Today

- TCP/IP and the layers
- IP TCP and UDP
- sockets
- DNS

Circuits & Datagrams

Circuit switching

- connection setup/takedown
- ACK'd delivery
- timeout/retry
- sequenced
- no duplicates
- streams

Datagrams

- individually addressed
- lost, out of sequence, delayed
- duplicated
- messages

IP Addresses

```
           +--------+--------+--------+--------+
IP     A   |0 net(7)|   host (24)               |
           +--------+--------+--------+--------+
                          32 bits
           +--------+--------+--------+--------+
       B   |10   net (14)    |     host (16)   |
           +--------+--------+--------+--------+
           +--------+--------+--------+--------+
       C   |110   net (21)            |host(8) |
           +--------+--------+--------+--------+
           +--------+--------+--------+--------+
       D   |1110     multicast (28)            |
           +--------+--------+--------+--------+

IPv6    128 bits
```


IP addresses

- localhost 127.0.0.1
- broadcast 255.255.255.255
- subnet broadcast 192.12.68.255
- private A 10.*.*.*
- private B's 172.16-31.*.*
- private C's 192.168.0-254.*
- UT 128.169.*.*
- ORNL 128.219.*.*, 134.167.*.*, 192.31.96.*

protocol families

- Various ways of providing network services
- many proprietary
- TCP/IP, OSI, SNA, DECnet, IPX, Appletalk, XNS, ...
- NOT interoperable
- NOT programmed the same
- but they do about the same thing

TCP/IP

- ARPA + BSD '81
- defined by RFCs
- no RFC for API
- packaged with BSD UNIX (free)
- non-proprietary
- basis of Internet
- many vendors, many media
- new stuff – IETF

IP

IPv4 Internet Protocol (RFC791)

- connectionless (datagram)
- unreliable
- checksum on header only
- fragmentation/assembly based on interface MTU
- 32-bit address (src/dest)
- transport protocol field
- TTL (hop count)
- routing (network layer)

transport layer

- end-to-end services to application
- API (BSD sockets, XTI/TLI)
- flow control
- error recovery
- ICMP (not), UDP, TCP

ICMP

Internet Control Message Protocol (RFC792)

- arguably part of IP
- error and control
- flow control (hop-to-hop)

UDP

User Datagram Protocol (RFC768)

- connectionless (datagram)
- 16-bit port
- unreliable
- optional checksum

TCP

Transmission Control Protocol (RFC793)

- connection-oriented
- 16-bit port
- reliable
- timers, checksums, sequence numbers
- flow control (end-to-end), 16-bit window
- urgent data
- segmentation based on MSS

services & transport

TCP

smtp,telnet,ftp,X
finger,talk,rcp,rlogin
rsh,nntp,irc,http,lpr

UDP

tftp,ntp,snmp, audio/video
multicast/broadcast
(either) nfs,dns,rpc
(both) PVM

ICMP

ping, traceroute

client/servers

clients

- user activated
- connects to well-known address
- sends/receives data
- closes connection
- non-privileged
- concurrency provided by OS

servers

- activated by system
- runs forever (awaits requests)
- usually privileged
- worry about security
- handle multiple requests either **iteratively** or **concurrently**
- iterative servers for fast, single-response requests (e.g., time)
- concurrent servers usually fork() (e.g. telnetd) or asynchronous I/O (select()), or have multiple copies running (nfsd)

BSD sockets

- transport layer interface
- API, subroutine library
- no standards (BSD is de facto)
- supports multiple protocol families (TCP/IP, XNS, UNIX, OSI, IPv6, ATM)
- flexibility is paid for in complexity
- mixture of filling data structures and function calls
- supports I/O abstraction
- **full duplex**, can't read what you write

socket calls

socket() get a socket descriptor for given protocol family and type

bind() associate name (address/port, etc.) with a server (usually) socket

connect() client establishes a connection to a server

listen() connection-oriented server tells system its going to be passive.

accept() server accepts incoming connection request and creates a new socket

close() will try to deliver any unsent data

Now you can do **read(), write(), send(), recv** or connectionless **sendto(), recvfrom()**

socket()

int socket(family, type, protocol)

