


**Exercise Course for Telematics:  
Computer Networks and the Internet**  
(Winter Semester 2002/03)

Prof. Dr. Dieter Hogrefe  
Dipl.-Inform. Michael Ebner  
Dr. Xiaoming Fu (fu@cs.uni-goettingen.de)

Telematics group  
University of Göttingen, Germany



Telematics group  
University of Göttingen, Germany

**Overview of Sockets: the TCP/IP user interface**

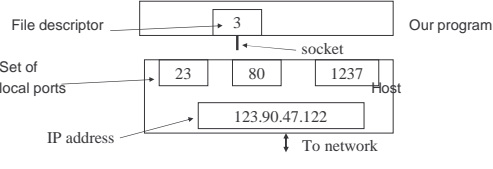
- Socket: Create a new communication end point
- Bind: Attach a local address to a socket
- Listen: Announce willingness to accept connections
  - Give queue size
- Accept: Block the caller until a connection attempt arrives
  - Accept next connection request
- Connect: Actively attempt to establish a connection
- Send: Send some data over the connection
- Recv: Receive some data from the connection
- Close: Release the connection

WS 2002/03 2

Telematics group  
University of Göttingen, Germany

**Step 1: Create a socket**

- Suppose we make the call:  
s = **socket**( ... assorted parameters here ... );  
and the result is file descriptor s = 3
- We have the following situation:

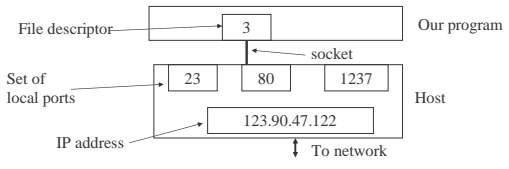


WS 2002/03 3

Telematics group  
University of Göttingen, Germany

**Step 2: Bind the socket to a port**

- Suppose we do:  
result = **bind**( ... 80, ... );  
and the result is not -1 (that is, it worked!)
- We now have the following situation:

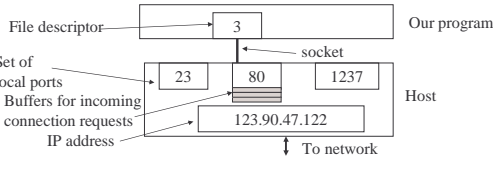


WS 2002/03 4

Telematics group  
University of Göttingen, Germany

**Step 3: Listen for connections**

- Suppose we do:  
result = **listen**( 2 );  
and the result is not -1 (that is, it worked!)
- We now have the following situation:

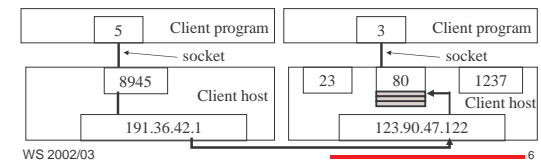


WS 2002/03 5

Telematics group  
University of Göttingen, Germany

**Step 4: Incoming connection request**

- Suppose we do:  
name.sin\_port = 80;  
name.sin\_addr.s\_addr = inet\_addr("123.90.47.122");  
result = **connect**( s, name, sizeof( name ) );  
and the result is not -1 (that is, it worked!)
- We now have:

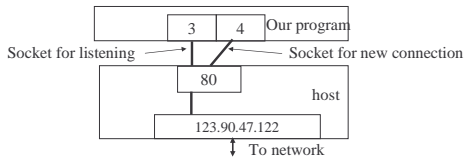


WS 2002/03 6

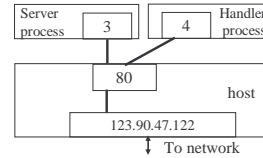


### Step 5: Accept a connection request

- Suppose we do:  
newSocket = **accept**( s, addr, &addr\_length );  
and the result is a new socket with file descriptor 4.
- We now have the following situation:



### But, what we really want is...



- Why?
  - Server can handle multiple, simultaneous connection requests
  - A separate handler process can be created for each request



