended API)