- returns a socket descriptor
- family: AF_UNIX, AF_INET, AF_NS, AF_INET6 (actually should be PF_UNIX etc.)
- type: SOCK_STREAM, SOCK_DGRAM, SOCK_RAW
- protocol: usually 0
- fails: bad args, no fd's/memory
- just sets up kernel structures

#include <sys/types.h>
#include <sys/socket.h>

socket structures

```
/* sys/socket.h */
struct sockaddr {
        u_short sa_family;   /* address family */
        char    sa_data[14]; /* up to 14 bytes of direct address */
}

/* sys/un.h */
struct  sockaddr_un {
        short   sun_family;   /* AF_UNIX */
        char    sun_path[108];  /* path name (gag) */
};

/* netinet/in.h */
struct sockaddr_in {
        short   sin_family;  /* AF_INET*/
        u_short sin_port;     /* network byte order */
        struct  in_addr sin_addr;     /* network byte order */
        char    sin_zero[8];
};
```


connect()

connect(sockfd,struct sockaddr *server,lth)

- client fills struct sockaddr with server address/port
- client connects to server
- optional for UDP client, but handy for I/O model and to detect "connection refused" (at read/write time)
- can block
- fails: bad args, timeout, refused, unreachable, already connected
- if fails, must close()/socket()/connect()

data transfer

read(), write()
send(sockfd, const void *buff,lth,flags)
recv(sockfd, void *buff,lth,flags)

- only on connected sockets
- read()/write() (plus others) on streams will require looping to insure all data is read or written
- returns length read or written
- fails: EOF, reset, interrupted

readn.c

```
readn(int fd, void *ptr, int nbytes)
{
   int   nleft, nread;

   nleft = nbytes;
   while (nleft > 0) {
      nread = read(fd, ptr, nleft);
      if (nread < 0) {
 if (errno == EINTR)
   nread = 0; /* do read again */
         else return(nread);    /* error, return < 0 */
      }else if (nread == 0)
         break;         /* EOF */

      nleft -= nread;
      ptr   += nread;
   }
   return(nbytes - nleft);    /* return >= 0 */
}
```


writen.c

```
writen(int fd, const void *ptr, int nbytes)
{
   int   nleft, nwritten;

   nleft = nbytes;
   while (nleft > 0) {
      nwritten = write(fd, ptr, nleft);
      if (nwritten <= 0) {
 if (errno == EINTR)
   nwritten=0;  /* do it again */
         else return(nwritten);    /* error */

      }
      nleft -= nwritten;
      ptr   += nwritten;
   }
   return(nbytes - nleft);
}
```


readline.c

```
int
readline(int fd, char *ptr, int maxlen)
{
   int   n, rc;
   char  c;

   for (n = 1; n < maxlen; n++) {
     again:
      if ( (rc = read(fd, &c, 1)) == 1) {
         *ptr++ = c;
         if (c == '\n')
            break;
      } else if (rc == 0) {
         if (n == 1)
            return(0);  /* EOF, no data read */
         else
            break;      /* EOF, some data was read */
      } else {
 if (errno == EINTR) goto again;
         return(-1); /* error */
      }
   }

   *ptr = 0;
   return(n);
}
```



```
/*  tcpday ipaddress  simple tcp daytime client */
#include        <stdio.h>
#include        <sys/types.h>
#include        <sys/socket.h>
#include        <netinet/in.h>
void err_sys(char *msg) {perror(msg); exit(1);}

#define PORT 13
char *host = "127.0.0.1";    /* localhost */
#define MAXBUF 128
main(argc, char *argv[])
{
   int sd, n;
   struct sockaddr_in sin;
   char buff[MAXBUF + 1];

   if (argc > 1) host = argv[1];

   sd = socket(AF_INET,SOCK_STREAM,0);
   bzero(&sin,sizeof(sin));
   sin.sin_family = AF_INET;
   if ((sin.sin_addr.s_addr = inet_addr(host)) == -1 )
      err_sys("inet_addr");
   sin.sin_port = htons(PORT);  /* net byte order*/
   if (connect(sd, (struct sockaddr *)&sin, sizeof(sin)) < 0)
     err_sys("connect");
   while ((n = read(sd,buff,MAXBUF))>0){
           buff[n]=0;
           printf("%s",buff);
   }
}
```

```
nslookup cetus1a
Server:  TONKA.CS.UTK.EDU
Address:  128.169.94.60

Name:    cetus1a.cs.utk.edu
Address:  128.169.94.21

   in ~dunigan/cs494-unp
tcpday 128.169.94.21
Wed Sep  1 20:39:40 1999
    on SunOS (NOT solaris/truss)
trace tcpday 128.169.94.21
....
socket (2, 1, 0) = 3
connect (3, "".., 16) = 0
read (3, "Wed Sep  1 20:39:52 1999\n\r", 128) = 26
ioctl (1, 0x40125401, 0xefffef8c) = 0
write (1, "Wed Sep  1 20:39:52 1999\n\r", 26) = 26
Wed Sep  1 20:39:52 1999
read (3, "", 128) = 0
....
```