### Programming with sockets

- **Styles of communication:**
  - stream: reliable, two-way byte streams
  - datagram: unreliable, two-way record-oriented
  - etc.
- **Communication domains**
  - UNIX
    - endpoints (sockets) named with file-system pathnames
    - supports stream and datagram
  - Internet
    - endpoints named with IP addresses
    - supports stream and datagram
  - others
- **Protocols**
  - e.g., TCP/IP, UDP/IP



### Using Datagram Sockets (1)

- **Receiver steps**
  - 1) create socket**
    - **socket system call**  
int socket(int domain, int type, int protocol);  
fd = socket(PF\_UNIX, SOCK\_DGRAM, 0);
  - 2) set up receiver's name**
    - **put name in sockaddr\_un structure**  
struct sockaddr\_un {  
short sun\_family; /\* PF\_UNIX \*/  
char sun\_path[108]; /\* path name \*/  
} name;  
  
name.sun\_family = PF\_UNIX;  
memcpy(name.sun\_path, path, strlen(path));



### Using Datagram Sockets (2)

- **Receiver steps (continued)**
  - 3) bind receiver's name to socket**
    - bind takes a generic *struct sockaddr* argument and is given the combined length of the first part of the structure and that portion of the second part that is used

```
name_len = sizeof(name.sun_family) +
strlen(name.sun_path);
bind(fd, (struct sockaddr *)&name, name_len);
```



### Using Datagram Sockets (3)

- **Receiver steps (continued)**
  - 4) receive (and send) data**
    - use *recvfrom* system call to obtain caller's address

```
int recvfrom(int s, char *buf, int len, int flags, struct
sockaddr *from, int *fromlen);
struct sockaddr_un sender_name;
int sender_len = sizeof(sender_name);

recvfrom(fd, buf, sizeof(buf), 0,
(struct sockaddr *)&sender_name, &sender_len);
```



## Using Datagram Sockets (4)

- **Sender steps**
  - 1) create socket
  - 2) set up sender's name (optional)
  - 3) bind sender's name to socket (optional)

WS 2002/03

13



## Using Datagram Sockets (5)

- **Sender steps (continued)**
  - 4) set up receiver's name

```
struct sockaddr_un recvr_name;
int recvr_len;

recvr_name.sun_family = PF_UNIX;
memcpy(recvr_name.sun_path, path, strlen(path));
recvr_len = sizeof(recvr_name.sun_family) +
    strlen(recvr_name.sun_path);
```

WS 2002/03

14



## Using Datagram Sockets (6)

- **Sender steps (continued)**
  - 5) send (and receive) data
  - use *sendto* system call to send datagram

```
int sendto(int s, const char *msg, int len, int flags, const
struct sockaddr *to, int tolen);
```

```
sendto(fd, buf, sizeof(buf), 0,
(const struct sockaddr *)&recvr_name, recvr_len);
```

WS 2002/03

15



## Using Stream Sockets (1)

- **Server steps**
  - 1) create socket
  - *socket* system call

```
int socket(int domain, int type, int protocol);
fd = socket(PF_INET, SOCK_STREAM, 0);
```

WS 2002/03

16



## Using Stream Sockets (2)

- **Server steps (continued)**
  - 2) set up server's address
  - put "wildcard" internet address in *sockaddr\_in* structure
    - server may have multiple interfaces; we want to be able to receive on all of them
    - must convert from *host byte order* to *network byte order*

```
struct sockaddr_in {
short sin_family;
u_short sin_port;
struct in_addr sin_addr;
char sin_zero[8]; /* padding */
} my_addr;
```

```
my_addr.sin_family = PF_INET;
my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
my_addr.sin_port = htons(port);
```

WS 2002/03

17



## Using Stream Sockets (3)

- **Server steps (continued)**
  - 3) bind server's name to socket

```
bind(fd, (struct sockaddr *)&my_addr,
sizeof(my_addr));
```
- 4) set up socket to be in listening mode
  - *backlog* is the max length of the queue of waiting connections

```
int listen(int fd, int backlog);
```

