Introduction to Network Programming using C/C++

Slides partly prepared by Olaf Bergmann (Uni Bremen TZI)

Would be giving brief introduction on...

Parsing Command line

Socket Related Address Structures

Host Name / IP Address resolution

Socket Creation

Making TCP and UDP Connection

Sending and Receiving Data

Mulitcasting

Multiplexing I/O

Parse Command Line

`int getopt(cnt,argv,optstring)`

```
int oc;
while( (oc=getopt(argc,argv,"a:bi:sl:D:t:")) != -1)
{
  switch(oc) {
    case 'a' : addAddress(optarg); break;
    case 'b' : usage(); exit(0);
    case 'i' : addInterface(optarg); break;
    case 's' : summary = true; break;
    case 'l' : dumplen = GetInt(optarg); break;
    case 't' : controlAddress(optarg); break;
    case 'D' : duration = GetInt(optarg); break;
    default :
      opterr(oc);
  }
}
```

Address Structures

```
struct sockaddr_in {
     uint8_t          sin_len;    /* length of structure (16) */
     sa_family_t      sin_family; /* AF_INET */
     in_port_t        sin_port;   /* 16-bit TCP or UDP port number */
     struct in_addr   sin_addr;   /* 32-bit IPv4 address */
     char             sin_zero[8];
};
struct in_addr {
  in_addr_t           s_addr;     /* 32-bit IPv4 address */
};
struct sockaddr {
     uint8_t          sa_len;
     sa_family_t      sa_family;    /* address family: AF_xxx value */
     char             sa_data[14];  /* protocol-specific address */
};
```

Address Structures Contd...

bind(), recvfrom() and sendto() function uses sockaddr structure

A normal practice is to fill the stuct sockaddr_in and cast the pointer to struct sockaddr while socket operartions

```
struct hostent {
    char            *h_name;
    char            **h_aliases;
    int             h_addrtype;
    int             h_length;
    char            **h_addr_list;
    char            *h_addr;
    };
```

gethostbyname() returns the resolved address in struct hostent format. A hostname may have multiple interfaces, so hostent structure is designed to hold the multiple addresses of the resolved hostname

Address Conversion functions (1)

Dotted decimal notation: aaa.bbb.ccc.ddd (IPv4 only)

```
in_addr_t inet_addr (char *buffer)
in_addr_t inet_aton (char *buffer)
char *inet_ntoa (in_addr_t ipaddr)
```

aaa.bbb.ccc.ddd (IPv4), aaaa:bbbb:cccc:dddd:eeee:ffff:gggg:hhhh (IPv6)

```
int inet_pton(int af, const char *src, void *dst)
```

dst: in_addr or in6_addr

```
const char *inet_ntop(int af, const void *src, char *dst, size_t)
```

src: in_addr bzw. in6_addr
char dst[INET_ADDRSTRLEN] bzw. char dst[INET6_ADDRSTRLEN]

gethostbyname() - converts hostname (xyz.hut.fi) to struct hostent format

Conversion Functions (2)

Network vs. Host Byte Order

- All data in the network is sent as “Big Endian”
- Conversion into local representation required (Intel)
  - (depends on the CPU architecture but should always be done for portability)

```
netshort  = htons (hostshort)
netlong   = htonl (hostlong)
hostshort = ntohs (netshort)
hostlong  = ntohl (netlong)
```

BSD Socket Interface

The BSD mechanism for Inter-Process Communication (IPC)

Transparency between local and remote communications

Socket Descriptor: feels like file i/o or stdin/stdout

Supports different types of communications, u.a.

SOCK_STREAM: TCP	SOCK_DGRAM: UDP
SOCK_RAW: Raw IP	SOCK_PACKET: Link-Layer-Frames

Socket Creation

```
int socket(domain,type,proto)
int bind(sd,addr,addrlen)
```

```
int createSocket(const sockaddr_in &addr)
{
    int sd=socket(AF_INET,SOCK_DGRAM,0);
    if (sd<0) return -1;

    int yes = 1;
    setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, (char*)&yes, sizeof yes);
    fcntl(sd,F_SETFL,O_NONBLOCK);
    if (bind(sd,reinterpret_cast<const sockaddr *>(&addr),sizeof addr)<0) {
        std::cerr << strerror(errno) << std::endl;
        return -1;
    }
    return sd;
}
```

Socke t domain
AF_INET, PF_INET6
Socke t type
SOCK_STREAM, SOCK_DGRAM, …
Protocol
0 (a ny), 6 (tcp), 17 (udp)

Creating UDP and TCP connections

UDP:

Create a socket with SOCK_DGRAM

Bind the socket to a address (particular IP and port number)

Ex- **bind (int sd, struct sockaddr *, socklen_t len);**

Now the socket can be used for send and receive operations

TCP:

Create a socket with SOCK_STREAM

Bind the socket to a address (particular IP and port number

If program need to accept any connection request, then listen on the socket

Listen() - allows to specify the number of backlogs of connection requests that can be buffered

Connections (TCP) contd..

```
connect (int sd, struct sockaddr *target, socklen_t len);
```

Creates (synchronously) a connection

Function call only complete when the connection is established, if a timeout occurs without response (may be several minutes), or when ICMP error messages indicate failure (e.g., destination unreachable)

Accepting an incoming connection (cannot reject anyway)

```
new_sd = accept (int sd, struct sockaddr *peer, socklen_t *peerlen);
```

Creates a new socket descriptor for the new connection

The original one (sd) continues to be used for accepting further connections

Closing a connection

`shutdown (int sd, int mode)`

0: no further sending, 1: no further reception, 2: neither sending nor receiving

`close(sd)` to clean up – beware of data loss!