TCP/UDP ports

- 16-bit port number selects service/process on a machine
- "well-known" port numbers (see IANA), mail = 25, telnet = 23
- reserved 1-1023, privileged (/etc/services)
- transient (ephemeral) ports assigned by system (1024-5000)
- your choice above 5000
- TCP port is not same as UDP port
- server.sin_port = htons(7777);
- if port is 0, system will assign

/etc/services

```
...
ftp-data        20/tcp
ftp             21/tcp
telnet          23/tcp
smtp            25/tcp          mail
time            37/tcp          timserver
```


byte order & operations

- byte order differs between architectures
- for interoperability, network byte-order is required for info in protocol headers
- **htonl(), htons(), ntohl(), ntohs()**

```
#include <netinet/in.h>
```

byte operations

```
bcopy(void *src, void* dest, bytes)
bzero(void *buff, bytes)
bcmp(void *s1, void *s2, bytes)
```

for Sys V, memcpy(dst,src,n), memset(), memcmp()

address conversion

convert internet addresses to/from ASCII

```
#include <arpa/inet.h>

char *inet_ntoa(struct in_addr in)
int inet_aton(const char *s, struct in_addr *a)
in_addr_t inet_addr(char *string)
```

- *inet_ntoa()* not re-entrant
- *inet_aton()* replaced *inet_addr()* because -1 is legit (255.255.255.255)
- *inet_aton()* not on CS suns
- address in network byte order

```
   if ((sin.sin_addr.s_addr = inet_addr(host)) == -1 )
 err_sys("inet_addr");
becomes
   if (inet_aton(host,&sin.sin_addr)==0)
 err_sys("inet_aton");
```

addressing

- network addresses assigned by IANA
- host addresses assigned by local admin or DHCP
- mapping from host name to address via /etc/hosts, NIS/YP, or Domain Name System (DNS), gethostbyname()

```
/etc/hosts
127.0.0.1       localhost
128.169.94.1    cs.utk.edu utkcs cs
128.169.94.21   cetus1a.cs.utk.edu cetus1a
...
```


DNS

Domain Name System (RFC1034)

- host tables too big (1984)
- hierarchical, distributed (duncan.cs.utk.edu)
- /etc/resolv.conf defines local servers
- gethostbyname() sends UDP query packet to local server(s) and awaits reply
- **named** is the DNS daemon
- **named** has pointers to root servers
- **named** maintains cache
- supports reverse mapping (gethostbyaddr())

this can be slow – careful

getby and friends

names to numbers (return address of structure)

```
#include <netdb.h>
gethostbyname(char *name)
gethostbyaddr(char *addr, len, type)
getservbyname(char *name, char *proto)
getprotobyname(char *name)
```

others

```
getpeername(sockfd, struct sockaddr *peer, int *len)
getsockname(sockfd, struct sockaddr *local, int *len)
```


