Networks lab

headers :-

<unistd.h>
<sys/types.h>
<sys/socket.h>
<netinet/in.h>
<arpa/inet.h>.

htons

h tons
↕
inet_ntoa

n tohs

1) Sockets
   ↳ sockfd ← file-descriptors

IPV4 internet protocol.

$ sockfd = socket (AF_INET, SOCK_STREAM, 0)

if sockfd < 0 then socket could not be created.

int socket (int domain, int type, int protocol)

Socket-family
↳ AF_INET ✓
AF_IPX
AF_PACKET

SOCK_STREAM (TCP) ✓
SOCK_DGRAM (UDP)

fcntl command
↳ F_SETOWN,

EACCES error → Not allowed to create a socket

2) struct sockaddr_in → (netinet/in.h)
   ↳ {
   short sin_family;
   unsigned short sin_port;
   struct in_addr sin_addr;
   char sin_zero[8];
   }

struct in_addr { unsigned long s_addr; };

s. sin_family = AF_INET (sock_family) ✓

s.sin_port = htons (Port)

↓

(unsigned short)

host to network short converts integer to network byte order.

inet_aton (<ip> , & s.sin_addr)

↓

Converts IPV4 to Network byte order or Binary form.

↓

Returns non zero if <ip> is valid else not.

Allowed forms of <ip>

a.b.c.d	[each 8 bit]
a.b.c	[8, 8, 16]
a.b	[8, 24]
a	[32]

(3) Connect call :-

Connect (int sockfd, struct sockaddr *addr, socklen_t addrlen)

(Connects sockfd to address specified by addr).

addrlen = size of (addr)

on success → returns 0

on failure → returns -1

④ bind is similar to connect call but is used for servers

It assigns a address specified by the ~~sockaddr~~ addr to sockfd.

⑤ listen

listen (int sockfd, int n)

on success ← returns 0

on failure ← return -1

n is max. ~~size~~ length of the queue of pending connections with sockfd.

(more useful in case of iterative server)

⑥ accept

newsockfd = accept (~~cli~~ sockfd, (struct sockaddr *) & cli_addr, & clilen)

clilen = sizeof (cli_addr)

~~accept~~ on success → newsockfd

on failure → -1

accept takes the first connect request from the queue and creates a new connected socket.

sockfd is not affected by this call.

if socket is nonblocking and queue is empty then accept() fails with EAGAIN or EWOULDBLOCK

accept 4 → involves flags
0 → same as accept

SOCK_NONBLOCK → so makes sockfd nonblocking

SOCK_CLOEXEC

(7) Send / Sendto

ssize_t Send (sockfd, const void* , size, flags)

↓

accepts string, ints (any thing)

ssize-t Sendto (Sockfd, const void *, size, flags,
const struct sockaddr* addr,
socklen-t addrlen)

send ≡ sendto (- - - - , NULL, 0)

Send is used in case of connected state

Send is same as write (with zero flag)

After connection Send is same as sendto and the two remaining arguments are ignored.

if sockfd is not properly connected it returns ENOTCONN error

And if msg is too long to be parsed then it returns EMSGSIZE error and msg is not transmitted.

flags: 1) MSG-CONFIRM → if ACK is required (for SOCK-DGRAM)

2) MSG-DONTWAIT → makes send, sendto call non blocking

~~(8) recv / recvfrom~~

3) MSG - DONTROUTE x

4) MSG - EOR (for SOCK - SEQPACKET types)

5) MSG - NOSIGNAL (Doesn't generate SIGPIPE)

6) MSG - OOB (for Out of band data on sockets)

(8) recv / recvfrom (

Same as that of send / sendto (- - - - , * addr, (Not const) * socklen)

recv & read are same being the flag = 0

if connected then recv & recvfrom are same (- - - - , NULL, NULL)

returns the length of the msg received.

flags :- 1) MSG - DONTWAIT (Same as send / sendto)

2) MSG - PEEK (Get the data without clearing the buffer)

3) MSG - WAITALL ((operations block until the request is satisfied)

(maybe until buffer is filled dont recv)

(Signals are involved)

→ Doesn't works for Datagram sockets (UDP)

(9) close() :

on success → returns 0

on error → returns -1

It frees the associated resources of the file and if already unlinked it deletes the file.

UDP sockets

memset (& serv_addr, 0, sizeof (serv_addr))

* No acknowledgement
* No retransmissions
* out of order
* message oriented. (TCP is byte oriented)
* connection less. (TCP is bi directional)

if sending is only involved, bind is not required.
for UDP

bind() skipping is possible in TCP also.

Poll

<sys/poll.h>

struct pollfd pfd[2];

1) fd (int) ← file descriptor
2) events (short)
3) revents (short)

events → bitmask value of the events specifying the applications interested.

can be 0.

revents → output (set by kernel)

int ret = poll (pfd, <pfd+len>, Timeout)
(in this case it is 2)

if ret < 0 Poll failed
ret = 0 time out
else anything can happen

* poll / select

```
struct pollfd fdset[2]

fdset[0].fd = tcpsockfd;

fdset[0].events = POLLIN;

fdset[1].fd = udpsockfd.

fdset[1].events = POLLIN;

ret = poll(fdset, 2, timeout); // blocking call
```

timeout → (int type in milliseconds)

```
if (ret < 0) => error

else if (ret == 0) => timeout

else if (ret > 0) => x of the sockfds have data.
                     (say x)
{
    if (fdset[0].revents == POLLIN)
        { read from tcpfd } → (accept)
    else { read from udpfd }
}
```

POLLIN ⟶ Data read (New data has arrived)

↓

New connection request has arrived.

POLLHUP → Disconnection request

↓

Connection broken.

POLLOUT ⟶ (write)

↓

Socket has enough buffer space to write data

POLLIN | POLLOUT → Read / Write

POLLERR → Some async error

POLLHUP → A side has shut down the connection

POLLPRI → Urgent data (SIGURG)

for making all calls as non-blocking :-

fcntl (sockfd, F_SETVAL, O_NONBLOCK)

To know the status

val = fcntl (sockfd, F_GETVAL, 0)

for including permissions use bitwise or.

fcntl (sockfd, F_SETVAL, val | O_NONBLOCK)