Unix and Minix Networking

- Network Protocols
- Unix networking
- Minix networking

Communication Protocols

- Protocol: a set of rules for communication that are agreed to by all parties
- Protocol stack: networking software is structured into layers
  - Each layer N, provides a service to layer N+1, by using its own layer N procedures and the interface to the N-1 layer.
  - Example: International Standards Organization/Open Systems Interconnect (ISO/OSI)
TCP/IP Protocol Stack

- It has fewer layers than ISO to increase efficiency.
- Consists of a suite of protocols: UDP, TCP, IP...
- TCP is a **reliable** protocol -- packets are received in the order they are sent.
- UDP (user datagram protocol) is an **unreliable** protocol (no guarantee of delivery).

Socket Communication

- Client-server socket communication and Socket primitives
  - Berkeley sockets (BSD Unix)
Berkeley Socket Primitives

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket</td>
<td>Create a new communication endpoint</td>
</tr>
<tr>
<td>Bind</td>
<td>Attach a local address to a socket</td>
</tr>
<tr>
<td>Listen</td>
<td>Announce willingness to accept connections</td>
</tr>
<tr>
<td>Accept</td>
<td>Block caller until a connection request arrives</td>
</tr>
<tr>
<td>Connect</td>
<td>Actively attempt to establish a connection</td>
</tr>
<tr>
<td>Send</td>
<td>Send some data over the connection</td>
</tr>
<tr>
<td>Receive</td>
<td>Receive some data over the connection</td>
</tr>
<tr>
<td>Close</td>
<td>Release the connection</td>
</tr>
</tbody>
</table>

Linux Network Architecture

- File access path versus socket access path

Slides 6-12 courtesy of Raoul Rivas, UIUC
Sockets in Linux kernel

- Contains sys calls like socket, connect, accept
- Implements POSIX socket interface
  - independent of protocols
- Maps socket data structures to integer handlers
- Calls lower layer functions
  - sys_socket()—>sock_create

Protocol Families

- Implements different socket families: INET, UNIX
- Extensible through modules and fn pointers
- Calls net_proto_family—>create for family-specific initialization
Protocols

- Families have multiple protocols
  - INET: TCP, UDP
- Protocol functions stores in proto_ops
- Some functions unused in a protocols: dummy fins
- Some functions same across protocols: shared

Packet Creation

- At sending function, packetize the buffer
- Packets represented as sk_buff data structure
- Contains pointers to
  - transport layer header
  - link layer header
  - received timestamp
  - Device that received it
Fragmentation and Routing

- Fragmentation is performed inside `ip_fragment`
- route filled in by `ip_route_output_flow`
- Routing mechanisms used
  - Route cache
  - Forwarding Information Base
  - Slow routing

Data Link Layer

- Responsible for packet scheduling
- `dev_queue_xmit` enqueues packets for transmission
  - `qdisc` of device
- Send in process context
- If device bust, schedule for later
- `dev_hard_start_xmit` sends to device
**NIX Networking Commands**

- Ethernet MAC address: d0:73:d5:2a:12:51
- IP address: 192.168.1.2 or 128.119.240.2
- ping
- ifconfig

```
# ifconfig

/dev/ip: address 10.0.2.15 netmask 255.255.255.0 mtu 1500
```

- Linux: netstat -rn
- Linux: route

<table>
<thead>
<tr>
<th>Destination</th>
<th>Gateway (Netmask)</th>
<th>Flags</th>
<th>MSS Window</th>
<th>Irtt</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.0.0</td>
<td>192.168.1.1 (0.0.0.0)</td>
<td>UG</td>
<td>0 0</td>
<td>0 eth0</td>
<td></td>
</tr>
<tr>
<td>0.0.0.0</td>
<td>192.168.55.100 (0.0.0.0)</td>
<td>UG</td>
<td>0 0</td>
<td>0 14br0</td>
<td></td>
</tr>
<tr>
<td>169.254.0.0</td>
<td>0.0.0.0 (255.255.0.0)</td>
<td>U</td>
<td>0 0</td>
<td>0 14br0</td>
<td></td>
</tr>
</tbody>
</table>

**Minix INET**

- "inet" system process handles networking in Minix
  - Source code "servers/inet"
- Implements ethernet layer, IP layer and TCP/UDP
- Ethernet card is a I/O device
  - Device driver is in "drivers"
    - e1000 is Intel gigabit driver
- TCP/IP code is in "inet" and "inet-generic"

http://www.nyx.net/~ctwong/minix/  note: minix v2, not v3
Data link Layer

- Hardware: ethernet, modem etc
- Can have more than one device (major and minor #)
- ioctl call used to set parameters such as comm speed

- The driver itself runs as a user process

- I/O Involves: VFS, INET and driver process
  - same concept as any block device driver

INET Server

- inet.c - main function for INET Server
  - handles various message types from VFS and DL_ETH

<table>
<thead>
<tr>
<th>m_type</th>
<th>DEVICE</th>
<th>PROC_NR</th>
<th>COUNT</th>
<th>POSITION</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEV_OPEN</td>
<td>minor dev</td>
<td>proc nr</td>
<td>mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEV_CLOSE</td>
<td>minor dev</td>
<td>proc nr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEV_IOCTL_S</td>
<td>minor dev</td>
<td>proc nr</td>
<td>cmd</td>
<td>nwido..</td>
<td>address</td>
</tr>
<tr>
<td>DEV_READ_S</td>
<td>minor dev</td>
<td>proc nr</td>
<td>count</td>
<td></td>
<td>address</td>
</tr>
<tr>
<td>DEV_WRITE_S</td>
<td>minor dev</td>
<td>proc nr</td>
<td>count</td>
<td></td>
<td>address</td>
</tr>
<tr>
<td>CANCEL</td>
<td>minor dev</td>
<td>proc nr</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
INET Server

- **buf.c** - buffering code to allocate data for sending and receiving network packets
- **mnx_eth.c** - code for sending and receiving ethernet frames to/from ethernet driver
- **inet_config.c** - configure networking devices
  - /dev/eth, /dev/ip, dev/tcp, /dev/udp
- **mq.c** — message queue structure
  - mq_list is message queue and mq_t is one message entry

- **sr.c** - code to interface with file system
  - DEV_OPEN, DEV_CLOSE, DEV_READ, DEV_WRITE...
- **generic/udp.c** - code for UDP protocol
  - udp_port data structure is used for a UDP socket port
- **generic/tcp.c** - code for TCP protocol
  - tcp_port data structure used to TCP socket port