netdb.h

```
struct   hostent {
   char  *h_name; /* official name of host */
   char  **h_aliases;   /* alias list */
   int   h_addrtype; /* host address type */
   int   h_length;   /* length of address */
   char  **h_addr_list; /* list of addresses from name server */
#define  h_addr   h_addr_list[0] /* address, for backward compatibli
};

struct   servent {
   char  *s_name; /* official service name */
   char  **s_aliases;   /* alias list */
   int   s_port;     /* port # */
   char  *s_proto;   /* protocol to use */
}

struct   protoent {
   char  *p_name; /* official protocol name */
   char  **p_aliases;   /* alias list */
   int   p_proto; /* protocol # */
};
```

example

```
char *host;  /* name or adddress */
struct servent  *pse;
struct hostent  *phe;
struct sockaddr_in sin;

bzero((char *)&sin, sizeof(sin));
sin.sin_family = AF_INET;
if ( pse = getservbyname("ftp","tcp"))
  sin.sin_port = pse->s_port;
else err_sys("getserv");
if ( (sin.sin_addr.s_addr = inet_addr(host))== -1){
  if ( phe = gethostbyname(host) )
   bcopy(phe->h_addr, (char *)&sin.sin_addr, phe->h_length);
  else err_sys("badhost");
}
```


myhostent.c

```
#include <stdio.h>
#include <sys/types.h>
#include <netdb.h>      /* for struct hostent */
#include <sys/socket.h>    /* for AF_INET */
#include <netinet/in.h>    /* for struct in_addr */
#include <arpa/inet.h>     /* for inet_ntoa() */

main(argc, argv)
int   argc;
char  **argv;
{
   register char     *ptr;
   register struct hostent *hostptr;
   u_long in_addr;

   while (--argc > 0) {
      ptr = *++argv;
      if (*ptr >= '0' && *ptr <= '9') {
         in_addr = inet_addr(ptr);
         hostptr= gethostbyaddr(&in_addr,
              sizeof(struct in_addr), AF_INET);
      } else hostptr = gethostbyname(ptr);
      if ( hostptr  == NULL) {
         printf("gethostby failed for %s\n",ptr);
         continue;
      }
      printf("official host name: %s\n", hostptr->h_name);
```



```
      /* go through the list of aliases */
      while ( (ptr = *(hostptr->h_aliases)) != NULL) {
         printf(" alias: %s\n", ptr);
         hostptr->h_aliases++;
      }
      printf(" addr type = %d, addr length = %d\n",
            hostptr->h_addrtype, hostptr->h_length);

      switch (hostptr->h_addrtype) {
      case AF_INET:
         pr_inet(hostptr->h_addr_list, hostptr->h_length);
         break;

      default:
         err_ret("unknown address type");
         break;
      }
   }
}
```



```
/*
 * Go through a list of Internet addresses,
 * printing each one in dotted-decimal notation.
 */

pr_inet(listptr, length)
char  **listptr;
int   length;
{
   struct in_addr *ptr;

   while ( (ptr = (struct in_addr *) *listptr++) != NULL)
      printf(" Internet address: %s\n", inet_ntoa(*ptr));
}

-----
  myhostent duncan.cs.utk.edu
        official host name: duncan.cs.utk.edu
        addr type = 2, addr length = 4
        Internet address: 128.169.201.83
  myhostent 128.219.8.19
        official host name: max.epm.ornl.gov
        alias: max
        alias: maximus
        alias: sun2
        addr type = 2, addr length = 4
        Internet address: 128.219.8.19
  myhostent xxx
   gethostby failed for xxx
```

measuring elapsed time

```
double seconds()
{
#include <sys/time.h>
   struct timeval ru;
   gettimeofday(&ru, (struct timezone *)0);
   return(ru.tv_sec + ((double)ru.tv_usec)/1000000);
}

.....
  start = seconds();
   ....
  end = seconds();
```


tools

commands

- nslookup
- strace/truss/trace

library

- readn()/writen()
- seconds()
- sys_err()
- getaddr()

understand possible failures

next time

TCP server