WS 2002/03

18

Telematics group  
University of Göttingen, Germany

## Using Stream Sockets (4)

- **Server steps (continued)**
  - 5) wait for a connection
    - when connection is received, a new socket is created for communication on it
    - as an option, the address of the connector (client) is passed back
 

```
int accept(int fd, struct sockaddr *addr, int *addrlen);
```
  - 6) receive and send data
    - simple *read* and *write* system calls can be used

WS 2002/03 19

Telematics group  
University of Göttingen, Germany

## Using Stream Sockets (5)

- **Client steps**
  - 1) create socket
    - as an option, one can bind the socket to a particular port number
    - port numbers less than 1024 are reserved for privileged users

WS 2002/03 20

Telematics group  
University of Göttingen, Germany

## Using Stream Sockets (6)

- **Client steps (continued)**
  - 2) find internet address of the server
    - look it up in *domain name service* (DNS)
    - each host may have a list of interfaces; we choose the first one

```
struct hostent *hostinfo;
hostinfo = gethostbyname("botrytis.cs.brown.edu");
memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = PF_INET;
memcpy(&server_addr.sin_addr,
hostinfo->h_addr_list[0], hostinfo->h_length);
server_addr.sin_port = htons(port);
```

WS 2002/03 21

Telematics group  
University of Göttingen, Germany

## Using Stream Sockets (7)

- **Client steps (continued)**
  - 3) connect to server
 

```
connect(fd, (struct sockaddr *)&server_addr,
sizeof(server_addr));
```
  - 4) send and receive data
    - simple *write* and *read* system calls can be used

WS 2002/03 22

Telematics group  
University of Göttingen, Germany

## Exercise: Basic Socket Programming

- Objective: to write a client-server program that does simple file transfer across network.
- Two parts: client and server
  - The client requests the server send a specified file from the computer where the server resides. When the client receives the file, the file is saved on the local disk where the client is running.
- Hints:
  - The server listens to a specific port (non-reserved one, i.e., above 1024) that has been mutually agreed upon between the client and the server. When the server receives a request from a client, the request should contain the target file name with the complete path.
  - The server responds to the client in two steps:
    - First, the server sends back an acknowledgment indicating whether or not the file requested by the client can be transferred. If the requested file cannot be transferred (e.g. the file is not readable, the file does not exist), the client should print an error message and quit the connection. If the acknowledgment says the file is transferable
    - Second, the server actually sends the file to the client. The client will save the file to the local disk.
- Basic client-server example: `http://user.informatik.uni-goettingen.de/~fu/teaching/prog/socket_tcp/, socket_udp/`

WS 2002/03 23


Telematics group  
University of Göttingen, Germany

## Understanding TCP/IP Configuration Files

- Configuration files
  - `/etc/hostname`
  - `/etc/hosts`
  - `/etc/services`
  - `/etc/host.conf`
  - `/etc/nsswitch.conf`
  - `/etc/resolv.conf`
- Exercise:
  - Find out the transport protocol and its port:
    - ftp, telnet, smtp, http, rsvp, rtp
  - Find out the dns server(s) used by your computer

WS 2002/03 24

Telematics group  
University of Göttingen, Germany




## /etc/hostname

- Only one line: main name of your machine

WS 2002/03 25

Telematics group  
University of Göttingen, Germany




## /etc/hosts

- Mapping between IP address(es) and hostname(s)
  - To easily remember the machines you may connect to
  - Typically, composed of your machine, machines in your LAN, other frequently used machines
  - Others: via DNS, specified in /etc/resolv.conf
- Example:
 

```
127.0.0.1 localhost
192.168.1.1 mycomputer
192.168.1.2 server
192.168.1.3 router
192.168.3.45 othercomputer otheralias
212.58.224.56 www.bbc.com
```