Sending Data

Connection-oriented (TCP)

```
write (int sd, char *buffer, size_t length);
writev (int sd, struct iovec *vector, int count);
```

List of buffers, each with pointer to memory and length

```
send (int sd, char *buffer, size_t length, int flags)
```

May be used for out-of-band data

Connectionless (UDP)

```
sendto (int sd, char *buffer, size_t length, int flags,
        struct sockaddr *target, socklen_t addrlen)
sendmsg (int sd, struct msghdr *msg, int flags)
```

Target address

Pointer to the memory containing the data

Control information

Receiving Data

Connection-oriented (TCP)

```
read (int sd, char *buffer, size_t length);
readv (int sd, struct iovec *vector, int count);
```

List of buffers, each with pointer to memory and length

```
recv (int sd, char *buffer, size_t length, int flags)
```

May be used for out-of-band data

Connectionless (UDP)

```
recvfrom (int sd, char *buffer, size_t length, int flags,
          struct sockaddr *target, socklen_t addrlen)
recvmsg (int sd, struct msghdr *msg, int flags)
```

Sender address

Pointer to the data

Control information

Further Functions

`getpeername (int sd, struct sockaddr *peer, size_t *len)`

Obtain the address of the communicating peer

`getsockname (int sd, struct sockaddr *local, size_t *len)`

Obtain the address of the local socket (e.g., if dynamically assigned)

Modify socket parameters

`getsockopt (int sd, int level, int option_id, char *value, size_t length)`

`setsockopt (int sd, int level, int option_id, char *value, size_t length)`

Examples:

Buffer size, TTL, Type-of-Service, TCP-Keepalive, SO_LINGER, ...

`fcntl (int sd, int cmd [, long arg] [, ...]);`

Non-blocking I/O

Multicast reception

Multicast JOIN

```
setsockopt (sd, IPPROTO_IP, IP_ADD_MEMBERSHIP,
  struct ip_mreq *mreq, sizeof (ip_mreq));
struct ip_mreq {
  struct in_addr imr_multiaddr;     /* IP multicast address of
                                       group */
  struct in_addr imr_interface;     /* local IP address of
                                    interface */
};
```

Multicast-LEAVE

```
setsockopt (sd, IPPROTO_IP, IP_DROP_MEMBERSHIP, struct
ip_mreq *mreq, sizeof (ip_mreq));
```

Optional: Allow repeated use of an address (needed for multicasting)

```
char one = 1;
setsockopt (sd, SOL_SOCKET, SO_REUSEADDR, &one, sizeof
(char))
```

I/O Multiplexing (select)

```
int select(maxfdset,read,write,ext,timer)
```

```
Calculate file descriptor sets (FDSET)
Determine earliest timeout
Call select()
Error?
    Fatal - Terminate
    Repairable (e.g. interrupted system call) - repeat
Timeout?
    Timer handling; use struct timeval { … } to specify (sec, usec) pair
    NULL pointer == blocking (no timeout), (0, 0) == polling
Success
    Determine active file descriptors and handle events
```

fd_set Makros used by select

```
fd_set base_set working_set;
FD_ZERO (&working_set);
FD_SET (fd, &base_set);
   .
   .
   .
if (FD_ISSET(fd, &working_set))
    . . .
```

Select() example

```
.
rc_select = select (sd + 1, &working_set, NULL, NULL, &select_timeout);
/* Check to see if the select call failed. */
if (rc_select < 0)
{
  perror("select() failed");
  check errorno and act accordingly
}
/* Check to see if the 'n' minute time out expired.       */
if (rc_select == 0)
{
  fprintf(stderr, "\n select() timed out. \n");
  return -1;
}
/* Check to see if there is a incoming connection request  */
if (FD_ISSET(sd, &working_set))
{
        .......
        .......
```

I/O Multiplexing (poll)

```
int poll(pollfd,n_fd,timeout)
```

```
struct pollfd {
        int     fd;             // file descriptor
        int     events;  // events to watch for
        int     revents; // occurred events
};
```

Poll events:

POLLIN	input pending
POLLOUT	socket writable (only needed with non-blocking i/o)
POLLHUP, POLLERR	

Timeout is specified in milliseconds

- -1 == no timeout, 0 == return immediately (perform real polling)

Handling otherwise identical to select()