Today

- name resolution
- TCP server
- bumps in the net
- socket options

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)
gethostname(char *name, len)
uname(struct utsname *name)
```

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 compat
};

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
```

gethostbyname()

- remember this can be slow
- may use hosts file, NIS, and/or DNS
- some later versions will take an IP address
- does not set **errno**, rather **h_errno**
- use *herror(char *msg)* or *const char * hstrerror(int err)*
- failures: no such host, no address

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()**

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
- use *readn() writen()*
- returns length read or written
- fails: EOF, reset, interrupted

```
/*  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);
   }
}
```

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)

bind()

int bind(sockfd, struct sockaddr *local,lth)

- binds local address to sockfd
- user fills struct sockaddr first
- required for server
- optional for client (except if AF_UNIX is expecting a reply)
- system will supply local address if client doesn't do bind
- lth of structure is required since struct sockaddr is different size for each protocol
- failures: bad args, port in use

listen()

listen(sockfd, backlog)

- server call after bind() before accept()
- specify length of connection request queue
- connection requests awaiting accept
- data queued
- if queue is full, requests ignored
- BSDs multiply backlog by 1.5
- backlog is limited (silently) to 5 on SunOS
- don't use backlog of 0
- SYN flooding attack
- failures: bad args

accept()

accept(sockfd,struct sockaddr *peer, int *lth)

- server accepts a connection request
- server sets max lth
- function returns NEW socket descriptor
- address info on peer is placed in *peer
- blocks
- failures: bad args, no mem

TCP server

```
/* daysrv [port] */
#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 7654
#define BUFFSIZE        128
#define BACKLOG 5
char buff[BUFFSIZE];

main(argc, argv)
int     argc;
char    *argv[];
{
    int port = PORT;
    int reap();
    int n,sockfd, newsockfd, clilen;
    struct sockaddr_in serv_addr, cli_addr;
```

```
if (argc > 1) port = atoi(argv[1]);
if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        err_sys("server: can't open stream socket");
/* setup struct for bind, so clients can find us */
bzero((char *) &serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
serv_addr.sin_port        = htons(port);

if (bind(sockfd, (struct sockaddr *) &serv_addr,
   sizeof(serv_addr)) < 0)
  err_sys("server: can't bind local address");

listen(sockfd, BACKLOG);
printf("server ready on port %d\n",port);
```

```
   for(;;){
       int ticks;

       clilen = sizeof(cli_addr);
       newsockfd = accept(sockfd,
        (struct sockaddr *) &cli_addr, &clilen);
       ticks=time(0);
       strncpy(buff,ctime(&ticks),sizeof(buff));
       write(newsockfd,buff,strlen(buff));
       close(newsockfd);
   }

}
```

getsockname

can let system assign port

```
serv_addr.sin_port        = htons(port);

if (bind(sockfd, (struct sockaddr *) &serv_addr,
   sizeof(serv_addr)) < 0)
  err_sys("server: can't bind local address");
```

becomes

```
serv_addr.sin_port = 0;

if (bind(sockfd, (struct sockaddr *) &serv_addr,
   sizeof(serv_addr)) < 0)
  err_sys("server: can't bind local address");
 clilen = sizeof(cli_addr);
 getsockname(sockfd,
     (struct sockaddr *) &cli_addr, &clilen);
 port = ntohs(cli_addr.sin_port);
```

server trace

trace daysrv

```
...
socket (2, 1, 0) = 3
bind (3, "".., 16) = 0
listen (3, 5) = 0
ioctl (1, 0x40125401, 0xf7ffee8c) = 0
write (1, "server ready on port 7654\n", 26) = server ready on p
26
accept (3, 0xf7fffa48, 0xf7fffa64) =

accept (3, 0xf7fffa48, 0xf7fffa64) = 4
gettimeofday (0xf7fff9c0, 0) = 0
write (4, "Sun Sep  5 14:04:24 1999\n", 25) = 25
close (4) = 0
accept (3, 0xf7fffa48, 0xf7fffa64) =
```

server status

checking server status

```
netstat -a
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp        0      0  *.7654                 *.*                    LISTEN
...
```

other states: ESTABLISHED CLOSE_WAIT

lsof to find what process has what port

tcpcli.c

```
main(argc, argv)
int   argc;
char  *argv[];
{
   bzero((char *) &serv_addr, sizeof(serv_addr));
   serv_addr.sin_family      = AF_INET;
   serv_addr.sin_addr.s_addr = inet_addr(SERV_HOST_ADDR);
   serv_addr.sin_port        = htons(SERV_TCP_PORT);
   if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
      err_sys("client: can't open stream socket");
   if (connect(sockfd, (struct sockaddr *) &serv_addr,
                  sizeof(serv_addr)) < 0)
      err_sys("client: can't connect to server");

   str_cli(stdin, sockfd);    /* do it all */

   close(sockfd);
   exit(0);
}
```

strcli.c

```
str_cli(fp, sockfd)
register FILE  *fp;
register int   sockfd;
{
   int   n;
   char  sendline[MAXLINE], recvline[MAXLINE + 1];

   while (fgets(sendline, MAXLINE, fp) != NULL) {
      n = strlen(sendline);
      if (writen(sockfd, sendline, n) != n)
         err_sys("str_cli: writen error on socket");
      n = readline(sockfd, recvline, MAXLINE);
      if (n < 0)
         err_sys("str_cli: readline error");
      recvline[n] = 0;  /* null terminate */
      fputs(recvline, stdout);
   }

   if (ferror(fp))
      err_sys("str_cli: error reading file");
}
```