WS 2002/03 26

Telematics group  
University of Göttingen, Germany



## /etc/services


- Maintains the mapping between port number and service name.
  - Used by several system programs, e.g., inetd
    - Try to explain /etc/inetd.conf, e.g.,  

```
telnet stream tcp nowait root /usr/sbin/in.telnetd telnetd
```
  - RFC 1700 specifies well-known ports

http ←	Port 80
Telnet ←	Port 23
SMTP ←	Port 25
ftp ←	Port 21
ssh ←	Port 22

WS 2002/03 27

Telematics group  
University of Göttingen, Germany



## /etc/host.conf, /etc/nsswitch.conf

- Specifies the order to find name service
- Example:
  - /etc/host.conf
 


```
order hosts, bind à First look up in file host.conf, then per DNS
multi on à If several hosts are matched, return all
```
  - /etc/nsswitch.conf
 

```
passwd: compat
group: compat
shadow: compat
hosts: files dns
networks: files
protocols: db files
services: db files
ethers: db files
rpc: db files
```

    - First look up in local files, otherwise DNS

WS 2002/03 28

Telematics group  
University of Göttingen, Germany




## /etc/resolv.conf

- Specifies the order and address(es) DNS servers
- Example:
 

```
domain ifi.informatik.uni-goettingen.de à hosts without domain limitation
search ifi.informatik.uni-goettingen.de int.informatik.uni-goettingen.de à hosts without domain limitation, order: ifi first, then int
nameserver 134.76.80.11 à first DNS server
nameserver 134.76.81.12 à secondary DNS server
```

WS 2002/03 29

Telematics group  
University of Göttingen, Germany



## Familiar with TCP/IP Configuration Commands

- /sbin/ifconfig
- /sbin/route
- /sbin/netstat
- Exercise:
  - Find your network interfaces and the gateway of our university

WS 2002/03 30

Telematics group  
University of Göttingen, Germany

## /sbin/ifconfig

- Config / check your network interface(s):
  - IP address
  - Network mask
  - Broadcast address
  - Others

1. Basic configuration operation (root):  
`ifconfig interface IP-address [netmask <netmask>] [broadcast broadcast-address]`
2. Enable, Disable an interface (root):  
`ifconfig interface down`  
`ifconfig interface up`

WS 2002/03 31

Telematics group  
University of Göttingen, Germany

## /etc/ifconfig (cont.)

3. Check status of your network interface(s).

Example:

```

/sbin/ifconfig -a
eth0 - Link encap:Ethernet HWaddr 00:06:5B:3E:52:70
      inet addr:134.76.81.12 Bcast:134.76.81.255 Mask:255.255.255.0
      UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
      RX packets:118420883 errors:91 dropped:0 overruns:0 frame:0
      TX packets:100970495 errors:0 dropped:0 overruns:0 carrier:0
      collisions:19418174 txqueuelen:100
      RX bytes:3928383102 (3.6 GiB) TX bytes:1447082870 (1.3 GiB)
      Interrupt:16 Base address:0xc000
lo - Link encap:Local Loopback
      inet addr:127.0.0.1 Mask:255.0.0.0
      UP LOOPBACK RUNNING MTU:16436 Metric:1
      RX packets:26038674 errors:0 dropped:0 overruns:0 frame:0
      TX packets:26038674 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:0
      RX bytes:2025713619 (1.8 GiB) TX bytes:2025713619 (1.8 GiB)
  
```

WS 2002/03 32

Telematics group  
University of Göttingen, Germany

## /sbin/route

- Operates the routing table  
`route [options] [add|del] [parameter]`

1. Browse routing table
  - The gateway for the packets sent to the destination
  - The network mask of the route (here to the gateway)

```

/sbin/route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
134.76.81.0 * 255.255.255.0 UG 0 0 0 eth0
default 134.76.81.254 0.0.0.0 UG 0 0 0 eth0
  
```

```

/sbin/route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
134.76.81.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
0.0.0.0 134.76.81.254 0.0.0.0 UG 0 0 0 eth0
  
```

U: route is ready  
G: thru gateway  
H: dest is a host

WS 2002/03 33

Telematics group  
University of Göttingen, Germany

## /sbin/route (cont.)

2. Add and delete routes (root)  
`/sbin/route add [del [-net | -host] <target> [gw <gw>] [netmask <netmask>] [[dev] <interface>]`

Example:

```

/sbin/ifconfig eth0 134.76.81.254 netmask 255.255.255.0 broadcast 134.76.81.255
/sbin/route add - net 134.76.81.0

/sbin/route del - net 134.76.81.0

/sbin/route add - host 134.76.81.0
  
```

WS 2002/03 34

Telematics group  
University of Göttingen, Germany

## /bin/netstat

- Check status of TCP/IP services

```

/bin/netstat
-a (all)
-e (extended)
-r (routes)
-i (interfaces)
  
```

WS 2002/03 35

Telematics group  
University of Göttingen, Germany

## Familiar with TCP/IP Diagnosis Tools

- Diagnosis Tools:
  - ping
  - host, dnslookup
  - traceroute
  - Tcpdump
- Exercise:
  - Check your self-defined packet to [www.mit.edu](http://www.mit.edu), and explain why/where there is a significant change of the transmission delay.
  - Find the topology from your computer to [www.dfn.de](http://www.dfn.de)

WS 2002/03 36



## ping

- Based on ICMP to check reachability
- Options
  - ping -c count
  - ping -n (only display IP address)
  - ping -r (record route)
  - ping -q (only display the final statistical results)
  - ping -v (detailed info)

```
ping www.mit.edu
PING DANDELION-PATCH.mit.edu (18.181.0.31): 56 data bytes
64 bytes from 18.181.0.31: icmp_seq=0 ttl=241 time=174.2 ms
64 bytes from 18.181.0.31: icmp_seq=1 ttl=241 time=157.0 ms
64 bytes from 18.181.0.31: icmp_seq=2 ttl=241 time=154.0 ms
64 bytes from 18.181.0.31: icmp_seq=3 ttl=241 time=159.7 ms

--- DANDELION-PATCH.mit.edu ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 154.0/161.2/174.2 ms
```



## traceroute

- Principle: uses increasing TTL to send UDP packets, returning ICMP responses when detecting packet loss in a gateway
- Can be used to detect the bottleneck / congestion status
- Example:

```
traceroute www.tu-clausthal.de
traceroute to www.tu-clausthal.de (139.174.2.88), 30 hops max, 38 byte packets
 1 134.76.81.254 (134.76.81.254) 0.622 ms 0.573 ms 0.524 ms
 2 ge-theol-sued1.gwdg.de (134.76.249.145) 1.054 ms 0.516 ms 0.398 ms
 3 ge-gwdg1-theol.gwdg.de (134.76.249.105) 0.542 ms 0.496 ms 0.478 ms
 4 cl2012-int.gwdg.de (134.76.249.201) 0.525 ms 0.477 ms 0.438 ms
 5 ar-goettingen1.g-win.dfn.de (188.1.46.193) 38.824 ms 41.363 ms 41.853 ms
 6 ar-goettingen2.g-win.dfn.de (188.1.89.130) 46.177 ms 48.924 ms 48.533 ms
 7 7200vx.rz.tu-clausthal.de (139.174.251.12) 55.102 ms 54.797 ms 50.701 ms
 8 r-vlan-tuc.rz.tu-clausthal.de (139.174.253.254) 53.586 ms 55.611 ms 50.63 3 ms
 9 www.tu-clausthal.de (139.174.2.88) 54.176 ms 55.207 ms 53.562 ms
```



## tcpdump

- Similar to a packet sniffer (root)
  - Detect any packets passing by this node
  - Store these information and give a statistical report
- Typical usage:  
`/usr/sbin/tcpdump [-i interface] [src host <sh>] [dst host <dh>] [src port <sp>] [dst port <dp>] [tcp|udp|icmp]`



## Exercise: Streaming File Transfer

- Objective: implementing a file server and “streaming media” client
  - You should let the file transfer complete quickly, and let the file played back concurrently, beginning as soon as a minimum amount of data has been received.
  - You can't simply use xmms `http://www.cs.uni-goettingen.de/~fu/tmp/song.mp3`, because xmms has no input buffer and hence it will have to stay connected to the server until it exits.
- Work individually
- Reading:  
<http://www.cs.uni-goettingen.de/~fu/teaching/stream.html> (TBD)