tcpserv.c

```
main(argc, argv)
int   argc;
char  *argv[];
{
   int         sockfd, newsockfd, clilen, childpid;
   struct sockaddr_in   cli_addr, serv_addr;
   if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
      err_sys("server: can't open stream socket");
   bzero((char *) &serv_addr, sizeof(serv_addr));
   serv_addr.sin_family      = AF_INET;
   serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
   serv_addr.sin_port        = htons(SERV_TCP_PORT);

   if (bind(sockfd, (struct sockaddr *) &serv_addr,
        sizeof(serv_addr)) < 0)
      err_sys("server: can't bind local address");

   listen(sockfd, 5);
```

```
   for ( ; ; ) { /* iterative server */
      clilen = sizeof(cli_addr);
      newsockfd = accept(sockfd,
 (struct sockaddr *) &cli_addr, &clilen);
      if (newsockfd < 0)
         err_sys("server: accept error");
      str_echo(newsockfd); /* process the request */
      close(newsockfd);    /* parent process */
   }
}
```

strecho.c

```
#define  MAXLINE  512

str_echo(sockfd)
int   sockfd;
{
   int   n;
   char  line[MAXLINE];

   for ( ; ; ) {
      n = readline(sockfd, line, MAXLINE);
      if (n == 0)
         return;     /* connection terminated */
      else if (n < 0)
         err_sys("str_echo: readline error");

      if (writen(sockfd, line, n) != n)
         err_sys("str_echo: writen error");
   }
}
```

commands args

client host port

```
/* handle hostname or host address (d.d.d.d) */
#include <netdb.h>
main(argc,argv)
char *argv[];
{
   char *host = "localhost";
   int port = DEF_PORT;
   struct hostent  *phe;
   struct sockaddr_in sin;

   if (argc > 1) host = argv[1];
   if (argc > 2) port = atoi(argv[2]);
   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 errexit("can't get \"%s\" host entry\n", host);
   }
...
```

What if?

things that go bump in the net

1. unknown hostname
2. bad IP address
3. TCP connect, and no server process
4. TCP connect, server host down
5. active TCP session, ctrl-c server
6. inactive TCP session, ctrl-c server
7. active TCP session, server computer crashes
8. inactive TCP session, server computer crashes
9. inactive TCP session with KEEPALIVE, server computer crashes
10. inactive TCP session, server computer crashes and reboots
11. start 2nd copy of server
12. server tries to bind to port < 1024
13. A sends faster than B can receive

socket options

- modify socket characteristics
- setsockopt(), getsockopt()
- must refer to open sockfd, before connect/bind

```
#include <sys/types.h>
#include <sys/socket.h>
getsockopt(fd, level, optname, void *val, int *len)
setsockopt(fd, level, optname, void *val, int len)
```

level specifies general or protocol type.

socket options

- **SO_RCVBUF SO_SNDBUF** specifies the size of the send/recv buffers (windows) for TCP, or the UDP datagram size
- **SO_DEBUG** enable kernel tracing of a TCP connection. use **trpt** to display trace.
- **SO_KEEPALIVE** enable periodic transmissions on a TCP connection to verify both ends are alive.
- **SO_LINGER** permit abortive close. Unsent data is discarded.
- **SO_REUSEADDR** permit concurrent use of local port
- **TCP_NODELAY** (need netinet/tcp.h and netinet/in.h for level of IPPROTO_TCP), causes small packets to be sent immediately.

examples sockopt.c checkopt.c in book source

sockopt.c

```
#include <sys/types.h>
#include <sys/socket.h>    /* for SOL_SOCKET and SO_xx values */
#include <netinet/in.h>    /* for IPPROTO_TCP value */
#include <netinet/tcp.h>      /* for TCP_MAXSEG value */

main()
{
   int   sockfd, maxseg, sendbuff, optlen;
   if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
      err_sys("can't create socket");
   optlen = sizeof(maxseg);
   if (getsockopt(sockfd, IPPROTO_TCP, TCP_MAXSEG,
       &maxseg, &optlen) < 0)
      err_sys("TCP_MAXSEG getsockopt error");
   printf("TCP maxseg = %d\n", maxseg);
   if (getsockopt(sockfd, SOL_SOCKET, SO_SNDBUF,
      &sendbuff, &optlen) < 0)
      err_sys("SO_SNDBUF getsockopt error");
   printf("send buffer size = %d\n", sendbuff);
   sendbuff = 16384; /* just some number for example purposes */
   if (setsockopt(sockfd, SOL_SOCKET, SO_SNDBUF,
       &sendbuff, sizeof(sendbuff)) < 0)
      err_sys("SO_SNDBUF setsockopt error");
   optlen = sizeof(sendbuff);
   if (getsockopt(sockfd, SOL_SOCKET, SO_SNDBUF,
       &sendbuff, &optlen) < 0)
      err_sys("SO_SNDBUF getsockopt error");
   printf("send buffer size = %d\n", sendbuff);
}
```

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/netstat/lsof
- strace/truss/trace

library

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

understand possible failures

next time

UDP